DISSERTATION

ENHANCING COLLABORATIVE PEER-TO-PEER SYSTEMS USING RESOURCE AGGREGATION AND CACHING: A MULTI-ATTRIBUTE RESOURCE AND QUERY AWARE APPROACH

Submitted by

H. M. N. Dilum Bandara

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Fall 2012

Doctoral Committee:

Advisor: Anura P. Jayasumana

V. Chandrasekar
Daniel F. Massey
Indrajit Ray

ABSTRACT

ENHANCING COLLABORATIVE PEER-TO-PEER SYSTEMS USING RESOURCE AGGREGATION AND CACHING: A MULTI-ATTRIBUTE RESOURCE AND QUERY AWARE APPROACH

Resource-rich computing devices, decreasing communication costs, and Web 2.0 technologies are fundamentally changing the way distributed applications communicate and collaborate. With these changes, we envision Peer-to-Peer (P2P) systems that will allow for the integration and collaboration of peers with diverse capabilities to a virtual community thereby empowering it to engage in greater tasks beyond what can be accomplished by individual peers, yet are beneficial to all the peers. Collaborations involving application-specific resources and dynamic quality of service goals will stress current P2P architectures that are designed for best-effort environments with pair-wise interactions among nodes with similar resources. These systems will share a variety of resources such as processor cycles, storage capacity, network bandwidth, sensors/actuators, services, middleware, scientific algorithms, and data. However, very little is known about the specific characteristics of real-world resources and queries as well as their impact on resource aggregation in these collaborative P2P systems. We developed resource discovery, caching, and distributed data fusion solutions that are more suitable for collaborative P2P systems while characterizing real-world resource, query, and user behavior. The contributions of this research are: (1) a detailed analysis of real-world resource, query, and user characteristics and their impact on resource discovery solutions, (2) a tool to generate large synthetic traces of multi-attribute resources and range queries, (3) resource and query aware P2P-based multi-attribute resource discovery solution that is both efficient and load balanced, (4) a community-based caching solution that enhances both the communitywide and system-wide lookup performance in large-scale P2P systems, and (5) demonstrated the applicability of NDN (Named Data Networking) for DCAS (Distributed Collaborative Adaptive Sensing) systems by developing a distributed multi-user, multi-application, and multi-sensor data fusion solution based on

CASA (Collaborative Adaptive Sensing of the Atmosphere). Proposed solutions and the analysis are applicable to a wide variety of contexts such as DCAS systems, P2P clouds, grid/cloud computing, GENI (Global Environment for Network Innovations), and mobile social networks. Next, each of the five contributions is described briefly.

First, we derived an equation to capture the cost of multi-attribute resource advertising and querying. The nature of parameters in the equation under different systems is determined by analyzing datasets from PlanetLab, SETI@home, EGI grid, and a distributed campus computing facility. These datasets exhibit several noteworthy features that affect the performance. The attributes of both the resources and queries are highly skewed and correlated. Attribute values have different marginal distributions and change at different rates. Queries are less specific where each query tends to specify only a small subset of the available attributes and large ranges of attribute values. These properties of resources and queries are then used to qualitatively and quantitatively evaluate the fundamental design choices for P2P-based multi-attribute resource discovery. Design choices are evaluated based on the cost of advertising/querying, load balancing, and routing table size. Compared to uniform queries, real-world queries are relatively easier to resolve using unstructured, superpeer, and single-attribute-dominated-query-based structured P2P solutions. However, they introduce significant load balancing issues to existing designs. Cost of RD in structured P2P systems is effectively $O(N)$ (N is the number of nodes) as most range queries are less specific.

Second, a set of mechanisms is presented to generate realistic synthetic traces of multi-attribute resources (with both static and dynamic attributes) and range queries using the statistical behavior learned from real-world datasets. Such traces are useful in large-scale performance studies of resource discovery solutions, job schedulers, etc., in collaborative P2P systems as well as grid, cloud, and volunteer computing. Random vectors of static attributes are generated using empirical copulas that capture the entire dependence structure of multivariate distribution of attributes. Time series of dynamic attributes are randomly drawn from a library of multivariate time-series segments extracted from the datasets. These segments are identified by detecting the structural changes in time series corresponding to a selected attribute. Time series corresponding to rest of the attributes are split at the same breakpoints and randomly

drawn together to preserve their contemporaneous correlation. Correlation among static and dynamic attributes is preserved by grouping the time-series segments based on their static attributes. Multi-attribute range queries are generated using a probabilistic finite state machine that preserves the popularity of attributes and correlations among attribute values. A tool is developed to automate the synthetic data generation process. It is independent of the dataset hence data from any other platform may be used as the basis for trace statistics.

Third, a resource and query aware P2P-based multi-attribute resource discovery solution is presented that is both efficient and load balanced. The solution consists of five heuristics that can be executed independently and distributedly. The first heuristic tries to maintain a minimum number of nodes in the overlay while pruning nodes that do not significantly contribute to the range-query resolution. Removing nonproductive nodes reduces the cost (e.g., hops and latency) of advertising resources and resolving queries. The second and third heuristics dynamically balance the key and query load distributions by transferring keys to neighbors as well as by adding new neighbors when existing ones are insufficient. The last two heuristics, namely fragmentation and replication, form cliques of nodes that are placed orthogonal to the overlay ring. Such a node placement dynamically balances the highly skewed key and query loads while reducing the query cost. By applying these heuristics in the presented order, a resource discovery solution that better responds to real-world resource and query characteristics is developed. Efficacy of the solution is demonstrated using a simulation-based analysis under a variety of single and multi-attribute resource and query distributions derived from real workloads.

Fourth, we developed a distributed caching solution that exploits P2P communities to improve the communitywide and system-wide lookup performance. The solution consists of a sub-overlay formation scheme and a Local-Knowledge-based Distributed Caching (LKDC) algorithm. Sub-overlays enable communities to forward queries through their members. While queries are forwarded, LKDC algorithm causes members to identify and cache resources of interests to them, resulting in faster resolution of queries for popular resources within each community. Distributed local caching requires global information (e.g., hop count and popularity of contents) that is difficult and costly to obtain. Moreover, the problem is

NP-complete when the size of contents/resources varies. However, by relaxing the content size constraint (which is acceptable for the purpose of improving lookup performance), and by means of an analysis of globally optimal behavior and structural properties of the overlay, we developed the LKDC algorithm that not only relies on purely local information but also provides close-to-optimal caching performance. The caching solution automatically adapts to changing popularity and user interests. It works with any skewed distribution of queries in addition to introducing minimal modifications and overhead to the overlay network. For example, simulations based on Chord overlay show a 40% reduction in average path length using only 20 cache entries per node, and individual communities gained a 17-24% improvement compared to system-wide caching.

Fifth, we present a proof of concept solution that demonstrates the applicability of NDN for multi-user, multi-application, and multi-sensor DCAS systems such as CASA. In this example, a network of weather radars name data based on their geographic location and weather feature (e.g., reflectivity of clouds or wind velocity) independent of the radar(s) that generated them. This enables end users to specify an area of interest for a particular weather feature while being oblivious to the placement of radars and associated computing facilities. Conversely, the DCAS system can use its knowledge about the underlying system to decide the best radar scanning and data processing strategies. Such sensor-independent names also enhance resilience, enable processing data close to the source, and benefit from NDN features such as in-network caching and duplicate query suppression consequently reducing the bandwidth requirements of the DCAS system. We also present mechanisms to support sensor-specific and event-specific names that are also important in DCAS systems. The solution is implemented as an overlaid NDN enabling the benefits of both the NDN and overlay networks. Simulation-based analysis using reflectivity data from an actual weather event showed 87% reduction in average bandwidth consumption of radars and 95% reduction in average query resolution latency.

ACKNOWLEDGMENTS

I would like to acknowledge and extend my heartfelt gratitude to all of those who supported and encouraged me in any aspect that has made this dissertation possible.

I am heartily thankful to my supervisor, Prof. Anura P. Jayasumana, for believing in my abilities, inspiration, in-depth critique of the research, and patience with my occasional ignorance. Initially, it was extremely challenging to find a new research problem. While the middle phase was technically and theoretically challenging, the final phase was even more challenging due to an inability to properly focus. Prof. Jayasumana taught me how to find new research problems by discovering variations, generalizations, and observing emerging trends, to do fundamental research, as well as to keep going while realizing research has both good and bad days. His encouragement, guidance, and support in every aspect of the dissertation and beyond helped me conquer all these challenges while making it the most rewarding experience of my life.

Special thanks are also due to members of my graduate committee, Prof. V. Chandrasekar, Dr. Daniel F. Massey, and Dr. Indrajit Ray for valuable suggestions and criticisms that drove me to give my best throughout this research. Extensive support from Dr. Michael Zink (UMass), Dr. Panho Lee, and Dr. Sanghun Lim was immensely helpful.

I am grateful to the Engineering Research Center (ERC) for Collaborative Adaptive Sensing of the Atmosphere (CASA) for supporting part of my research under the ERC program of the National Science Foundation (award number 0313747). CASA gave me once in a lifetime opportunity to be part of an interdisciplinary and intercollegiate team that defines the state-of-the-art in weather research that has a direct and greater impact to the society. Support from Dr. Paula S. Reese (UMass), and all the research and administrative staff of CASA was extremely helpful.

I would like to thank Veeresh Rudrappa and Sudharshan Varadarajan for helping with data collection and processing which have been immensely useful in my research. I would also like to thank Dr. Jeannie Albrecht, Dr. Vivek Pai, and Grid Observatory team for providing the SWORD, CoMon, and grid

datasets. System administrators from the Engineering Network Services (ENS) and the Dept. of Computer Science were also helpful in collecting data within the campus. I am grateful to Vidarshana, Saket, Dulanjalie, Pritam, and many other colleagues at the Computer Networking Research Laboratory (CNRL), for their invaluable feedback, suggestions, support, and enduring hours and hours of presentations. I also would like to recognize the peers and every individual that helped me in any respect and may not have been mentioned above.

Finally yet importantly, I am indebted to my parents and my loving wife for their unconditional love, continuous support, and encouragement. I thank my son Manula, who gives me much joy and strength. This dissertation is dedicated to them.

To my parents, my loving wife Sudeshini, and son Manula

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Decreasing communication costs and Web 2.0 technologies are fundamentally changing the way we communicate, learn, socialize, and collaborate to create a better world while propelling us to a new era of societal development [Az09]. Peer-to-Peer (P2P) computing is a natural fit to this new era because it is user driven, autonomous, distributed, and utilizes resource-rich edge devices, as well as encourages sharing and collaboration. P2P systems have tremendous scalability and are applicable in a wide variety of application domains such as file sharing, VoIP (Voice over Internet Protocol), IPTV (Internet Protocol Television), content delivery, distributed processing, multi-player gaming, and social networks. These systems currently have a user base of several hundreds of millions. They contributed to more than 4.6 Exabytes of Internet traffic per month in 2011 [Ci12].

Section 1.1 presents the motivation. Summary of the contributions is presented in Section 1.2. Section 1.3 presents the outline of the dissertation.

1.1 Motivation

Resource-rich computing devices, decreasing communication costs, and Web 2.0 technologies are fundamentally changing the way distributed applications communicate and collaborate. With these changes, we envision P2P systems that play an even greater role in collaborative applications. Such collaborative applications provide tremendous opportunities to create value by combining the societal trends with P2P systems. Peer collaboration is expanding beyond its conventional applications wherein files or processor cycles are shared by peers to perform similar tasks. Future collaborative P2P applications will look for diverse peers that could bring in unique capabilities to a virtual community thereby empowering it to engage in greater tasks beyond what can be accomplished by individual peers, yet are beneficial to all the peers. This is similar to a modern team that thrives due to the diversity of members' expertise. Thus, a

collaborative P2P system is a P2P system that aggregates a group(s) of diverse resources (e.g., hardware, software, services, and data) to accomplish a greater task [Ba12b]. These systems will share a variety of *resources* such as processor cycles, storage capacity, network bandwidth, sensors/actuators, special hardware, middleware, scientific algorithms, application software, services (e.g., web services and spawning nodes in a cloud), and data to not only consume a variety of contents but also to generate, modify, and manage those contents. Such collaborations involving diverse and application-specific resources as well as dynamic Quality of Service (QoS) goals will stress the current P2P architectures.

Collaborative P2P systems are applicable in a wide variety of contexts such as Distributed Collaborative Adaptive Sensing (DCAS) [Ku06, Le12, Mc05, Mc09], grid [Ca04, Sh07], cloud [Ar09], and opportunistic [Co10] computing, Internet of Things [Pf11], social networks, and emergency management. To illustrate the salient features and characteristics of collaborative P2P systems we use four representative collaborative applications.

First is Collaborative Adaptive Sensing of the Atmosphere (CASA) [Ku06, Mc05, Mc09], a DCAS system based on a dense network of weather radars that collaborate and adapt in real time to detect hazardous atmospheric conditions such as tornados and severe storms. Collaborative P2P data fusion provides an attractive implementation choice for real-time radar data fusion in CASA [Le12], wherein multiple data volumes are constantly being generated, processed, and pushed and pulled among radars, storage, and processing nodes. Radars, processing, and storage elements involved in tracking a particular weather event may continue to change as the weather event migrates in both time and space. Thus, new groups of resources may have to be aggregated and current resources are released as and when needed. Moreover, certain rare but severe weather events require specific meteorological algorithms (e.g., signal processing and forecasting) as well as more computing, storage, and bandwidth resources to track and forecast/nowcast about the behavior of those events. It is neither feasible nor economical to provision resources for such rare peak demands everywhere on the CASA system. Instead, a collaborative P2P system can exploit the temporal and spatial diversity of weather events to aggregate underutilized resources from anywhere in the system as far as the desired performance and QoS goals are satisfied. For example, radar

data related to a tornado in Oklahoma can be timely delivered and processed in Texas, if underutilized resources are available in Texas due to calm weather. Therefore, a collaborative P2P system can satisfy such rapid and high resource demands while enhancing the overall resource utilization of CASA. Moreover, P2P architectures are robust under random node failures; therefore, provides a best-effort data fusion framework under hostile conditions. P2P architectures also reduce the risk of single point of failure. This is desirable in CASA-like systems that have to perform critical functions under hazardous weather conditions, which can potentially interfere with some of the system infrastructure. On the other end of the spectrum, we are also seeing the emergence of crowd sourced, community-based weather monitoring systems [Sl11] that aggregate armature weather stations and community-based computing resources to provide local/national weather forecasts. Collaborative P2P systems are a natural fit for such community driven resource collaborations among the users.

Second, cloud computing is transforming the way we host and run applications because of its rapid scalability and pay-as-you-go economic model [Ar09]. Open cloud initiatives are pressing for interoperability among multiple cloud providers and sites of the same provider. Moreover, its centralized data and proprietary application model contradicts with the Free and Open Source Software (FOSS) movement; hence, considered a threat by users who want to be in control of their data and applications [Ar09]. Community cloud computing [Br09], based on underutilized computing resources in homes/businesses, targets such issues including centralized data, privacy, proprietary applications, and cascading failures in modern clouds. Certain applications also benefit from a mixture of dedicated and voluntary cloud resources [An10, Fo09]. A collaborative P2P system is the core of such a multi-site or community-cloud system that interconnects dedicated/voluntary resources while dealing with rapid scalability and resource fluctuations. Such systems are referred to as *P2P clouds*.

The third application, Global Environment for Network Innovations (GENI), is a collaborative and exploratory platform for discoveries and innovation [El09]. GENI allows users to aggregate diverse resources (e.g., computing, networks, sensors, and software) from multiple administrative domains for a

common task. A collaborative P2P system meshes well with GENI because of its distributed, dynamic, heterogeneous, and collaborative nature.

Fourth, the value of social networks can be enhanced by allowing users to share diverse resources available in their mobile devices [Co10]. For example, a group sharing their holiday experiences in a coffee shop could use one of their members' projection phone to show pictures from others' mobiles or tablets, or stream videos from their home servers. Moreover, in large social gatherings such as carnivals, sports events, or political rallies, users' mobile devices can be used to share hot deals, comments, videos, or vote for a certain proposition without relying on a network infrastructure. Such applications are already emerging under the domain of opportunistic networking and computing [Co10]. These applications, hereafter referred to as *mobile social networks*, also benefit from collaborative mobile P2P technology.

We can further envision an agglomeration of many collaborative P2P systems into a single unified P2P framework wherein peers contribute and utilize diverse resources for both altruistic and commercial purposes. For example, a cloud provider could contribute its processor cycles to the P2P community hoping to gain monetary benefits whenever possible, and during periods of lower demand it could provide similar or degraded services to gain nonmonetary benefits (e.g., to demonstrate its high availability or gain reputation). Alternatively, an application provider that accesses free/unreliable resources for its regular operations could tap into dedicated/reliable resources during periods of high demand [An10]. Such a framework could also enable resource-rich home users to earn virtual currency for their contributions that they can later use to access other services offered within the system [Ka11, Me10]. Such a framework also enables a level playing field for both small-scale and large-scale contributors.

CASA, P2P clouds, GENI, mobile social networks, and aggregated P2P systems depend on some form of collaboration among resources. These complex resources are characterized by multiple static and dynamic attributes. For example, CPU speed, free CPU capacity, free memory, bandwidth, operating system, and a list of installed applications/middleware may characterize a processing node in CASA, GENI, grids, and clouds. These multi-attribute resources need to be combined in a timely manner to meet the performance and QoS requirements of collaborative P2P applications. Yet, it is nontrivial to discover,

aggregate, as well as utilize heterogeneous and dynamic resources that are distributed. Moreover, while some of the resources are volatile and voluntary (e.g., resources in P2P clouds) other resources are stable and dedicated (e.g., resources in CASA). What resources are shared and to what extent they are shared also depend on the behavior of peer communities that are formed according to semantic, geographic, and organizational interests of users [Ba11c, Ba12e]. Thus, discovering and combining an optimum set of resources is an extremely complex but fundamental requirement for collaborative P2P applications.

The overall process of advertising, discovering, and combining resources is referred to as *resource aggregation*. A good resource aggregation solution should efficiently advertise all the resources and their current state, discover potentially useful resources, select resources that satisfy application requirements, match resources and applications according to their constraints, as well as bind resources and applications to ensure guaranteed service. Several solutions have been proposed to advertise, discover, and select individual resources in a variety of P2P systems [Bh04, Ca04, Co09b, Kw10, Sh07, Ta08]. Even the solutions (e.g., [Al08] and [Ke06]) that provide some form of resource aggregation are primitive in capturing inter-resource relationships, and are therefore unable to put together the best group of resources [Ba12b]. They are not designed for latency sensitive collaborative P2P applications such as P2P clouds or mission critical applications such as CASA. Moreover, in the absence of data and understanding of the real-world resource and query characteristics, these systems relied on many simplifying assumptions. For example, independent and identically distributed (i.i.d.) attributes [Bh04, Co09b, Sh07], uniform/Zipf's distribution of attribute values [Bh04, Co09b, Sh07], attributes having a large number of potential values [Sh09], and queries specifying a large number of attributes and a small range of attribute values [Al08, Bh04, Ca04, Co09b, Sh07]. Such assumptions affect both the designs and performance analysis, and consequently the applicability of solutions under real workloads. Moreover, no single solution efficiently and scalably supports all the requirements of real-world collaborative P2P systems. Therefore, more efficient and load balanced, resource aggregation solutions are needed.

Internet users value the ability to access content irrespective of its location, whereas the Internet was designed to facilitate end-to-end resource access. Conflict between the usage and design objectives

has led to many issues such as location dependence, traffic aggregation, and security. Consequently, many clean-state designs for the Internet propose to access/route data based on their application-layer content names [Ja09a, Ko07, St02]. Named Data Networking (NDN) [Ja09a] (a.k.a. Content Centric Networking (CCN)) is gaining traction as one of the viable clean-state designs particularly in the presence of CCNx open source implementation [Palo]. NDN enables in-network caching, multicasting, duplicate message suppression, and enhanced security and mobility. When data are not already dispersed within the network, NDN delivers user queries to potential data sources enabling on-demand data generation. In contrast, the majority of other content-naming solutions, e.g., [Ko07, St02], are based on distributed hash tables that index only the pre-generated data. Moreover, NDN supports different levels of abstractions and incremental deployment ranging from overlay networks, content delivery networks, and small ISPs to eventual Internet-wide deployment.

DCAS systems, including current CASA deployments [Li07a], typically bind data to the sensor(s) that generated them by assigning data names based on the sensor identifier. Alternatively, end users in many cases are interested in data related to a particular event in a given area of interests, and are not concerned about which sensor(s) generated the data. Therefore, naming data based on the source/sensor creates a conflict similar to that in the current Internet. Consequently, it reduces the ability to utilize the spatial and temporal locality in user interests and redundant sensors in DCAS systems to enhance the performance of distributed sensing and data fusion. NDN enables DCAS systems to overcome these limitations while benefiting from reduced bandwidth requirements of the data fusion system, enhanced resource utilization, resilience, security, and mobility. For example, a network of CASA weather radars may name data based on their geographic location and weather feature (e.g., reflectivity of clouds or wind velocity) independent of the radar(s) that generated them. This enables end users to specify an area of interest for a particular weather feature while being oblivious to the placement of CASA radars and associated computing facilities. Such sensor-independent names also enable processing data close to the source. Currently, NDN has to be deployed as an overlay network due to the absence of an Internet-wide deployment. However, use of overlay networks provides the added benefits such as the ability to deploy multiple

and application-specific naming conventions, application-specific routing mechanisms, fault tolerance, better QoS, and in-network data fusion [Ba13, Le12]. Therefore, DCAS systems can be made more efficient and robust by combining the benefits of NDN and overlay networks.

1.2 Contributions

The goal of this research is to develop better resource discovery and distributed data fusion solutions and necessary tools that can aggregate groups of heterogeneous, dynamic, and multi-attribute resources in collaborative P2P systems, while characterizing real-world resources, queries, and user behavior. This dissertation spans three key areas of research related to multi-attribute Resource Discovery (RD), single-attribute RD and distributed caching, and multi-sensor data fusion leading to five major contributions. First, a detailed analysis and characterization of real-world resources, queries, and content access patterns of P2P users are presented. We then used the learned characteristics to qualitatively and quantitatively evaluate the fundamental design choices for P2P-based resource discovery. Second, a tool to generate realistic synthetic traces of multi-attribute resources and range queries for large-scale simulation studies is developed. Third, our findings were also used to develop a resource and query aware, P2P-based RD solution that is both efficient and scalable. Fourth, we developed a P2P community-aware distributed caching solution and demonstrated its applicability both analytically and empirically. Fifth, we demonstrated the applicability of NDN for DCAS systems by developing a proof-of-concept multi-user, multi-application, and multi-sensor data fusion solution based on CASA. Next, each of the five contributions is described briefly.

We developed an equation to capture the overall cost of RD in terms of overlay messages involved in advertising multi-attribute resources and querying them. The nature of parameters in the equation under different systems is determined by analyzing datasets from PlanetLab, SETI@home, EGI grid, and a distributed campus computing facility. PlanetLab data are also used to analyze the multi-attribute range query characteristics. A representative subset of design choices for P2P-based RD is then qualitatively and quantitatively evaluated using the learned characteristics. These design choices are evaluated

based on the cost of advertising and querying resources, routing table size, load balancing, and scalability. These datasets exhibited several noteworthy features that affect the performance. The attributes of both the resources and queries were highly skewed and correlated. While resources are characterized by many attributes, most attributes had only a few distinct set of values. Attribute values had different marginal distributions and change at different rates. Queries were less specific where they tended to specify only a small subset of the available attributes and large ranges of attribute values. Therefore, real-world resource and query characteristics diverge substantially from the conventional assumptions. Simulation-based analysis indicated real-world queries are relatively easier to resolve using unstructured, superpeer, and single-attribute-dominated-query-based structured P2P architectures compared to uniform queries used in conventional studies. Cost of RD in ring-based structured P2P systems was effectively $O(N)$, where N is the number of nodes in the overlay, as most queries specified large ranges of attribute values. The cost of advertising dynamic attributes was significant and increased with the number of attributes. Furthermore, all the design choices were prone to significant load balancing issues where few nodes were mainly involved in answering the majority of the queries and indexing resources. Therefore, existing design choices are applicable only under very specific conditions and perform poorly under realistic workloads.

To evaluate the applications and protocols for scalability beyond what is available, it becomes necessary to consider resource and query configurations with higher number of nodes and attributes. However, it is impractical to gather traces with sufficient resolution and duration even for existing systems. Therefore, we developed a mechanism to gather representative statistical information about the real-world traces and generate synthetic trace arrays of larger dimensionality in number and time. The presented mechanism generates realistic synthetic traces of multi-attribute resources (with both static and dynamic attributes) and range queries. Such traces are useful in evaluating the performance of large-scale RD solutions and job schedules. The presented methodology is applicable to any multivariate resource and query dataset. First, the vectors of static attributes are generated using empirical copulas that capture the entire dependence structure of multivariate distribution of attributes. Second, time series of dynamic attributes are randomly drawn from a library of multivariate-time-series segments extracted from datasets.

These segments are determined by identifying the structural changes in time series corresponding to a selected attribute. Time series corresponding to rest of the attributes are split at the same structural break-points and randomly drawn together to preserve their contemporaneous correlation. Third, multi-attribute range queries are generated using a probabilistic finite state machine. Furthermore, a tool is developed to automate the synthetic data generation process and its output is validated using statistical tests.

While taking into account the complex characteristics of real-world resources and queries, we then developed a resource and query aware, P2P-based RD solution. The solution is based on five heuristics that can be executed independently and distributedly on a ring-like overlay. The first heuristic tries to maintain only a small subset of the nodes in the overlay as domain of resource attributes tend to be much smaller than the number of nodes. It prunes nodes that do not significantly contribute to range query resolution while reducing the cost (e.g., hops and latency) of resolving queries. The second and third heuristics dynamically balance the key and query load distribution of nodes by transferring keys to neighbors and by adding new neighbors when existing ones are insufficient. The last two heuristic, namely fragmentation and replication, form cliques of nodes to dynamically balance the skewed key and query loads associated with few popular resources. In contrast to the common practice of replicating along the overlay ring, cliques of fragments and replicas are placed orthogonal to the ring thereby providing lower query cost and better load distribution. By applying these heuristics in the presented order, a RD solution that better responds to real-world resource and query characteristics is developed. Simulation-based analysis is used to evaluate the efficacy of the proposed solution under a variety of single and multi-attribute resource and query distributions derived from real workloads.

Large P2P systems exhibit the presence of virtual communities based on semantic, geographic, or organizational interests of users. Resources commonly shared within individual communities are in general relatively less popular and inconspicuous in the system-wide behavior. Hence, most communities are unable to benefit significantly from performance enhancement schemes such as caching and replication that focus only on the most dominant queries. We first analyzed the similarities among P2P communities using the search clouds of several BitTorrent search engines. The analysis confirmed that user interests in

different communities overlap to some degree. Second, a survey was conducted to identify the number of communities accessed by BitTorrent users and their frequencies. Our findings showed users prefer to access contents from a few primary communities where 89% of the time they accessed at most two communities. Based on these findings, we then developed a community-based proactive caching solution for structured P2P systems that can overcome the limitations in existing solutions. Our solution consists of a sub-overlay formation scheme and a Local-Knowledge-based Distributed Caching (LKDC) algorithm. First, we propose a method whereby sub-overlays are formed within the overlay network, enabling communities to forward queries through their members. While the queries are forwarded, LKDC algorithm causes the peers running it to identify and cache resources that are popular within their communities. Therefore, lookup queries for popular resources within a community are resolved faster. Consequently, both the community-level and the system-level lookup performance improve. Distributed local caching requires global information such as hop count and content popularity that are difficult and costly to obtain. Moreover, the problem is NP-complete when contents/resources have varying sizes. However, by relaxing the content size constraint (which is acceptable for the purpose of improving lookup performance), and by analyzing the globally optimal behavior and taking into account the structural properties of the overlay, we show it is still possible to develop a close-to-optimal caching solution (namely LKDC) that relies purely on local statistics. The proposed solution is independent of how the communities are formed and adaptive to changing popularity and user interests. It works with any skewed distribution of queries. Simulations show a 40% reduction in overall average path length with per node cache sizes as low as 20. Less popular communities are able to reduce the path length by three times compared to system-wide caching.

We present a proof of concept solution that demonstrates the applicability of NDN for multi-user, multi-application, and multi-sensor DCAS systems. For example, a network of CASA weather radars may name data based on their geographic location and weather feature (e.g., reflectivity of clouds or wind velocity) independent of the radar(s) that generated them. Such sensor-independent names enable end users to specify an area of interest for a particular weather feature, while being oblivious to the placement of

sensors and associated computing facilities. Conversely, the data fusion system can use its knowledge about the underlying system to decide the best radar scanning and data processing strategies. Such sensor-independent names also enhance the resilience and enable processing data close to the source, as well as NDN benefits such as in-network caching and duplicate query suppression consequently reducing the bandwidth requirements of the DCAS system. Our solution is implemented as an overlaid NDN network enabling the benefits of both NDN and overlay networks. An extension is proposed for NDN to support many-to-one data retrieval, as multi-sensor data fusion applications need the ability to retrieve data from multiple sources that match a given name. We also propose novel mechanisms to support query subscriptions, data-generation-time-aware caching, and sensor-specific and event-specific queries. The overlay network enables geographic-name-based query routing. 2-dimensional version of CAN (Content Addressable Network) [Ra01] is used as the underlying overlay network, as it provides a direct mapping between the geographic space and overlay address space while preserving the locality. Simulation-based analysis is used to evaluate the efficacy of the proposed solution using design parameters from the CASA IP1 test bed [Br07, Mc09] and reflectivity data from an actual weather event. Simulation-based analysis showed 87% reduction in average bandwidth consumption of radars and 95% reduction in average query resolution latency.

1.3 Outline

The rest of the dissertation is organized as follows. The following chapter describes related work on P2P topologies, resource discovery, P2P communities, and P2P caching. Detailed background on collaborative P2P applications (e.g., CASA, GENI, and P2P clouds) and NDN is also provided. Chapter 3 presents the problem statement. Research goals, objectives, key phases of resource aggregation, and solution approach are also discussed. Chapter 4 presents the characteristics of multi-attribute resources and queries analyzed using four real-world datasets. Fundamental design choices for P2P-based RD are also evaluated both qualitatively and quantitatively. A set of mechanisms to generate large synthetic traces of multi-attribute resources with static/dynamic attributes and multi-attribute range queries is presented in

Chapter 5. Chapter 6 presents the resource and query aware RD solution and its performance analysis. Proposed community-based caching solution is presented in Chapter 7. Analysis of search clouds from several BitTorrent communities and users' preference to access multiple communities are also presented. Proof-of-concept solution that demonstrates the applicability of NDN for data fusion in multi-user, multi-application, and multi-sensor DCAS systems is presented in Chapter 8. Finally, concluding remarks and future work are presented in Chapter 9. The appendices provide details on survey findings and simulators.

Chapter 2

BACKGROUND AND RELATED WORK

Napster was the killer application that demonstrated the power of Peer-to-Peer (P2P) systems. It paved the way to many successors that are highly scalable and applicable in a variety of application domains. Overlay topology maintenance, Resource Discovery (RD), virtual communities, and caching are among the key research areas in P2P systems. In contrast to current P2P systems that are dedicated to a specific application and share similar resources (e.g., files), future collaborative P2P systems will look for diverse peers that could bring in unique capabilities to a virtual community thereby empowering it to engage in greater tasks. Such P2P systems will require both the adaptation of existing technologies and those yet to be discovered.

This chapter provides a brief description on background and existing work that motivated and relevant to the ideas presented in the dissertation. Section 2.1 describes the work related to overlay topology formation and content/resource lookup. Several applications that can benefit from a collaborative P2P approach are discussed in Section 2.2. Section 2.3 describes P2P-based RD solutions. Communities in P2P systems and P2P caching solutions are presented in Sections 2.4 and 2.5, respectively. Named data networking is presented in Section 2.6.

2.1 Overlay Topologies

P2P architectures can be broadly categorized as structured and unstructured based on the overlay topology formation. We discuss several unstructured (Section 2.1.1) and structured (2.1.2) P2P solutions that are relevant to the following discussion.

2.1.1 Unstructured Peer-to-Peer Systems

Overlay construction in unstructured P2P systems is highly flexible where peers join the network according to some loose set of rules without any prior knowledge about the topology [Lu04]. These topologies typically satisfy the properties of power-law random graph models [Ri02] hence are robust to random node failures. Unstructured P2P architectures can be further categorized as deterministic and nondeterministic [Ra08].

In 1999, Napster [Lu04] emerged as the killer application in P2P systems by enabling a user to download a file from another randomly selected user having the file. It maintains the *resource index* in the form of a centralized database (see Fig. 2.1(a)) which keeps track of the list of files in a peer and its IP address and port number. Resource *lookup*, i.e., the process of searching for resources, is accomplished by querying the database. Overlay connections are established based on the resource interests and peers constantly establish and terminate connections forming an unstructured overlay. Such systems are called *deterministic unstructured P2P systems* because the resources in the database are guaranteed to be found. However, a centralized database leads to a single point of failure and limits the scalability.

In 2001, BitTorrent [Lu04, Po05, Qi04] proposed a unified protocol to communicate across multiple distributed databases enabling users to look up resources from any of the resource indexes. Figure 2.1(b) illustrates the three-layer topology in BitTorrent and a more detailed illustration on interactions

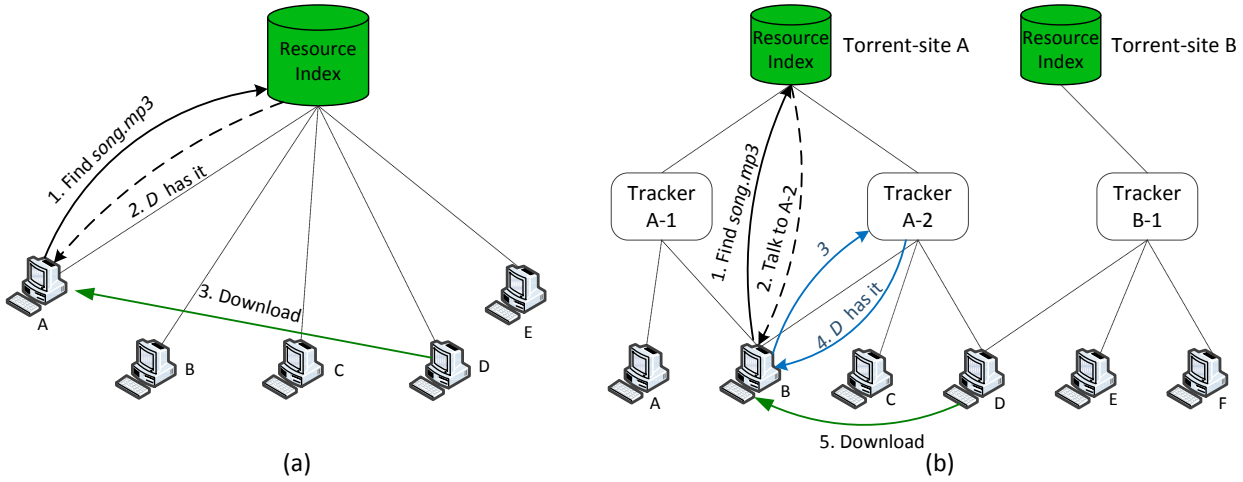


Figure 2.1 – Deterministic unstructured overlays: (a) Napster; (b) BitTorrent.

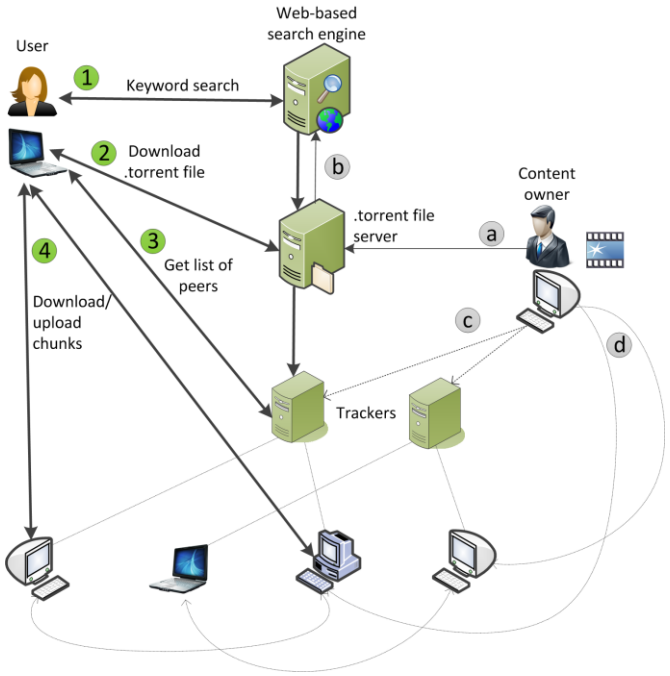


Figure 2.2 – Interaction among different elements in BitTorrent.

among different elements of the system is given in Fig. 2.2. Resource indexes are typically accessed through a website that is referred to as the torrent search engine, torrent site, or community (see Fig. 2.2). A resource is advertised using a *torrent file* which stores the name of the file, its length, the number of *chunks* (i.e., equal sized segments of a file), and a list of SHA1 hash values for each of the chunks. Torrent file also contains a list of *trackers* (i.e., set of nodes that keeps track of the list of peers downloading/uploading the same file).

A user willing to share a file, first generates a torrent file and saves it in a torrent server (step *a* in Fig. 2.2). Then its URL is advertised to one or more torrent search engines (step *b*). Trackers are also informed of the existence of the file (step *c*). Another user interested in downloading that file has to query one of the torrent search engines (step 1). If the query is successful, user's peer first downloads the relevant torrent file and then extracts the list of trackers (step 2). It then contacts one or more trackers and request for a random list of peers sharing the same file (step 3). It then establishes separate connections to a subset of those peers and tries to download the file (step 4). After downloading a chunk, peer advertises itself to the tracker indicating it also has one of the chunks. Peer periodically contacts the tracker(s) to get

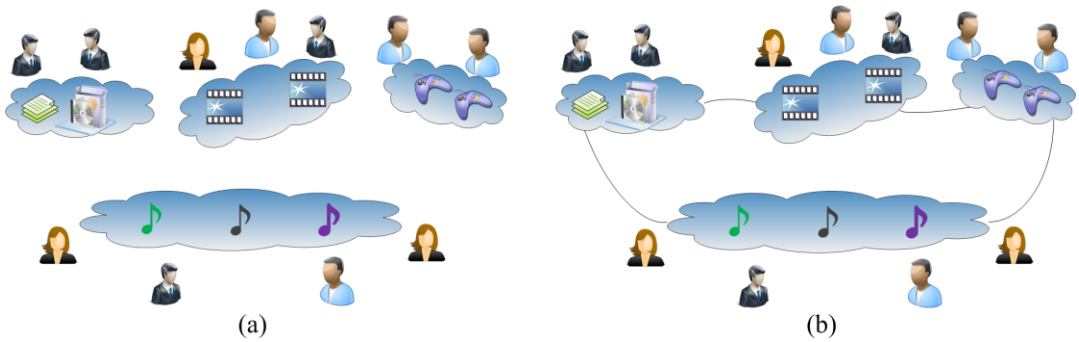


Figure 2.3 – Evolution of BitTorrent communities: (a) Islands of communities; (b) Islands of communities connected using a distributed hash table.

a new set of random peers for newer chunks that it wants to download or for the chunks for which it cannot find a satisfactory uploading peer. BitTorrent enforces fairness by allowing a peer to upload only to the peers that allow it to download. This fairness measure combined with the rarest-first chunk-scheduling policy, enhances the system throughput by enabling bandwidth rich peers to download faster [Fa09]. Similar to Napster, BitTorrent is also a deterministic unstructured P2P system, as resources indexed within a torrent search engines are guaranteed to be found and the overlay topology depends on the resource interests. However, a peer can look up only the resources that are tracked by the trackers that it is aware of; therefore, not all files in the system are guaranteed to be found. BitTorrent has tremendous scalability that is proven by its user base of hundred million users. As BitTorrent grew, many torrent search engines with specific interests on movies, games, software, etc., emerged in a top-down manner. Most search engines deployed their own trackers leading to islands of BitTorrent deployments (see Fig. 2.3(a)). These isolated search engines are referred to as BitTorrent *communities*. Isolation became a problem, as users with diverse interests had to search in many communities to find peers with better upload capacities. Consequently, BitTorrent protocol version 4.2 enabled content lookup across multiple communities using a distributed hash table. Thus, the current BitTorrent system is a top-down aggregation of diverse communities (Fig. 2.3(b)).

Gnutella is the first P2P system to completely distribute both the resource lookup and downloading [Lu04, St08]. A peer joins the overlay network by contacting one of the existing peers and gets a random list of IP addresses of other peers in the network. It then establishes new connections to those peers.

Neighbors constantly share information about other peers in the network, enabling new connections to be established to those nodes once the existing neighbors leave the network. As seen in Fig. 2.4(a), this leads to a random overlay that forms a power-law topology [Ri02] unless a specific Gnutella implementation limits the number of concurrent connections [St08]. Power-law topologies are resilient and reduce the diameter of a network. However, resource lookup is not straightforward as the topology and resource placement are unrelated. Gnutella use flooding with a limited scope to lookup resources. As seen Fig. 2.4(a), each node floods a lookup query to its neighbors, which in turn floods to their neighbors, and the process continues. Scope is defined by a Time To Live (TTL) value that limits the number of hops to forward a query. If a query is successfully resolved, results are returned to the query source through the reverse path. Flooding is extremely costly and does not guarantee to find a resource due to its limited scope. Consequently, some of the Gnutella variants, e.g., Freenet [Lu04], propose to use random walks with a limited TTL. Though random walk reduces the overhead, resources are not guaranteed to be found due to the limited TTL. Therefore, Gnutella is a *nondeterministic unstructured overlay*. Moreover, given an arbitrary network, it is not straightforward to determine the appropriate value of TTL for either flooding or random walks.

Second generation Gnutella [Lu04, St08] and KaZaA [Lu04] proposed a two-layer overlay where resource rich peers, namely *superpeers*, formed a separate overlay while acting as proxies for rest of the peers (see Fig. 2.4(b)). A peer with high capacity in terms of bandwidth, processing power, and/or storage

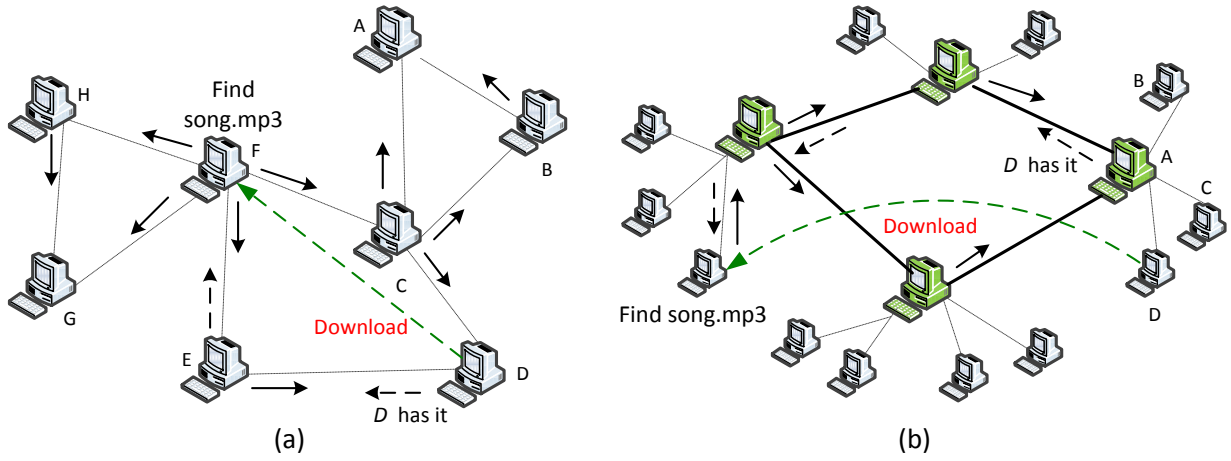


Figure 2.4 – Nondeterministic unstructured P2P systems: (a) Gnutella; (b) Superpeer networks.

is typically promoted as a superpeer. Each superpeer keeps track of the resources available in a subset of peers. Superpeers issue lookup queries on behalf their peers, which are flooded to all the superpeers. This increases the query-hit rate and reduces the lookup latency. Lookup overhead is also relatively low as only the superpeers are involved in flooding. However, scalability is still limited due to the flooding. Several Gnutella variants also propose to use gossiping or random walks among superpeers. Yet, resources are not guaranteed to be found due to the limited scope in flooding and random walks, and unpredictability of gossiping. Therefore, superpeers also belong to the category of nondeterministic unstructured overlays.

2.1.2 Structured Peer-to-Peer Systems

Overlay topologies in structured P2P systems are tightly controlled and resources are indexed at specific locations in such a manner that subsequent lookup queries can be resolved with a bounded overhead [Lu04]. Each peer and a resource in these systems are assigned a unique identifier called a *key*. Each *key* has a corresponding *value* that can be either the resource itself or a pointer to its location. These systems typically maintain the *resource index*, i.e., collection of (*key*, *value*) pairs, in the form of a Distributed Hash Table (DHT). A peer that participates in the DHT is called a *node* (not all peers in the system need to be part of the DHT). Each (*key*, *value*) pair is indexed at a node having a close by *key* in the key space. The resources are indexed and looked up using *put*(*key*, *value*) and *get*(*key*) messages that are forwarded to appropriate nodes using a deterministic overlay. To facilitate such forwarding, each node keeps a set of *pointers* to nodes that are spaced at exponentially increasing gaps in the key space. Such a deterministic overlay and an exponentially increasing set of pointers enable messages to be routed with a bounded path length of $O(\log N)$, where N is the number of nodes in the system. Therefore, structured P2P systems are appropriate for large-scale implementations due to high scalability and some guarantees on performance.

Chord [St03] is the most well-known, flexible, and robust structured P2P system [Gu03]. Figure 2.5(a) illustrates the ring-like overlay maintained by Chord. It maps both the nodes and resources into a

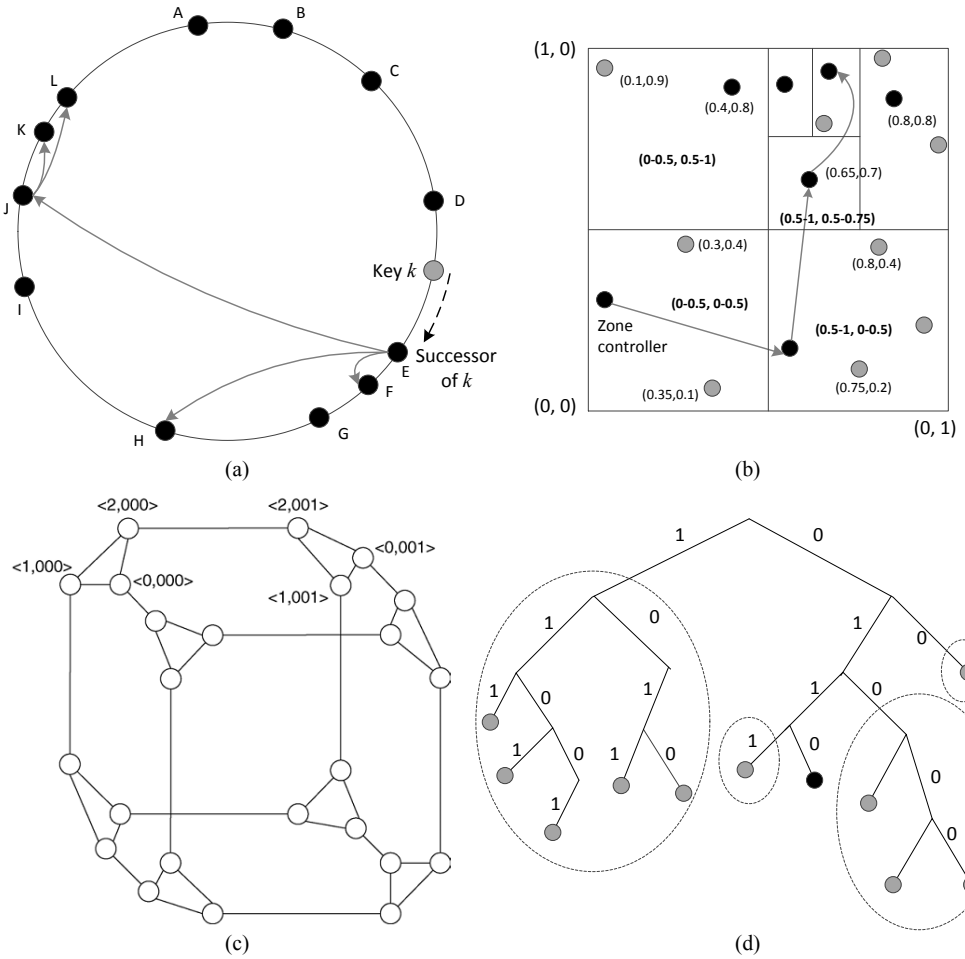


Figure 2.5 – Structured overlay designs: (a) Chord ring; (b) CAN *d*-dimensional torus; (c) Cycloid cube connected cycle [Sh06]; (d) Kademlia binary tree (scattered lines show sub-trees in which node 0110 must keep pointers to).

circular key space using consistent caching [Ka97]. A node is assigned to a random location within the ring and a resource is indexed at the *successor* of its *key*, i.e., the closest node in the clockwise direction. Each node n maintains a set if pointers, called *fingers,* to nodes that are at $(n + 2^{i-1}) \bmod 2^b$, where b is the key length in bits and $1 \leq i \leq b$. For example, node E in Fig. 2.5(a) keeps fingers to nodes F, H, and J. Routing table at a node consists of these fingers, and is called the *finger table*. The fingers are used to recursively forward a message to a given key within a bounded path length of $O(\log N)$. For example, node E can reach node L through the route $E \rightarrow J \rightarrow L$. A node may also identify redundant nodes for each of the fingers to reduce the latency and enhance robustness, e.g., if E knows about K, a message may also take the path $E \rightarrow K \rightarrow L$. Each node maintains $O(\log N)$ finger entries which are refreshed periodically.

The cost of adding a new node or removing an existing one from the overlay is $O(\log^2 N)$. $O(\log N)$ bound for path lengths is guaranteed only when predecessor and successor entries of nodes are valid. Therefore, another periodic stabilization protocol is used to make sure that these entries are valid. Thus, it is costly to maintain the Chord overlay on a dynamic network through messages can be routed efficiently.

Content Addressable Network (CAN) [Ra01] is based on a *d*-dimensional torus (*d*-torus). Figure 2.5(b) illustrates a 2-dimensional torus that is partitioned into a set of *zones*. Nodes are assigned random identifiers in the *d*-dimensional space. Resources are assigned identifiers by hashing their unique names. First node keeps track of the entire *d*-torus. When a new node is added, it is given a random *key*. It is then routed to the zone that is responsible for indexing its *key*. The zone is then divided into two equal volume zones and each zone is assigned to the new node and to the owner of the previous zone. Zones are further divided or combined as nodes join and leave. A node is responsible for keeping track of (*key*, *value*) pairs that maps to its zone. A resource is located by forwarding a query message to the zone responsible for indexing the key specified in the query using greedy routing. CAN nodes maintain up to $2d$ routing entries to their neighboring zones (two nodes are neighbors if their coordinate spans overlap along $d - 1$ dimensions and abut in the remaining dimension). These routing entries are used to route a message within $O(dN^{1/d})$ hops using greedy routing, where N is the number of nodes in the system. CAN's routing scheme alleviates the local minima problem that occurs in other greedy routing schemes such as geographic routing as it calculates the distance from a given identifiers to the edge of a zone instead of to a specific point. CAN further proposes several enhancements to reduce the lookup overhead, e.g., increasing *d*, Round Trip Time (RTT) based next hop selection, zone formation based on distance to known landmarks, and large zones. A lower number of alternative paths and failure of neighbors reduce CAN's resilience.

Pastry [Ro01] is a hypercube-based solution that represents keys using a string of digits where each digit is in base 2^b. Pastry routes messages using prefix-based routing, where it tries to reach the given *key* or a numerically closest node by correcting one digit at a time. Each node maintains a routing table, neighborhood set, and leaf set. Routing table consists of a set of $\log_{2^b} N$ rows each with $2^b - 1$ entries, which points to nodes that are spaced at different distances in the key space. For each raw i in the table, a

node tries to maintain $2^b - 1$ pointers to nodes that have identical i prefixes to its key. A node also keeps a set of pointers to neighbors in the key space (called the left set) and to physically close neighbors (called the neighbor set). Pastry uses these two sets to enhance the routability and reduce the lookup latency by forwarding messages to neighbors with the least RTT. Pastry routes messages within $\log_{2b} N$ unless several nodes with adjacent keys fails simultaneously. To enhance the resiliency and load balancing, a (*key*, *value*) pair is stored in multiple nodes that are closer to the given *key*. Pastry routing tables are relatively large and contain $O(b \log_b N)$ entries per node.

Cycloid [Sh06] extends the Pastry hypercube to form a cube connected cycle (see Fig. 2.5(c)). Each *key* is represented using a pair of indices (k, $a_{d-1}a_{d-2}\ldots a_0$) where k is the cyclic index, d is the dimension of the hypercube, and $a_{d-1}a_{d-2}\ldots a_0$ is the cubical index represented as a string of digits. Such a design limits the address space to $d \times 2^d$. (*key*, *value*) pairs are placed on the numerically closest node. Cycloid maintains a fixed number of pointers (typically seven or eleven) to neighbors based on cyclic (to nodes in the same cycle) and cubical index (to nodes in the hypercube). Cycloid also uses prefix-based routing and routes message within $O(d)$ hops. Though small routing table size is desirable, it could lead to lower resilience as Cycloid has a limited number of alternative paths.

Kademlia [Ma02] uses a novel XOR metric for distance calculation. (*key*, *value*) pairs are indexed at several nodes that are closest to the *key*. Each node maintains a k-bucket routing table where each bucket i keeps track of k nodes that are within the distance $[2^i, 2^{i+1})$ (see Fig. 2.5(d)). In contrast to other structured P2P systems that use a separate set of messages to maintain the overlay, Kademlia uses on going the lookup queries to identify new routing entries. This is enabled by the symmetric property of the XOR metric, which allows a node to receive lookup queries from precisely the same distribution of nodes contained in its routing table. A lookup query is resolved by sending a set of parallel queries to m nodes selected from the k-buckets according to their closeness (measured using XOR) to the given *key*. Contacted nodes may respond with a set of even-closer nodes. Another set of m nodes is then selected from those reported nodes and another set of queries is sent. The process repeats until a node with even better distance cannot be found. Finally, the closest set of nodes is queried to locate the resource. These parallel

and asynchronous queries reduce the delays due to failed nodes and increase the resilience. Kademlia also routes messages within $O(\log N)$ hops. It is used in many production P2P systems such as BitTorrent and eMule (to find resources that are indexed by other trackers) due to its ease of implementation and high resilience.

Table 2.1 compares these solutions and several other structured P2P systems (refer [Lu04] for more details). Though these solutions provide guaranteed RD and have a bounded lookup overhead, they have several fundamental limitations. DHTs require all copies of the same key to be stored in the same node or neighborhood. This leads to load imbalance when query popularity is skewed [Ba11e, Ba12a, Kl04, Sr01]. Moreover, it leads to single points of failure. Many of the structured P2P solutions propose to use replication to overcome this issue e.g., [Ma02] and [St03]. However, replication could lead to an inconsistent or stale resource index when resources are highly dynamic or the resource owner leaves the network. Moreover, the load balancing issue remains when multiple copies of a (*key*, *value*) pair need to be checked for consistency. Furthermore, their performance bounds are guaranteed only if the overlay network is consistent. Maintaining overlay consistency is costly (except in Kademlia) even in a network with moderate churn. Moreover, average-case performance of these systems is too high for large-scale latency sensitive systems such as CASA and P2P clouds. Therefore, we are still in the need for a truly

Table 2.1 – Summary of structured P2P solutions.

Scheme	Architecture	Routing Mechanism	Lookup Overhead*	Routing Table Size*	Join/Leave Cost	Resilience
Chord	Circular key space	Successor & long distance links	$O(\log N)$	$O(\log N)$	$O(\log^2 N)$	High
CAN	d-torus	Greedy routing through neighbor zones	$O(dN^{1/d})$	$2d$	$2d$	Moderate
Pastry	Hypercube	Correct one digit in key at time	$O(\log_B N)$	$O(b \log_b N)$	$O(\log_b N)$	Moderate
Tapestry	Hypercube	Correct one digit in key at time	$O(\log_B N)$	$O(\log_b N)$	$O(\log_b N)$	Moderate
Viceroy	Butterfly network	Predecessor & successor links	$O(\log N)$	$O(1)$	$O(\log N)$	Low
Kademlia	Binary tree, XOR distance metric	Iteratively find nodes close to key	$O(\log N)$	$O(\log N)$	$O(\log N)$	High
Cycloid	Cube connected cycles	Links to cyclic & cubical neighbors	$O(d)$	$O(1)$	$O(d)$	Moderate

* N – number of nodes in overlay, d – number of dimensions b – base of a key identifier

Table 2.2 – Structured vs. unstructured P2P systems.

	Unstructured P2P	Structured P2P
Overlay construction	High flexibility	Low flexibility
Resources	Indexed locally (typically)	Indexed remotely on a distributed hash table
Query messages	Broadcast or random walk	Unicast
Content location	Best effort	Guaranteed
Performance	Unpredictable	Predictable bounds
Overhead	High	Relatively low
Object types	Mutable, with many complex attributes	Immutable, with few simple attributes
Peer churn & failure	Supports high failure rates	Supports moderate failure rates
Load balancing	Relatively better load distribution	The load is imbalanced when queries and/or resources are skewed
Resilience	High	Single points of failure
Consistency of index	High (nodes keep their resources)	Low (indexed remotely)
Applicable environments	Small-scale or highly dynamic environments with (im)mutable objects, e.g., mobile P2P	Large-scale & relatively stable environments with immutable objects, e.g., desktop file sharing
Examples	Gnutella, LimeWire, Kazaa, BitTorrent	Chord, CAN, Pastry, Kademlia, BitTorrent

distributed P2P system that combines the desirable properties of both the structured and unstructured P2P systems. Table 2.2 summarizes the properties of both types of P2P systems.

2.2 Collaborative Peer-to-Peer Systems

Collaborative P2P systems are applicable in a wide variety of contexts such as Distributed Collaborative Adaptive Sensing (DCAS) [Ku06, Le12, Mc05, Mc09], grid [Ca04, Sh07], cloud [Ar09], and opportunistic [Co10] computing, Internet of Things [Pf11], social networks, and emergency management. Next, we discuss several representative collaborative applications in detail.

2.2.1 Collaborative Adaptive Sensing of the Atmosphere

Current weather radar networks are typically comprised of physically large, high power, and highly expensive radars spaced at several hundreds of kilometers apart. For example, WSR-88D Next Generation Weather Radars (NEXRAD) in the U.S. are spaced at ~345 km apart in the western U.S. [Mc09]. These widely spaced, long-range radars use high power transmitters to sense the atmosphere that is 200-300 km away (see Fig. 2.6(a)). While these systems have led to significant improvements in weather forecasting and warning, they are unable to see the lower 3 km of the atmosphere due to the

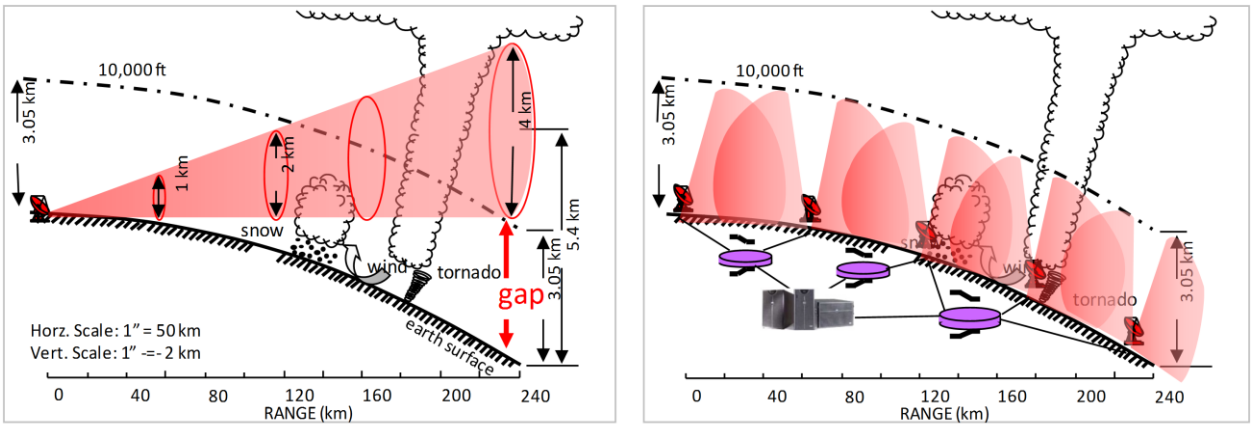


Figure 2.6 – Evolution of weather radar networks: (a) A long range and high-power radar; (b) Tracking lower 3 km of the atmosphere using a network of radars.

Earth's curvature and terrain blockage (Fig. 2.6(a)). Ability to sense the lower 3 km of the atmosphere is important for accurate detection and forecasting of localized weather events such as tornados and flash floods.

Collaborative Adaptive Sensing of the Atmosphere (CASA) is a DCAS system that is revolutionizing how we observe, evaluate, understand, and predict hazardous weather events such as tornados and flash floods. Figure 2.6(b) illustrates the simultaneous observation of a weather phenomenon by a network of CASA radars. Central to the CASA research effort is the use of large numbers of low-cost small radars, spaced close enough to see the lower 3 km of the atmosphere in spite of Earth's curvature and to avoid resolution degradation caused by radar beam spreading [Mc09]. Such a dense network of radars enables the same weather event to be sensed from multiple angles consequently increasing the accuracy of sensing, detection, and forcasting. CASA radars, processing nodes, and data-fusion algorithms communicate with each other to adjust their sensing and data processing strategies in direct response to the evolving weather and to changing end user needs [Ku06, Mc09]. CASA also employs many small sensors such as pressure sensors and micro-weather stations [Pe11a, Pe12] to further enhance the detectability and forcasting accuracy.

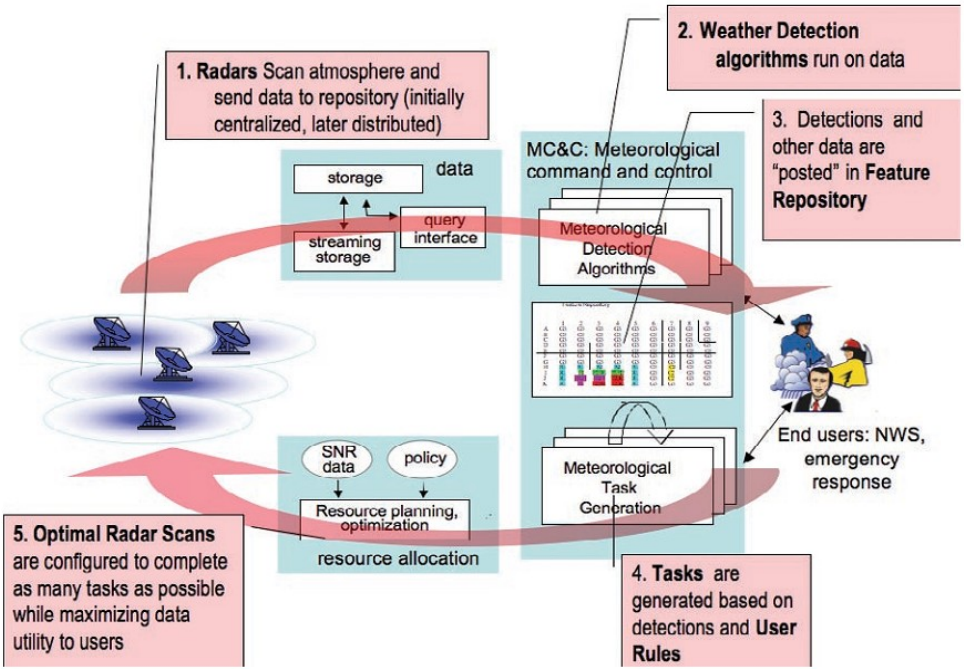


Figure 2.7 – Major processing steps of the closed-loop MC&C software architecture [Mc09].

CASA IP1 test bed [Br07, Mc09] that was in Oklahoma consisted of four radars placed on a rhombus with inter-node spacing of 30 km. IP1 is currently being relocated to Dallas, TX and will be expanded into an eight-radar network. IP1 radars covered an area of ~7,000 km^2 using a transmission range of 40 km and were connected to the Internet. IP1 radars are controlled through the Meteorological Command and Control (MC&C) which closes the loop between sensing and radar tasking [Mc09]. Closed-loop operation of MC&C is depicted in Fig. 2.7. MC&C ingests data from radars, identifies meteorological features in data, reports features to end users, and determines future scan strategies of radars based on the detected weather features and end users' information needs. To satisfy the CASA's goal of detecting severe weather events within 60 seconds, closed loop is executed every 30 seconds [Zi05].

CASA supports a diverse set of meteorological algorithms (referred to as *applications*) and end users. Table 2.3 lists a subset of the applications that are currently supported by CASA. Each application pulls one or more types of data from one or more radars. For example, radar images that we see on TV newscasts are drawn using reflectivity data from clouds that are typically generated by a radar. More accurate reflectivity images can be generated using the Network-Based Reflectivity Retrieval (NBRR)

Table 2.3 – CASA applications. Adapted from [Ba13].

Application	Description	No of Radars	Data Type(s)
Reflectivity	Reflectivity of clouds	1	Reflectivity
Velocity	Wind velocity	2-3	Doppler velocity, reflectivity
Network-Based Reflectivity Retrieval (NBRR)	Reflectivity of clouds detected using multiple radars	3+	Reflectivity
Nowcasting	Short term (10-30 min) high resolution forecasts of active weather events	1-3	Reflectivity
Quantitative Precipitation Estimation (QPE)	Estimating current precipitation using intensity of rain & water droplet size	1-3	Reflectivity, differential phase, correlation coefficient
Tornado tracking	Detect & track a tornado as it forms & moves	2+	Doppler velocity, reflectivity
Air surveillance	Low-flyer surveillance for law enforcement	1-3	Doppler velocity, reflectivity

[Li07b] algorithm that pulls reflectivity data from three or more radars that sense the same region in atmosphere within an acceptable time window. Both Doppler velocity and reflectivity data from two to three radars are needed to estimate the wind velocity accurately. The same data are useful in tornado-tracking and low-flyer surveillance [Pe11b] applications. Therefore, multiple applications tend to access subsets of the same data. Applications require different amounts of computational, storage, and bandwidth resources as they use different types of data, volumes of data, and meteorological algorithms. Known weather patterns, geography, cost, and availability of infrastructure determine where the applications are deployed. For example, tornado-tracking applications are deployed only in areas that are likely to have tornados.

These applications are accessed by a diverse set of end users (see Table 2.4) such as the National Weather Service (NWS), Emergency Managers (EMs), scientists, media, and commercial entities. Users may issue queries periodically for weather surveillance or when an interesting weather event is detected within their Area Of Interest (AOI). For example, a NWS forecast office sends a separate query for each of the applications listed in Table 2.3 (except for air surveillance) for counties under their jurisdiction. For surveillance purposes, they may pull data from reflectivity and velocity applications every five minutes regardless of the current weather conditions. However, when an active weather event is detected, reflectivity, velocity, NBRR, nowcasting, and QPE applications are queried at a higher sampling rate. These queries are periodically issued for the area of active weather (which may change with time) until the weather event subsides or move out of their jurisdiction. A researcher trying to understand the physical

Table 2.4 – CASA end users and their data access patterns. Adapted from [Ba13, Ku06].

End User	Description	Applications	Rule Trigger	AOI	Sampling Interval
National Weather Service (NWS)	Responsible for issuing warnings	Reflectivity	Periodic	Counties under jurisdiction	1 min
		Velocity			
		NBRR, nowcasting, QPE	High reflectivity	Area of active weather (even if 30-80 km away)	
		Tornado tracking	Rotating wind, ground spotters		
Emergency Managers (EMs)	Siren blowing, helping first responders, act as spotters	Reflectivity	Periodic	Counties under jurisdiction	1 min
		Velocity			
		NBRR, nowcasting, QPE	High reflectivity	Area of active weather (even if 30-80 km away)	2 min
		Tornado tracking	Rotating wind, ground spotters		1 min
Researchers	To understand physical properties of weather events, test new algorithms	Reflectivity	Periodic	Area of active weather	1 min
		Velocity	High wind		30 sec
		NBRR, nowcasting, QPE	High reflectivity		1 min
		Tornado tracking	Rotating wind		30 sec
Media	Forecasting, public warning	Reflectivity	Periodic	Counties/states under media coverage	1 min
		Velocity			
		NBRR, nowcasting, QPE	High reflectivity	Area of active weather (even if 30-80 km away)	2-5 min
		Tornado tracking	Rotating wind, ground spotters		
Commercial entities	Transportation agencies, utilities, commercial entities	Reflectivity	Periodic	Counties/states under interest	5-60 min
		Velocity			

properties of a tornado may use velocity and tornado-tracking applications every 30 seconds to acquire samples more frequently. Alternatively, commercial entities may sample their AOIs at a much lower sampling rate, as they are interested in mid to long-term changes in weather. The public is not expected to interact with CASA directly instead access data from media.

A potential nationwide CASA radar network deployment in the U.S. is estimated to require 10,000 radars [Mc09]. While such a dense network of radars can substantially improve the detection, forecasting, and warning time, it creates many challenges due to the sheer number of sensors involved, the heterogeneous network and communication infrastructure, and volume of data generated. CASA IP1 radars generate raw data at rates up to 800 Mbps, which reduces to 3.3 Mbps with preprocessing. In some cases, e.g., to preserve the accuracy or for archiving purposes, it is preferable to transfer raw data. The next generation of solid-state CASA radars is expected to generate raw data at several Gbps. Even though

the system is mission critical, CASA uses the Internet as the preferred medium of communication because of its flexibility, global accessibility, and low cost. Therefore, new transport protocols, resource aggregation solutions, and multi-sensor Data Fusion (DF) solutions are needed to timely transmit and process large volumes of radar data while overcoming dynamic network conditions. Real-time nature of CASA also necessitates the radar data generation, transmission, processing, and re-tasking to be completed within 30 seconds. However, most of this time is used to generate the data leaving a fraction of this time for data transmission and fusion. Moreover, certain rare but severe weather events require specific algorithms/applications and more computing, storage, and bandwidth resources to track and forecast/nowcast about their behavior. Hence, the key design consideration of the radar network is its ability to meet the application-specific real-time requirements while optimizing resource usage.

With each of the sensor nodes allowed to conduct in-network processing and provide computation and communication resources in response to user requests [Do05], the multi-radar DF can be performed collaboratively and concurrently by different nodes. Multi-radar DF involves collecting data from multiple remote radars and processor-intensive digital signal processing. The applications therefore place unique weather event and context sensitive demands on the system infrastructure. In these circumstances, a distributed, dynamic, and collaborative approach based on the P2P architecture is attractive to aggregate underutilized resources from multiple sensors and processing nodes across the network. However, the underlying network and processing infrastructure may be subjected to adverse conditions due to severe weather, resource failure, link degradation, and variable cross-traffic along wired and wireless links. Given the importance of the application, P2P architecture should be robust enough to function under such adverse conditions by automatically masking any inadequate resource using other resources. For example, lack of bandwidth between a processing node and a storage node may be compensated by processing data faster to accommodate the extra delay introduced while transferring data to the storage node. Though it is mission-critical, provisioning a CASA radar network for rare peak demand is neither economically feasible nor practical due to the spatial and temporal locality of hazardous atmospheric events. Instead, the available resources have to be dynamically managed to meet the requirements with an acceptably high

probability. A collaborative P2P architecture provides many of the attributes needed for such resource sharing. However, a simple resource aggregation approach is not sufficient for multi-sensor DF, as the data need to be combined in such a manner that the real-time requirements and application-specific data selection requirements are met. For example, to meet real-time bounds, we need to ensure that the nodes selected for DF have the required processing capabilities (including computation capability, bandwidth, and latency) and the appropriate selection of software/hardware for the specific type of processing. There is thus a need for a framework capable of timely aggregating diverse set of resources in mission-critical DCAS systems.

2.2.2 Global Environment for Network Innovations

The Global Environment for Network Innovations (GENI) is a collaborative and exploratory platform for discoveries and innovation [El09]. It is a suite of research infrastructures rapidly put together to explore the future Internet at scale. GENI allows users to aggregate resources (e.g., processing nodes, storage, networks, sensors, and actuators) from multiple administrative domains for a common task. Figure 2.8 illustrates the GENI resource aggregation framework where a user requests to create a slice by aggregating a given set of resources. The clearinghouse responds to the user request by aggregating a set

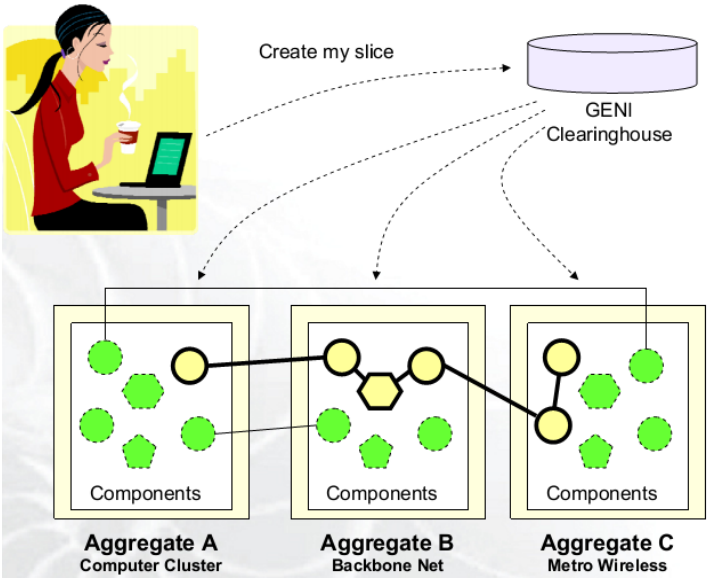


Figure 2.8 – GENI resource aggregation framework [El09].

of resources that are spread across several administrative domains. In its current design, GENI clearinghouse passively sends a list of potentially useful recourses to the user hoping he/she has the skill to put them into a workable system. In practice, a user will find it is extremely complex to build even a satisfactory system for two reasons. First, a user tends to request an arbitrary set of resources relying on the instincts or principle of least effort [Br05] rather than on specific requirements of the application. Therefore, the user ends up requesting either insufficient or too many resources. Second, as the clearinghouse focuses only on individual resources not all combinations of the selected resources may be suitable or capable of working together. This degrades the QoS of the user's application or in certain cases user may not be able to build a workable solution at all. Ideally, the clearinghouse should take these complexities away from the user and should be able to intelligently aggregate an optimum set of resources based on the application requirements. To do so, it needs to take into account the entire group of resources and their inter-resource relationships. A centralized clearinghouse will be insufficient as GENI continues to grow attaching many users and resources that are geographically distributed. Therefore, a collaborative P2P system is a good fit for GENI because of its distributed, dynamic, and collaborative nature.

2.2.3 Peer-to-Peer Clouds

Cloud computing is transforming the way we host and run applications because of its rapid scalability and pay-as-you-go economic model. The datacenter hardware and associated software resources are referred to as the *cloud* and the process of delivering services over the Internet through these hardware and software is referred to as *cloud computing* [Ar09]. Thus, cloud computing is an abstract term that captures a variety of concepts that combine hardware, software, and networking resources to different extents. Cloud services can be broadly classified as [Fo09]:

1. *Infrastructure as a Service (IaaS)* – These services provide access to bare-bone hardware through user configured Virtual Machines (VMs). Typically, these services are offered as bundles of resource instances with different capabilities. For example, Amazon EC2 name their computing resources as standard, micro, high-memory, high-CPU, and cluster compute [Amaz].

2. *Platform as a Service (PaaS)* – Cloud providers offer middleware services enabling users to develop applications rapidly. Complexity of these service platforms varies, e.g., from relatively simple .NET common language runtime in Moorcroft Azure [Micr] to relatively advanced Google App Engine.
3. *Software as a Service (Saas)* – At this level, an entire application or suite of applications is exposed as a service. Google Docs, Office Live, and Salesforce.com are some of the well-known examples.

With increased levels of integration, even the systems within and across cloud computing datacenters exhibit attributes of distributed systems where groups of resources such as processing nodes, storage, bandwidth, and special hardware (e.g., GUPs and FPGAs) may be grouped to execute complex collaborative applications. These applications even need to establish virtual networks within the datacenter to isolate traffic and provide bandwidth guarantees. While the VMs increase the resource utilization of a datacenter, they make it harder to provide QoS guarantees. For example, both Amazon EC2 and Microsoft Azure platforms run multiple VMs on the same physical node. Though VMs can be configured not to exceed specific amounts of CPU utilizations, memory, and storage, their I/O performance cannot be controlled. Thus, the overall performance received by an application running on a VM depends on the behavior of other VMs on the same host. This could lead to unpredictable performance particularly in data intensive applications that are I/O intensive (e.g., CASA radar data fusion [Ir10] and high-energy physics applications [Ge11b]). Consequently, modern clouds are struggling to provide fine-grained Service Level Agreements (SLAs) [Am08, Micr] due to the inherent complexity of describing resource capabilities, inter-resource relationships, and application requirements. Cloud computing systems are scalable and allow users to rapidly respond to increasing application demands by purchasing additional resources on the on the fly, e.g., Amazon Auto Scaling. However, their response times are in minutes as it takes time to transfer an operating system image to a node and boot-up a new VM. In its current state, these systems do not provide the rapid scalability and adaptability that CASA-like systems require which have significantly higher peak demands that last only for tens of minutes. However, with the advancement of hardware and

operating system technologies, future cloud computing systems will be able to support latency sensitive and mission critical applications.

In spite of its distinct advantages, there are some criticisms on cloud computing. For example, its centralized data and proprietary application model contradict with the Free and Open Source Software (FOSS) movement; hence, considered a threat by users who want to be in control of their data and applications. One may even argue that it is a waste of resources in modern desktops, which are forced to act like thin clients while the datacenter performs all the heavy lifting. Though FOSS applications can be ported to run on a cloud, lack of free and open datacenters is an issue. This limitation can be overcome by making the FOSS community become a distributed datacenter. We can combine the power of P2P systems and cohesiveness of the FOSS community to build a P2P cloud-computing infrastructure where users contribute resources to the cloud while staying in control of their data and applications. Such a community cloud computing system, based on underutilized computing resources in homes/businesses, can overcome centralized data, privacy, proprietary applications, and cascading failures in modern clouds [Br09]. A collaborative P2P system is the core of such a system that allows users to contribute their underutilized resources while effectively dealing with rapid scalability and resource fluctuations.

2.2.4 Mobile Social Networks

The value of social networks can be enhanced by allowing users to share diverse resources available in their mobile devices. For example, as seen in Fig. 2.9, a person with a basic mobile phone could connect to a friend's smart-phone with GPS capability to locate a nearby ATM. In another example, a group sharing their holiday experiences in a coffee shop could use one of the members' projection phone to show pictures from others' mobiles or tablets, or videos streamed from their home servers. Such applications benefit from the diversity of resources in a community where members share resources with each other. Depending on the situation, users may even establish transient connections to achieve a common goal, e.g., during a conference or while responding to a disaster. For example, in large social gatherings such as carnivals, sports events, or political rallies, users' mobile devices can be used to share hot deals,

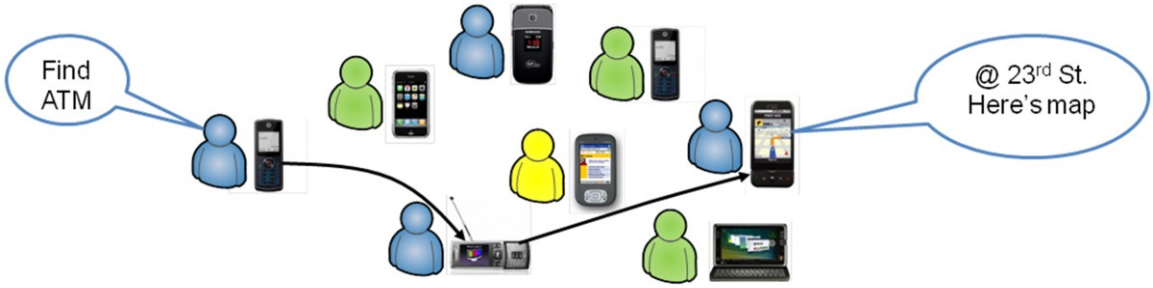


Figure 2.9 – A user trying to locate an ATM using his/her mobile social network.

comments, videos, or vote for a certain proposition without relying on a network infrastructure. Such applications are already emerging under the domain of opportunistic networking and computing [Co10]. Thus, social relationships among users have to be taken into account to find the willingness to share their resources. These networks also need to ascertain whether the two resources are nearby to avoid a certain service provider, minimize latency, or reduce packet loss. We refer to these applications as *mobile social networks*. Collaborative P2P systems are a natural fit for these emerging networks that are distributed, highly dynamic, and autonomous. Resource advertising, discovering, and matching based on inter-resource relationships and user constraints are among the fundamental requirements of mobile social networks.

2.3 Peer-to-Peer-Based Resource Discovery

Below we discuss a representative subset of P2P solutions that either address or have the potential to address requirements of collaborative P2P systems. Unstructured P2P-based solutions are discussed in Section 2.3.1 and structured ones are discussed in Section 2.3.2.

2.3.1 Unstructured Peer-to-Peer Solutions

Unstructured P2P systems are based on random overlays where resources are advertised and/or queried by sending messages through flooding or random walks. Flooding can be used either to advertise Resource Specifications (RSs) (i.e., attributes of a resource and any constraints on usage of the resource) or to select resources on the fly by sending multi-attribute range queries. In either way, all the nodes can

get to know about all the resources in the system thereby enable the best set of resources to collaborate. A processor cycle sharing system that is capable of executing real-time jobs is presented in [Ya06]. It is based on an analytical model that assumes the jobs are perfectly divisible and the job owner will execute, at least, a small fraction of the job. Job owner first floods the neighborhood to identify computing power, bandwidth, and reliability of its neighbors. The model then assigns different fractions of the job to neighbors based on their capabilities. The analytical model fails when jobs are not perfectly divisible or the job owner is unable to execute part of the job due to lack of resources. A simplified best-peer-selection algorithm for CASA multi-sensor DF is presented in [Le12]. However, flooding is extremely costly thus suitable only for small-scale applications.

Gossiping [Je06] is another alternative to disseminate RSs where a software agent exchanges RSs between two randomly selected nodes. Multiple concurrent agents are used to speed up the gossiping. A node selects resources by querying the RSs that are gathered from gossiping. Efficient Resource Discovery (ERD) [Th09] is a resource advertising and querying scheme for mobile ad-hoc networks. It prioritizes messages based on their TTL and time they have spent in the system allowing rapid dissemination of new messages. Though ERD is designed to advertise and select individual resources, it is possible to do simple resource matching as a node gets to know about multiple resources. For example, it can answer a query like "*are resources x and y from my friends?*". Though this approach is simple to implement, there is no guarantee that a node will get to know about any resource that it needs (even if it exists). Moreover, states of know resources may be stale as gossip propagation is unpredictable and slow.

The majority of the unstructured P2P solutions is based on random walks where RS advertisements and/or queries are forward through a series of nodes that are selected randomly. The random walk is a more specific form of gossiping where an agent typically carries only a selected set of RSs or multi-attribute range queries between two neighboring nodes. In [Ta08], a node sends out multiple agents to advertise its RS(s) to other nodes along their path and to collect those nodes' RSs. Multiple agents are sent to different regions of the network to sample larger portion of the network and enhance the robustness. Simple resource matching is also possible as agents collect RSs from multiple nodes. Though agents

advertise and discover resources, they do not provide guaranteed resource selection and even the state of the selected resources may be stale. Alternatively, inspired by the second-generation Gnutella P2P system, many solutions issue queries (using random walks) as and when they need to select resources. Each query agent walks from one node to another looking for resources satisfying the multi-attribute range query [Xi08b]. If the relevant resources are found, the agent goes back to the query originator either through the reverse path or directly. Agents have a limited lifetime, defined using a TTL, to prevent the accumulation of unresolved query agents within the P2P system. To increase the probability of discovering resources a node either sends several agents or sends an agent with a large TTL. However, it is nontrivial to determine the required number of agents or their TTL to guarantee RD in an arbitrary network. Therefore, such solutions can only provide a best-effort service. Nevertheless, query agents can identify the current state of a resource and provide resource binding (i.e., a guarantee that the resource can be used by the application for its intended purpose and time), as they reach individual nodes to check their resource availability. Resource matching can be also performed at the same time, if a node has multiple resources such as processing capabilities and storage. Past query results can be used to bias the random walk towards potential resources while reducing the query overhead and latency [Xi08b]. Another alternative is to build a hybrid system where one set of agents advertises the RSs while another set queries for the resources [Ta08]. While this speeds up the query resolution, it may not reflect the correct state of resources and eliminates the possibility of resource binding as queries are resolved by intermediate nodes.

Another alternative is a two-layer overlay (similar to KaZaa) where resource rich peers, namely *superpeers*, form a separate overlay while acting as proxies for rest of the peers [Kw10, Su08b, Xi08b]. Each superpeer keeps track of RSs of a subset of peers. Superpeers advertise and/or query resources on behalf of their peers by contacting other superpeers through flooding, gossiping, or random walks. A superpeer-based task assignment mechanism for volunteer desktop grids is presented in [Kw10]. Superpeers reduce the overhead as only a subset of the peers is involved in advertising and querying resources. Superpeers have the potential to act as matchmakers, if physically nearby peers or peers with different resources are assigned to the same superpeer. They can also provide resource-binding services

on behalf of their peers. However, its applicability is limited as it focuses only on individual nodes and assumes processing nodes and application requirements can be completely characterized by their MIPS (Million Instructions Per Second) ratings.

In conclusion, unstructured P2P solutions provide a best-effort service (except when messages are flooded) in selecting and binding dynamic resources. They are applicable in small to medium scale applications and highly dynamic environments such as ad-hoc and mobile social networks. Unstructured topologies are also attractive as they better distribute the index size and query load due to random topologies. It is more useful to issue queries on the fly as complex queries, having multiple attributes and range of attribute values, can be relatively easily resolved as agents reach individual nodes. This further simplifies the binding between a resource and the application interested in using it. A restricted form of resource matching is also possible. When a node cannot match all the relevant resources within itself, all the resources that satisfy the resource query have to be informed to the node that initiates the application. The application may then take the final decision on resource matching and binding. Nevertheless, random topologies in unstructured P2P systems make it hard to keep track of inter-resource relationships. Therefore, resource relationships have to be discovered on the fly. For example, after selecting potential resources an application could then request nodes to verify whether they can reach each other, measure bandwidth/latency between them, or evaluate their social relationships. However, discovering such relationships on the fly takes time and increases the overhead, e.g., sufficient number of packets and time are needed to estimate the bandwidth between two nodes. Superpeers provide a viable alternative where a superpeer can at least keep track of the relationships among resources that are connected to it.

2.3.2 Structured Peer-to-Peer Solutions

Structured P2P solutions are appropriate for large-scale P2P applications due to their scalability and some guarantees on performance. These systems typically index RSs in a DHT. DHTs are designed to index resources that are characterized by a single attribute. Thus, they are not designed for simultaneous

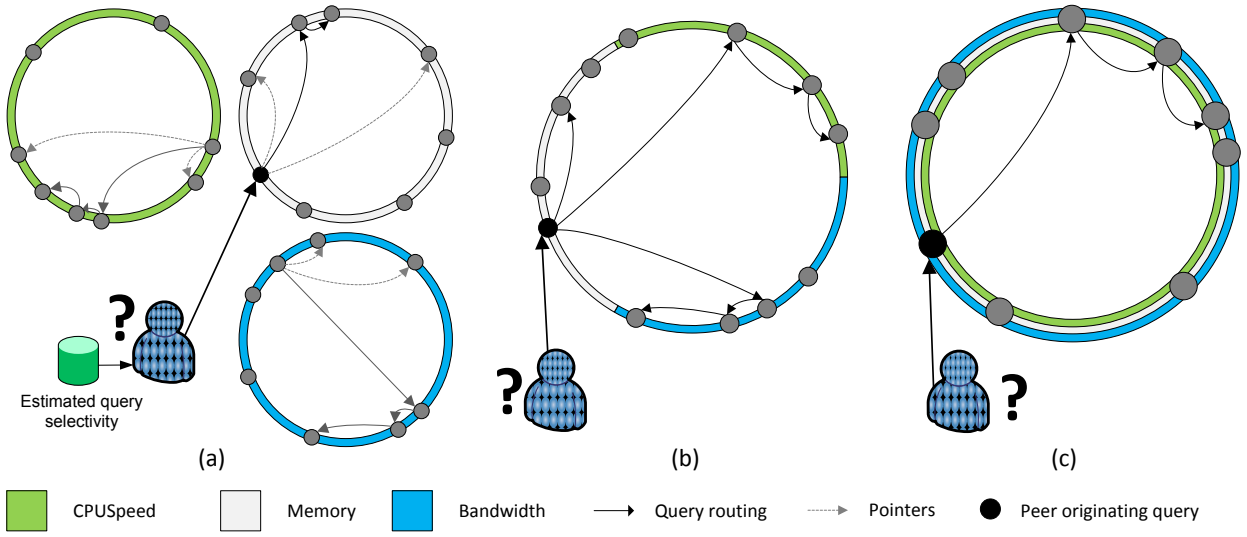


Figure 2.10 – Ring-based structured overlay designs: (a) Multi-ring – a separate overlay is created for each attribute type and queries are issued to the most selective attribute; (b) Partitioned-ring – address pace of the overlay ring is partitioned and assigned to different attribute types, and queries are resolved by issuing multiple sub-queries to each partition; (c) Over-lapped ring – multiple addresses spaces are mapped to the same overlay ring and queries are issued based on the most selective attribute.

selection of multiple and multi-attribute resources required by collaborative P2P applications. Next, we discuss several solutions that extend structured P2P systems to support multi-attribute resources.

Figure 2.10 illustrates three design choices based on an overlay network with a circular address space, also referred to as a *ring*. Mercury [Bh04] maintains a separate ring for each attribute type (see Fig. 2.10(a)). Each node advertises its RS(s) to the rings that corresponds to the attribute set of the resource(s). Mercury utilizes Single Attribute Dominated Queries (SADQ) where a query is issued only to the ring corresponding to the most selective attribute. For example, as seen in Fig. 2.10(a), the query travels 3-hops in the *CPUSpeed* and *bandwidth* rings, and 2-hops in the *Memory* ring. Thus, it is more efficient to resolve the query using the *Memory* ring. To enable SADQ, each node needs to keep track of all the attributes of RSs that it indexes. Mercury uses a random-sampling algorithm to estimate the query selectivity within each ring. A range query is resolved by forwarding it to a series of nodes through their successors. As the query propagates, resources that satisfy all the attributes specified in the query are aggregated and the last node returns the aggregated results to the application (i.e., query originator). The random sampling algorithm is also used to estimate the query load within each ring. Based on the estimated load,

nodes are reorganized within each ring to balance the query load. New rings can be added to support additional attributes and each ring can be customized to specific characteristics of attributes. However, a large number of routing entries associated with multiple overlay networks makes Mercury less useful for P2P systems with resources that are characterized by many attributes (e.g., CASA and GENI). Moreover, highly skewed resource and query loads cannot be balanced by adjusting the position of nodes on the ring.

Figure 2.10(b) illustrates an alternative design based on a partitioned ring. LORM (Low-Overhead, Range-query, and Multi-attribute) [Sh07] assigns a separate segment of the ring to each attribute type. It is built on top of the Cycloid overlay [Sh06] (see Section 2.1.2). Attribute values are represented as a bit string where prefix bits represent the segment (indicate attribute type) and suffix bits represent the position within a segment (indicate attribute value). Suffix bits are generated by applying a Locality Preserving Hash (LPH) function to the attribute value. A LPH function maps nearby attribute values to neighboring nodes in the ring. This enables the resolution of range queries by forwarding to a series of nodes. An application searching for resources issues a separate sub-query for each segment according to the required set of attributes. Query results are later combined at the application using a join operation like in databases.

MADPastry (Mobile AD-hoc Pastry) [Za05] proposes a similar scheme for large-scale mobile ad-hoc networks. It partitions the Pastry hypercube [Ro01] (see Section 2.1.2) to preserve the physical locality of nodes, which is determined using a set of landmarks. A query is first sent to the local partition. If required resources are not found locally, the query is then forwarded to other partitions one at a time. Another alternative is to build a set of hypercubes for each geographic region and then connect them to form a backbone [De09]. As illustrated in Fig. 2.11, a subset of the peers that are in the same neighborhood forms a hypercube while retaining their physical locality. A backbone is then formed by interconnecting these hypercubes through gateway peers. Resources are advertised to both the local and remote hypercubes through the backbone. Similar to MADPastry, resources are first queried within the local hypercubes. Remote hypercubes are queried only when resources are not found locally. Dynamic load balancing is achieved by adjusting dimensions of the hypercube according to the query load. For example,

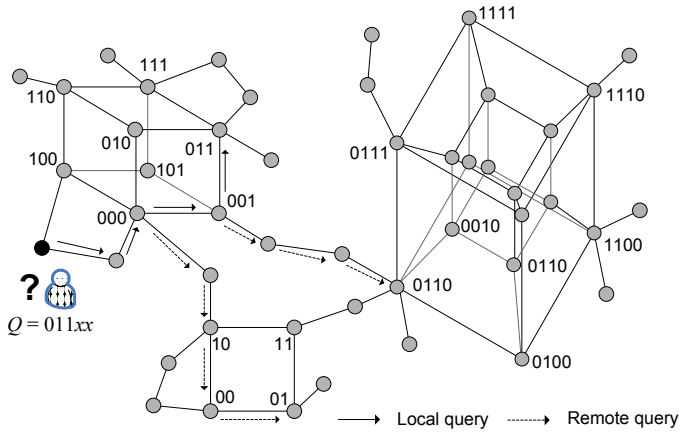


Figure 2.11 – Hypercubes connected to form a backbone.

when average query load of a hypercube exceeds a given threshold, the dimensions of the hypercube are increased and more neighboring peers are added to the hypercube to distribute the load. This approach is counterproductive, as popular queries will experience high latency because of the increased path length due to increase in hypercube dimensions.

LORM, MADPastry, and hypercube backbone maintain a much lower number of routing entries compared to Mercury. MADPastry and hypercube backbone also provide resource matching based on latency and hop count because of their ability to locate local resources. However, multiple sub-queries and their tendency to return a large number of unusable resources increase the lookup overhead in all three solutions.

MAAN (Multi-Attribute Addressable Network) [Ca04] proposes a SADQ mechanism for a ring (Fig. 2.10(c)). It maps attribute value of a resource to the same identifier space (i.e., ring) using a separate uniform LPH function for each attribute type. In addition to preserving the locality, uniform LPH functions also uniformly distribute the hash values across the ring. Use of an overlapped ring reduces the routing state and query overhead. However, uniform LPH fails to balance the load when the popularity of resource queries is skewed or there are many identical resources.

Figure 2.12 illustrates another design where RSs are mapped to points in a *d*-torus similar to CAN [Ra01]. Each dimension of the torus represents an attribute type. MURK (Multidimensional Rectangulation with Kd-trees) [Ga04] is one such solution that dynamically partitions the *d*-torus to many

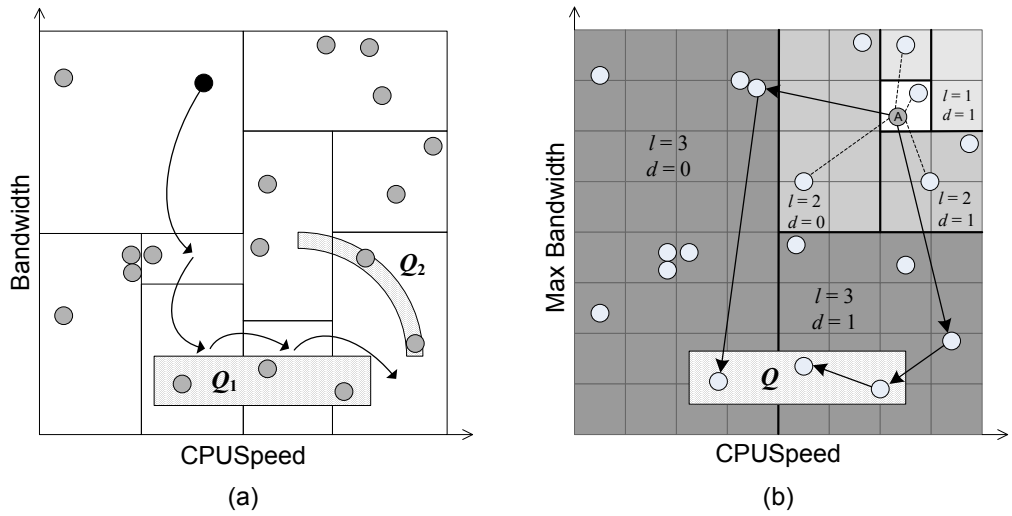


Figure 2.12 – 2-dimensional torus: (a) Multi-attribute space partitioned into a set of zones, a query is routed to all the zones that overlap with the query rectangle; (b) Multi-attribute space partitioned with respect to peer *A*, the query is routed to all the peers that overlap with the query rectangle, *l* – level, *d* – dimension [Co09b].

zones. Each node indexes all the RSs that map to its zone. MURK keeps track of these zones by organizing them in the form of a *k*-dimensional tree (*kd*-tree). Dynamic load balancing is achieved by splitting/aggregating zones based on the query load. A multi-attribute range query encloses a hyperrectangle (i.e., *d*-dimensional contiguous space) on the torus, e.g., Q_1 in Fig. 2.12(a). Queries are resolved using greedy forwarding. As a query propagates, each node reports matching resources to the application, as it is not straightforward to aggregate resources from multiple zones. Alternatively, MURK uses space-filling curves [Ja90, Or84] to map the *d*-torus to a ring by reducing its dimensionality. Though a ring enables a query to aggregate potential resources as it propagates, it introduces additional overhead as nearby resources on the *d*-torus are no longer mapped to a contiguous region on the ring.

The solutions presented above provide only the advertisement and selection of individual resources. SWORD [Al08] is a partitioned-ring-based architecture (see Fig. 2.10(b)) that also supports the selection of multiple resources and resource matching. A SWORD query can define required groups of resources, their inter-group and intra-group constraints, and penalties for not satisfying them. Because it is impractical to keep track of all the inter-resource relationships, SWORD cluster nodes into equivalence classes based on their Autonomous System (AS). A designated node from each AS keeps track of latency

and bandwidth relationships between two ASs. Resources are selected using either a SADQ (similar to MAAN) or multiple sub-queries (similar to LORM). The selected resources are then used to form a set of candidate groups based on the application requirements. Subsequently, resources in each candidate group are matched based on their inter-AS relationships. The groups are then ranked according to the extent they satisfy the application constraints and sent to the application enabling it to take the final decision on what group(s) to use. Though AS-based measurements provide a reasonable estimation of latency, such coarse-grained measurements do not work well for bandwidth and other complex inter-resource relationships.

Solutions discussed so far are applicable in semi-dynamic environments where resources do not change rapidly. Costa et al. argue that DHTs are inefficient and incapable of accurately maintaining the state in dynamic environments such as P2P clouds [Co09b]. Therefore, the authors propose to construct a resource-aware overlay in the form of a *d*-torus using only the static attributes that can be represented correctly (see Fig. 2.12(b)). The torus is recursively partitioned into a small set of hypercubes called *cells*. Queries are routed using pointers that each node keeps to one of the nodes in each cell. Nodes in the lowest-level cell are connected to each other. Gossiping is used to discover such nodes and their attribute values. A query is first routed to one of the lowest-level cells that overlap with the query hyperrectangle. Other overlapping cells are then recursively traversed using depth-first search until all the potential or a predefined number of resources are found. Dynamic attributes defined in a query are evaluated at individual peers. Therefore, [Co09b] can advertise, select, and bind resources. It also lessens load balancing and single points of failure problems in DHTs as resources are connected to each other according to their attribute values rather than indexed in a remote node. However, depth-first search increases the lookup overhead and latency particularly if queries are less selective. Moreover, gossiping does not guarantee discovery of a node in a sparse cell and all the nodes in the lowest-level cell.

Table 2.5 summarizes the different structured P2P solutions and all the solutions are compared in Table 2.6. There is no universal solution and each has its own distinct advantages and limitations. SADQ-based solutions have a low query resolution overhead. However, they could lead to unbalanced query load

Table 2.5 – Summary of structured P2P solutions.

Scheme	Architecture	Routing Mechanism	Lookup Overhead*	Routing Table Size*	Load Balancing
Mercury	Multiple rings	Successor & long distance links	$O(1/k \log_2 N)$	$k + 2$ per ring	Dynamic
LORM	Partitioned ring	Cycloid	$O(d)$	$O(1)$	Static
MADPastry	Partitioned ring (based on locality)	Pastry	$O(\log N)$	$O(\log l)$	Static
Hypercube backbone	Hypercube-based backbone	Hypercube	Local – $O(d)$ Remote – $O(D)$	$\Theta(d)$	Dynamic
MAAN	Single ring	Chord	$O(\log N + N r_{min})$	$O(\log N)$	Static
MURK	d-torus	CAN with long distance links	$O(\log_2 N)$	$2d + k$	Dynamic
SWORD	Partitioned ring, resource matching	Chord	$O(\log N + N r_{min}/d))$	$O(\log N)$	Static
Resource-aware d-torus	d-torus partitioned into cells	Links to peers in other cells	$O(N)$	$O(d)$	Static

* N – number of peers in overlay, k – number of long distance links, d – number of dimensions/attributes, D - network diameter, r_{min} – minimum range selectivity, l – number of landmarks

and inaccurate representation of dynamic resources due to the large number of replicas. Mercury, MAAN, LORM, and SWORD are extensible because they can add new attributes without requiring significant changes to the existing overlay. Except for SWORD, MADPastry, and Hypercube backbone, other solutions do not support resource matching. Resource-aware d-torus is the only solution that supports resource binding. Recently many other solutions have been proposed [Ra08] based on these alternative design choices. However, they also do not support resource matching and binding. Certain applications are able to compensate for lack of resources, e.g., distributed data fusion in CASA can compensate for lack of bandwidth between a processing and a storage node by processing data faster to accommodate the extra delay introduced while transferring data to the storage node. Such requirements can be represented by complex queries that are mapped to a polygon on the attribute space, e.g., Q_2 in Fig. 2.12(a). Resolving such a mapping is not straightforward and requires tight coordination between resource selection and matching. Given the limitations in existing solutions, there is still a need for a cohesive solution that can efficiently advertise, select, match, and bind resources.

Table 2.6 – Summary of all the P2P-based resource discovery solutions with respect key phases of resource discovery.

Scheme	Architecture	Advertise	Discover*	Select	Match*	Bind*
Flooding	Flood advertisements or queries	Yes	N/A	Guaranteed	When RSs are flooded	When queries are flooded
Gossiping	Agents share resource specifications they know	Yes	Yes	Low probability of success	Simple matching	No
Random walk	Agents carry resource specifications & queries	Yes	Yes	Moderate probability of success	Simple matching	When query agents are used
Superpeer	2-layer unstructured overlay	Yes	Yes	Guaranteed	Simple matching	Yes
Mercury	Multiple rings	Yes	N/A	Guaranteed	No	No
LORM	Partitioned ring	Yes	N/A	Guaranteed	No	No
MADPastry	Partitioned ring (based on locality)	To local & neighbor partitions	N/A	Guaranteed	Latency & hop count	No
Hypercube backbone	Hypercube-based backbone	To local & neighbor hypercubes	N/A	Guaranteed	Latency & hop count	No
MAAN	Single ring	Yes	N/A	Guaranteed	No	No
MURK	*d*-torus	Yes	N/A	Guaranteed	No	No
SWORD	Partitioned ring, resource matching	Yes	N/A	Guaranteed	Yes	No
Resource-aware *d*-torus	*d*-torus partitioned into cells	Static attributes only	N/A	Guaranteed	No	Yes

* N/A – Not applicable

2.4 Peer-to-Peer Communities

Recent data indicate the emergence of many small communities within a P2P system, with each community based on some common user interests [Zh10]. A *community* is a subset of peers that share some similarity in terms of resource semantics, geography, or organizational boundaries. Members of a community with common interests may or may not be aware of each other. It is known that peers have semantic relationships based on the type of content they frequently access [Ha06, Zh10]. For example, BitTorrent has many communities that are dedicated to music, movies, Linux distributions, games, etc. Users from the same country tend to access similar resources as well [Ha06, Kl04]. For example, for 60% of the files shared by eDonkey peers, more than 80% of their replicas were located in a single country [Ha06]. It was also observed that semantic and geographic similarities are more prominent for moderately popular files. Communities may also arise based on organizational boundaries, e.g., peers within an Autonomous System (AS), members of a professional organization, or a group of universities often forms a

community to share resources and limit unrelated external traffic. For example, BitTorrent has many private communities that users can only join through invitations [Me10]. We can further envision a distributed collection of large scientific databases such as Genome sequences, Geographic Information Systems (GIS), weather, census, and economic data that are accessed by various communities of users from academic, research, and commercial institutions.

It is well known that content popularity in P2P systems follows a Zipf's-like [Ad02] distribution [Ba11c, Kl04, Ra04, Ra07, Sr01]. Zipf's law says that the frequency of an event is inversely proportional to its rank in popularity, more formally:

$$f(r,\alpha,N) = \frac{1/r^{\alpha}}{\sum_{n=1}^{N} 1/n^{\alpha}} \tag{2.1}$$

where r is the rank, α ($\alpha > 0$) is the Zipf's parameter, and N is the total number of distinct events. When $\alpha = 1$ the distribution is called a Zipf's distribution otherwise it is called a Zipf's-like distribution. Popularity distribution is more skewed when α is large and/or N is small.

However, resources popularly shared within an individual community typically do not rank high in popularity in the context of the overall P2P system [Ba12e, Ha06] and often are inconspicuous in the system-wide behavior. Therefore, such communities are unable to benefit from various performance enhancements such as caching and replication that focus only on the most dominant or popular resources. However, the emerging technological trends such as social networking indicate that we will continue to see the emergence of a large number of small and diverse communities within large P2P systems. Future P2P architectures therefore should support such communities by providing customized services based on their distinct characteristics. Such architectures should allow the emergence, growth, existence, and disappearance of communities on a continual basis, while enabling them to be a part of a global community or a system. Conversely, the P2P system can significantly benefit by taking into account the characteristics and requirements of these communities.

2.5 Peer-to-Peer Caching

Caching is the most popular mechanism used to improve the lookup performance in P2P systems. Many structured and unstructured P2P systems (e.g., Freenet and CAN) utilize *passive caching* where peers keep track of the responses to their own queries hoping that they will be able to respond to future queries from others. Freenet caches query responses at the query initiator as well as at all the nodes along the path that the random walk traveled before finding the required resource. Passive caching is not that effective as the query responses are cached without considering their popularity or the overlay topology. Alternatively, *active caching* keeps track of query popularities and proactively cache resources or their contact details while taking into account how the queries are routed in a given overlay topology. Next, we discuss several caching solutions for unstructured (Section 2.5.1) and structured (2.5.2) P2P systems.

2.5.1 Unstructured Peer-to-Peer Solutions

In [Co02], it is proven that the expected lookup overhead in unstructured P2P systems is minimized when the number of cached copies of a resource is proportional to the square root of its popularity. In [Co02] and [Lv02], authors demonstrated that the square root allocation could be achieved by creating cache copies in proportion to the length of the random walk that was used to find the given resource. However, the authors do not discuss where those cache entries should be placed. An implementable solution based on the same approach is presented in [Mo05]. This allocation is valid only when the cache entries are automatically removed from the system due to constant peer churn and failure. In practice, however, some nodes tend to be active for a very long time [Ch02, Po05]. Therefore, some of the resources may be cached too much while tying up the caches in nodes. Both [Co02] and [Mo05] assume that a node can cache any number of resources and the query originator determines the resources to cache. In an alternative design, intermediate peers determine what to cache based on the queries that they forwarded but were not able to answer [So08]. Query success or failure is determined by forcing a query response to follow the reverse path. However, this doubles the lookup overhead. After determining the relative popularity of resources that a peer cannot answer, each peer tries to minimize the lookup cost by determining

the most suitable set of resources to maintain in its limited cache. This optimization problem is solved as a distributed knapsack problem that tries to maximize the lookup performance while minimizing the cache size. In spite of these efforts, even the most popular queries are unable to gain significant advantages from caching due to the randomness in unstructured overlay topologies and random walks.

2.5.2 Structured Peer-to-Peer Solutions

Beehive [Ra04] proposes an active caching/replication solution that achieves $O(1)$ mean look performance in structured P2P systems that utilize prefix-based routing (e.g., Pastry). A (*key*, *value*) pair is cached at an increasing set of levels in the key space starting from the root node (i.e., node that is responsible for indexing the *key*). Figure 2.13(a) illustrates a circular address space with three levels. Most popular resources are cached in all the nodes while moderately popular ones are cached at different levels in proportion to their popularity. For example, 3-hops are required to resolve a lookup query for key *0121* originating at node *Q* (Fig. 2.13(a)). When the key is cached at levels 0, 1, and 2, the query can be resolved within 0, 1, and 2 hops, respectively. Beehive assumes a Zipf's-like popularity distribution, and the Zipf's parameter and popularity of resources are determined using distributed statistics. Each node along a query path collects the statistics and forwards them to the root node of the given *key*. Each root node then solves an optimization problem to determine the caching level for each of the resources that

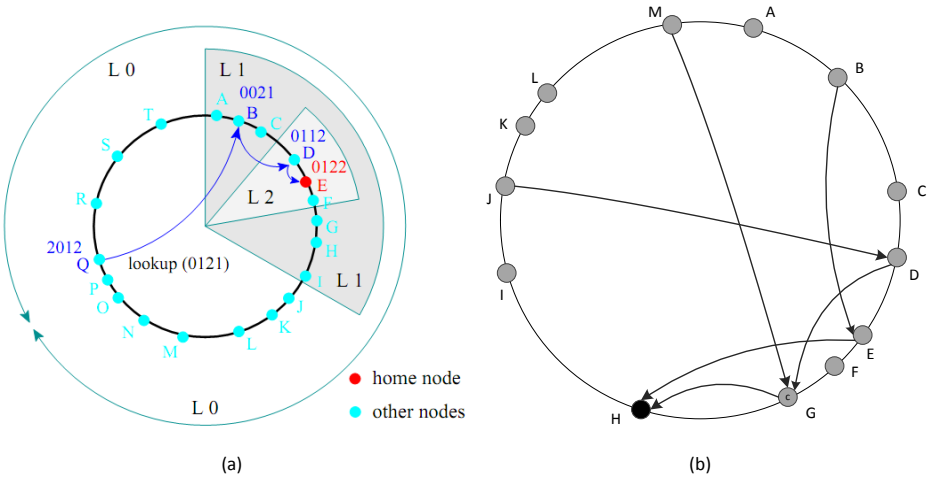


Figure 2.13 – Caching in structured P2P systems: (a) Beehive. Only three caching levels are shown [Ra04]; (b) PoPCache.

will result in a guaranteed mean lookup performance (for the entire P2P system) while minimizing the number cache entries. For example, Beehive (implemented on top of Chord) places the most popular set of resources on every node in the ring, second most popular set of resources on ½ of the nodes, third most popular set on ¼ of the nodes, and so on.

Beehive works with only the Zipf's-like popularity distributions and prefix-based routing. PoPCache [Ra07, Ra10] overcomes these limitations by placing cache entries based on the structure of the overlay topology (see Fig. 2.13(b)). When the popularity of a resource is relatively high, the root node first place cache entries at its predecessors (i.e., nodes that forward messages to it). When the popularity is even higher, cache entries are also placed at predecessors' predecessors. Most popular resources are cached at many nodes, but not necessarily in all the nodes, therefore PoPCache is more cache efficient than Beehive. For example, node *H* first places cache entries in predecessors *G* and *E*. If the resource is even more popular, cache entries are also placed at predecessors' predecessors (e.g., node *G* places cache entries at *D* and *M*). PoPCache also works with any DHT scheme and popularity distribution. It has been shown that the minimum mean lookup cost can be achieved by allocating the cache entries in proportion to the popularity of resources. Moreover, optimization problem can be also solved to minimize the number of cache entries.

Both Beehive and PoPCache force a large fraction of nodes to cache the globally popular resources regardless of their individual or community interests. Though Zipf's-like popularity distributions enable significant performance gain by caching few highly popular resources, diminishing return is gained with very large caches. This fact forces both the solutions to maintain a large number of cache entries while trying to provide a guaranteed mean look performance. Moreover, a guaranteed mean may not have much practical significance compared to a guaranteed distribution. Moreover, both solutions assume nodes have unlimited cache capacity. In spite of requiring large caches, such solutions are also inconsiderate of moderately popular resources. Furthermore, global popularity estimation is costly and error prone. Query arrivals in P2P systems show flash-crowd, diurnal, and seasonal effects [Ra04, Zh10].

Therefore, statistics such as periodic query counts or arrival rate estimates in Beehive and PoPCache are also inadequate to effectively decide when and what resources to cache.

2.6 Named Data Networking

Modern Internet users value the ability to access contents irrespective of their locations whereas the Internet was designed to facilitate end-to-end resource access. Conflict between the usage and design objectives has led to many issues such as location dependence, traffic aggregation, and security. Consequently, many clean-state designs for the Internet propose to access/route data based on their application-layer content names [Ja09a, Ko07, St02]. Named Data Networking (NDN) [Ja09a] (a.k.a. Content Centric Networking (CCN)) is gaining traction as one of the viable clean-state designs particularly in the presence of CCNx open source implementation [Palo]. NDN enables in-network caching, multicasting, duplicate message suppression, enhanced security, and mobility. When data are not already dispersed within the network, NDN delivers user queries to potential data sources enabling on demand data generation. In contrast, the majority of other content-naming solutions, e.g., [Ko07, St02], are based on DHTs that index only the pre-generated data. Moreover, NDN supports different levels of abstractions and incremental deployments ranging from overlay networks, content delivery networks, and small ISPs to eventual Internet-wide deployment.

Communication in NDN is receiver driven, i.e., the content consumer requests contents using their names. Hierarchical names are recommended due their ability to capture semantic relationships in data, aggregate names, and enforce security through the chain of trust. Typical hierarchical name could look like the following:

```
/youtube.com/sports/videos/football_finals.mpg/
```

NDN names can also refer to attributes and segments of a file. For example, the third segment of High Definition (HD) version of the same video can be named as:

```
/youtube.com/sports/videos/football_finals.mpg/HD/s3/
```

Names are also used to invoke remote methods and/or pass data. For example, following name may be used to request a temperature sensor in the machine room to report temperature readings every five minutes:

```
/machine_room/sensor123/temperature/sense/5min/
```

A consumer requests for contents by sending an *interest packet* with the name of the desired data. Interest packet is then routed using the name looking for a node having a copy of the desired data or for a source capable of generating that data. A node having the data or capable of generating them, responds with a *data packet*. At most one data packet is transmitted in response to an interest packet hence provides flow control and one-to-one delivery. A data packet is self-authenticating as the data generator signs it. Idempotent, self-identifying, and self-authenticating properties of NDN data packets enable the same packet to be shared across many consumers hence also support one-to-many delivery without transmitting a new data packet from the data source. Large data that do not fit into a single data packet are segmented and each segment is retrieved by issuing a separate interest packet. The default size of a data packet in CCNx is 4 KB.

Each NDN node uses three tables namely Content Store (CS), Pending Interest Table (PIT), and Forwarding Information Base (FIB) (see Fig. 2.14). CS caches data packets that go through a node enabling the node to locally respond to future interest packets with the same name. PIT keeps track of outstanding interest packets that are waiting for a corresponding data packet and suppress duplicate interest packets with the same name. PIT entries consist of a set of tuples (*name*, [$face_1$, $face_2$, ..., $face_i$]), where $face_i$ is the NDN equivalent of a network interface that an interest packet arrived from. This also enables multicasting. Typically, PIT entries timeout to keep the size of the table bounded. For example, CCNx recommends a timeout of 4 seconds. FIB is a name-based forwarding table that is used to determine where and how to forward an interest packet when it cannot be handled locally. FIB entries consist of a set of tuples (*name*, [$face_1$, $face_2$, ..., $face_i$]), where $face_i$ is a face that an interest packet can be forwarded to. An interest packet may be forwarded to multiple faces sequentially or simultaneously based on predefined routing policies. Various name-based routing algorithms may be used to update the FIB entries.

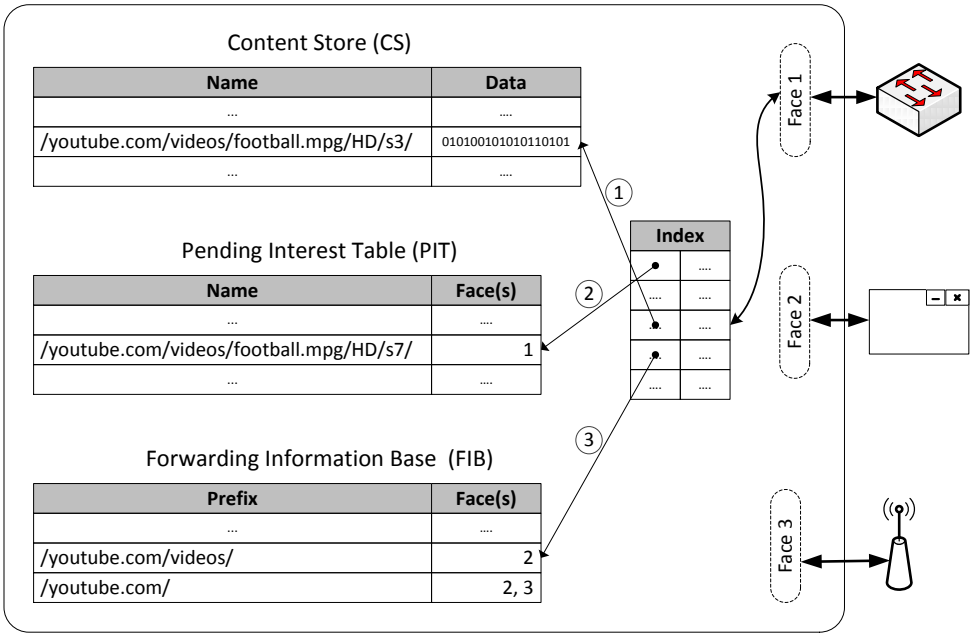


Figure 2.14 – Forwarding tables and their interactions within an NDN node.

When an interest packet arrives at an NDN node, first the CS is checked for a cached data packet with the same name (see Fig. 2.14). If such a packet exists, node transmits the data packet and drops the interest packet. If not, the node then checks the PIT for a pending interest with the same name. If such an interest exists, node appends the ingress *face* to the PIT and then drops the packet. This suppresses duplicate interest packets and reduces the bandwidth requirement by preventing multiple data packets from being retrieved for the same name. If a PIT entry does not exist, a new tuple (*name*, [*face*]) is added to the PIT. Interest packet is then forwarded to a face(s), found from the FIB, that might lead to a node with the desired data. When the data are found or generated by the source, data packet is sent to the face that forwarded the interest packet. Once a data packet arrives, intermediate nodes use the PIT entries to determine the faces that the packet needs to be forwarded back. Thus, PIT entries leave a trail of "bread crumbs" that is used to forward the data back to the consumer(s) using the reverse path. A copy of the data packet is also saved in the CS and subsequently the corresponding PIT entry is removed. A data packet that does not have a corresponding PIT entry is dropped. This may happen when the PIT entry has timed out or an attacker is trying to flood a node with data packets. Data consumers are expected to resend interest packets, if the data do not arrive within the desired time. Main criticism for NDN is the

large number of names that needs to be maintained in all three tables as possible number of unique names is in the order of trillions or more. These constraints are expected to be overcome with the advancement of hardware and firmware technologies.

2.7 Summary

There is a gap between the existing solutions and the requirements of collaborative P2P systems. The majority of existing resource discovery solutions focus on individual resources and even the ones that support some form of resource aggregation are primitive. These solutions are not designed for latency sensitive applications and do not support resource compensation. Thus, no single solution is capable of providing all the desirable features of collaborative P2P systems. In addition, there is a tremendous opportunity to exploit the power of P2P communities not only by aggregating their resources but also by providing a customized service to its members. NDN enables names-based access of contents while reducing the bandwidth requirements and enhancing mobility, resilience, and security which are key requirements of collaborative P2P systems. The goal of this research is to come up with a set of solutions that address the inadequacies in existing solutions and to exploit emerging technologies to make collaborative P2P systems a reality.

Chapter 3

PROBLEM STATEMENT

Future collaborative Peer-to-Peer (P2P) applications will look for diverse peers that could bring in unique resources and capabilities to a virtual community thereby empowering it to engage in greater tasks beyond what can be accomplished by individual peers, yet are beneficial to all of them. The ability to aggregate an optimum set of resources is a fundamental requirement for collaborative P2P applications. Yet, it is nontrivial to timely discover, group, and utilize heterogeneous and dynamic resources that are distributed. The majority of existing solutions for resource discovery focus only on discovering individual resources [Bh04, Ca04, Co09b, Kw10, Sh07, Ta08]. Even the solutions that support some form of resource grouping are primitive [Al08, Ke06]. However, to realize the true potential of collaborative P2P systems, it becomes necessary to be able to aggregate a group(s) of heterogeneous, multi-attribute, dynamic, and distributed resources as and when needed. Moreover, in the absence of data and understanding of the characteristics of real workloads, existing solutions are developed under many simplifying assumptions. Therefore, these solutions perform poorly when applied to real workloads that exhibit more complex characteristics [Ba11e, Ba12a, Ba12f]. A better alternative would be to first understand the complex characteristics of real workloads and then use the learned behavior to develop solutions that can better match the requirements of real-world systems. The goal of this research is to develop better resource discovery and distributed data fusion solutions and necessary tools that can aggregate groups of heterogeneous, dynamic, and multi-attribute resources in collaborative P2P systems, while bridging the gap by characterizing real-world resources, queries, and user behavior.

Section 3.1 presents the motivation of this problem. Research goals and objectives are presented in Sections 3.2 and 3.3, respectively. Solution approach is presented in Section 3.4.

3.1 Motivation

In contrast to conventional P2P systems that focus on pair-wise interactions among nodes with similar resources (e.g., file sharing, see Fig. 3.1(a)), we envision the emergence of collaborative P2P applications that thrive on the interactions among groups of diverse and distributed resources (see Fig. 3.1(b)). Collaborative P2P systems are applicable in a wide variety of contexts such as Distributed Collaborative Adaptive Sensing (DCAS) [Ku06, Le12, Mc05, Mc09], grid [Ca04, Sh07], cloud [Ar09], and opportunistic computing [Co10], Internet of Things [Pf11], mobile social networks (Section 2.2.4), and emergency management. These systems are expected to share a variety of *resources* such as processor cycles, storage capacity, network bandwidth, sensors/actuators, special hardware, middleware, scientific algorithms, application software, services (e.g., web services and spawning nodes in a cloud), and data to not only consume a variety of contents but also to generate, modify, and manage those contents.

We briefly discuss one representative application to illustrate the salient features and characteristics of collaborative P2P systems. A more detailed discussion and additional examples are given in Section 2.2. Collaborative Adaptive Sensing of the Atmosphere (CASA) [Mc05, Mc09] is a DCAS system based on a dense network of weather radars that operate collaboratively and adaptively to detect hazardous atmospheric conditions such as tornados and severe storms. Collaborative P2P data fusion provides a scalable implementation choice for real-time radar data fusion in CASA, as multiple data volumes are constantly being generated, processed, and pushed and pulled among groups of radars, processing, and

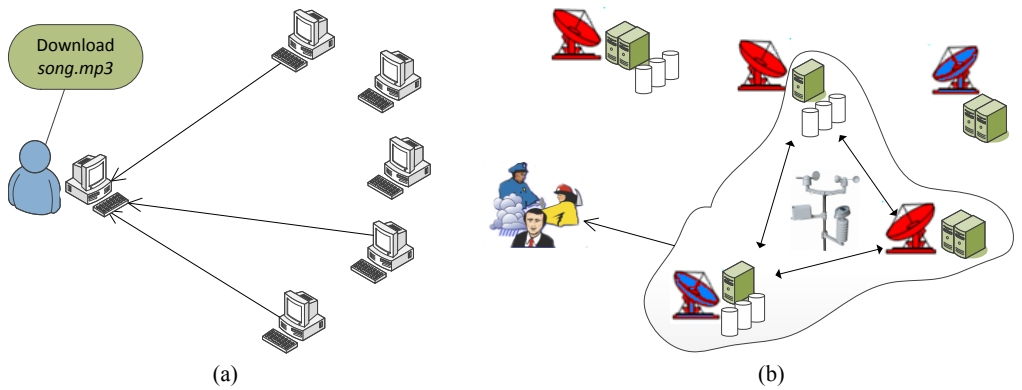


Figure 3.1 – Interaction among peers: (a) Individual interactions between pairs of peers in file sharing; (b) Interactions among a group of diverse peers in weather monitoring and forecasting.

storage nodes (see Fig. 3.1(b)). Radars, weather stations, processing, and storage elements involved in tracking a particular weather event continue to change as the weather event migrates in both time and space. Thus, new groups of resources need to be aggregated as and when needed. Moreover, certain rare but severe weather events require specific algorithms (e.g., signal processing and forecasting) and more computing, storage, and bandwidth resources to track and forecast/nowcast about their behavior. It is neither feasible nor economical to provision resources for such rare peak demands everywhere in the CASA system. Instead, a collaborative P2P system can exploit the temporal and spatial diversity of weather events to aggregate underutilized resources from anywhere in the system. Therefore, a collaborative P2P approach is appropriate for large-scale CASA deployments, as it can timely deliver and process radar data while satisfying the dynamic resource demands and enhancing the overall resource utilization.

Collaborative P2P applications need to be able to group the resources in a timely manner to meet the performance and Quality of Service (QoS) requirements. Thus, discovering and aggregating an optimum set of resources is a fundamental requirement for collaborative P2P applications. Yet, it is nontrivial to discover, group, and utilize heterogeneous and dynamic resources that are distributed. The majority of existing solutions for resource discovery focus only on individual resources [Bh04, Ca04, Co09b, Kw10, Sh07, Ta08]. Even the solutions that support some form of resource grouping are primitive [Al08, Ke06]. They are not designed for latency sensitive collaborative P2P applications such as CASA and P2P clouds (Section 2.2.3). In the absence of data and understanding of the characteristics of real workloads, designs and evaluations of these solutions have relied on many simplifying assumptions. For example, independent and identically distributed (i.i.d.) attributes [Bh04, Co09b, Sh07], uniform/Zipf's distribution of attribute values [Bh04, Co09b, Sh07], attributes having a large number of potential values [Sh09], and queries specifying a large number of attributes and a small range of attribute values [Al08, Bh04, Ca04, Co09b, Sh07]. Moreover, the cost of updating dynamic attributes is ignored. Such assumptions affect both the designs and performance analysis, and consequently the applicability of solutions under real workloads. Therefore, it is imperative to understand the resource and query characteristics of real-world systems. Once the characteristics are known, it is useful to evaluate the fundamental design choices in

P2P-based resource discovery with respect to the learned characteristics. Such an analysis will not only provide a better understanding of the applicability of existing solutions but also helps to identify the best practices for resource aggregation in emerging collaborative P2P systems.

Large P2P systems exhibit the presence of communities based on semantic, geographic, or organizational interests of users [Ha06, Zh10]. Resources commonly shared within individual communities are in general relatively less popular and inconspicuous in the system-wide behavior [Ba12e]. Therefore, most communities are unable to benefit significantly from performance enhancement schemes such as caching and replication that focus only on the most dominant queries within the entire P2P system. Conversely, creating topologically isolated clusters of communities is also not desirable, as a substantial fraction of queries in production systems tend to be for resources outside of a particular community [Ba11c, Ba12e]. Emerging technological trends such as social networking indicate that we will continue to see the emergence of a large number of small and diverse communities within large P2P systems. Future P2P architectures therefore should support such communities by providing customized services based on their distinct characteristics. Such architectures should allow the emergence, growth, existence, and disappearance of communities on a continual basis, while enabling them to be a part of a global community or a system. Conversely, the P2P system can significantly benefit by taking into account the characteristics and requirements of these communities. Hence, it is important to develop resource discovery solutions that are aware of communities' interests, adaptive, as well as message and storage efficient.

Modern Internet users value the ability to access contents irrespective of its location, whereas the Internet was designed to facilitate end-to-end resource access. Conflict between the usage and design objectives has led to many issues such as location dependence, traffic aggregation, and security. Consequently, many clean-state designs for the Internet propose to access/route data based on their content names [Ja09a, Ko07, St02]. DCAS systems, including current CASA deployments, typically bind data to the sensor(s) that generated them by assigning data names based on the sensor identifier. Alternatively, end users in many cases are interested in data related to a particular weather event in a given geographic area of interest, and are not concerned about which sensor(s) generated the data. Therefore, naming data

based on the source/sensor creates a conflict similar to that in the current Internet. It also limits the ability to utilize the spatial and temporal locality in user interests and redundant sensors in DCAS systems to enhance the performance and reduce the resource requirements of distributed data fusion. Emerging name-based content access solutions such as Named Data Networking (NDN) [Ja09a] can be exploited to reduce the bandwidth requirements, and to enhance the resource utilization, resilience, and security of DCAS systems. For example, a network of CASA radars may name the data based on their geographic location and weather feature (e.g., reflectivity of clouds or wind velocity) independent of the radar(s) that generated them. This enables the end users to specify an area of interest for a particular weather feature while being oblivious to the placement of CASA radars and associated computing facilities. Conversely, CASA can use its knowledge about the underlying system to decide the best radar scanning and data processing strategies during times of heavy usage and partial system failures. Currently, NDN has to be deployed as an overlay network due to the absence of an Internet-wide deployment. However, use of overlay networks provides added benefits such as the ability to deploy multiple and application-specific naming conventions, application-specific routing mechanisms, fault tolerance, better QoS, and in-network data fusion [Ba12h, Le12]. Therefore, it is important to explore the ways to make DCAS systems more efficient, scalable, and robust by benefiting from the advantages of both NDN and overlay networks.

3.2 Research Goals

The goal of this research is to develop scalable and efficient, resource discovery and distributed data fusion solutions and necessary tools that can timely aggregate groups of heterogeneous, dynamic, and multi-attribute resources in collaborative P2P systems, while characterizing real-world resources, queries, and user behavior. Such solutions will satisfy the need of collaborative P2P applications to aggregate diverse and distributed groups of resources as and when needed. Moreover, they will empower peers/applications to engage in greater tasks beyond the capabilities of an individual peer. Consequently, collaborative applications will be able to achieve their performance and QoS requirements by identifying

the most appropriate groups of resources from a vast pool of underutilized/unused resources that are dispersed within the collaborative P2P system.

The distributed, autonomous, and collaborative nature of peers and multifaceted scalability requirements demand for a distributed solution. It should support the aggregation of diverse resources by: (1) efficiently *advertising* all the resources and their current state, (2) *discovering* potentially useful resources, (3) *selecting* resources that satisfy application requirements, (4) *matching* applications and resources according to their constraints, and (5) *binding* resources and applications to ensure guaranteed service. Implementation of these phases is nontrivial because heterogeneous, dynamic, and multi-attribute resources and their diverse resource relationships make collaboration complex. Overall solution needs to be resource and query aware, adaptive, fault tolerant, and need to satisfy multiple aspects of scalability such as query resolution latency, number of resource attributes, messages, and routing-table entries. Timely aggregation of complex resources is becoming increasingly necessary even in conventional grid, desktop grid, volunteer computing, and cloud computing as these systems continue to grow and users understand how to develop parallel applications that can work with multiple resources. Thus, intended solutions are also applicable in grid and cloud computing which are special cases of the problem.

3.3 Research Objectives

The goals of this dissertation span three key areas of research related to multi-attribute resource discovery, single-attribute resource discovery and distributed caching, and demonstrating the applicability of NDN for distributed multi-sensor data fusion leading to five objectives as follows:

1. *Characterize real workloads and understand their implications on P2P-based resource discovery* – Develop an analytical model to capture the overall cost of resource discovery/aggregation in terms of overlay messages involved in advertising multi-attribute resources and querying them. Determine the nature of model parameters under different real-world systems. Qualitatively and quantitatively evaluate the fundamental design choices for P2P-based resource discovery using the learned characteristics.

2. *Develop related tools* – Develop tools to collect the data necessary to achieve the first objective and to generate synthetic resource and query traces for large-scale performance studies. To evaluate the applications and protocols for scalability beyond what is available, it becomes necessary to consider overlay network configurations with higher number of nodes and attributes. Yet, it is still necessary to adhere to the statistical characteristics, dependencies, and temporal patterns exhibited by real-world systems. It is impractical to gather large traces with sufficient resolution and duration even for existing systems. Therefore, our objective is to gather representative information about the traces collected under the first objective and then generate synthetic trace arrays of larger dimensionality in number and time.
3. *Resource and query aware algorithms* – Overlay topology should be formed in such a manner that simplifies resource advertising, querying, matching, and binding. The majority of existing solutions first form a topology and then index resources and issue queries. This creates a mismatch between the overlay, underlay, and resources being advertised and queried. Instead, we believe that better performance can be gained by building a novel topology that reflects the available resources and their demands (i.e., queries). Therefore, resource and query aware, topology formation and resource aggregation algorithms need to be developed.
4. *Provide customized services to P2P communities* – Develop solutions that provide customized services to different P2P communities while enhancing both the communitywide and system-wide performance. Such solutions can be used to enhance the P2P resource advertising, lookup performance, streaming, and service availability. New solutions need to be developed as prior work either focused on the system-wide behavior or clustered peers into communities according to a given similarity metric. Developing relevant analytical models is also of interest.
5. *Named data networking for distributed data fusion* – Demonstrate the applicability of NDN for distributed data fusion in DCAS systems by developing a proof of concept, distributed, multi-user, multi-application, and multi-sensor data fusion solution based on CASA. NDN needs to be deployed on top of an overlay network because of the lack of Internet-wide deployment. Extend

NDN to support multiple naming conventions, application-specific caching solutions, subscription schemes, and data delivery mechanisms that are necessary for DCAS systems.

3.4 Solution Approach

We first identify seven phases of resource collaboration that future collaborative P2P systems need to satisfy, which are depicted in Fig. 3.2 [Ba12b]. These separate phases are identified for the sake of conceptual understanding while an actual implementation may combine multiple phases together. Specific systems may skip some of them depending on the application requirements. Thus, complexities and implementation details on each of these phases significantly vary. The seven phases are as follows:

1. *Advertise* – Each node/peer advertises its resources, their capabilities, and usage constraints using one or more *resource specifications* (RSs) (see Section 4.2). An example multi-attribute RS of a computing node may look like the following:

$$RS = \begin{pmatrix} CPUSpeed = 2.0\,GHz,\ Architecture = \times 86, CPUFree = 53\%,\ MemoryFree = 1071MB, \\ OS = Linux\_2.6.31, Available = 12:00\,am - 6:00\,am,\ CPUUtilization \le 60\%, \\ UseBy = "Friends",\ Licenses = 2 \end{pmatrix}$$

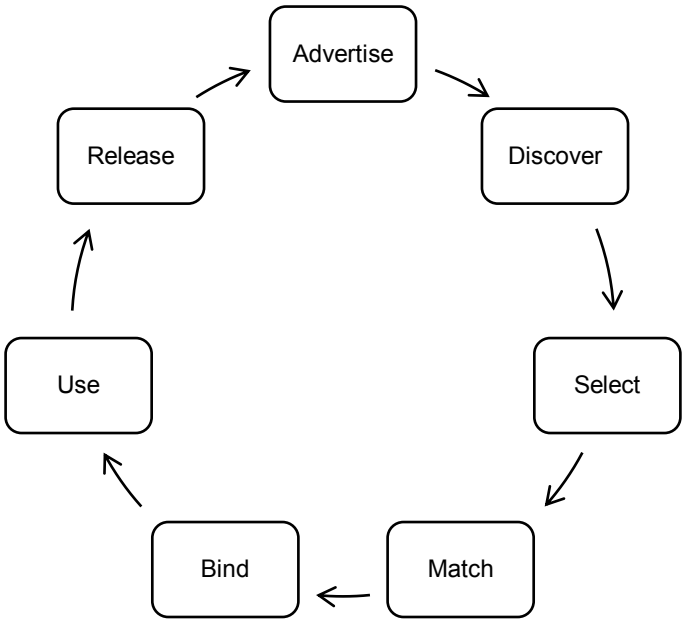


Figure 3.2 – Phases in resource collaboration.

Similar RSs can be given for other resources such as storage nodes and sensors. A node may either hold on to its RSs hoping others will discover its resources by sending probe messages, or explicitly send/advertise its specifications to a central database, neighboring nodes, or nodes in Distributed Hash Tables (DHTs).

2. *Discover* – Nodes may send probing messages to proactively discover and build a local repository of useful RSs, particularly if specifications are unadvertised. Such a collection of RSs can speed up the query resolution and may be used to keep track of inter-resource relationships such as latency, bandwidth, and trust.
3. *Select* – Select a group(s) of resources that satisfies the given application requirements. The application requirements are typically specified using multi-attribute range queries that list the required number of resources, one or more attributes, and ranges of attribute values. An example query searching for six computing nodes may look like the following (see Section 4.2):

$$Q = \begin{pmatrix} Resources = 6, CPUSpeed \in [2.0GHz, MAX], MemoryFree \in [256MB, 512MB], \\ Architecture = \times 86, OS = "Linux\_2.6.32" \text{ OR } "UNIX\_11.12" \end{pmatrix}$$

4. *Match* – Not all the combinations of resources satisfying a resource query may be suitable or capable of working together. It is important to take into account how two resources relate and interact with each other (e.g., bandwidth/latency between different pairs of peers), to ensure that they can satisfy the resource and application constraints. For example, data processing in CASA, GENI, or P2P clouds (Section 2.2.3) may not only require processing nodes and storage nodes but also require certain latency/bandwidth bounds among different pairs of such resources. For instance, a node with somewhat limited processing and storage capabilities may be a better match for a certain application than two nodes (one with high processing capabilities and the other with large storage), if they are separated by a low-bandwidth or a heavily loaded interconnect. Similarly, mobile P2P (Section 2.2.4) and ad-hoc networks may need to ascertain whether two resources are nearby to avoid a certain service provider, minimize latency, or reduce packet loss. Moreover, social relationships between users may affect the willingness to share their resources.

5. *Bind* – Once a subset of resources that match the application requirements are identified, it is necessary to ensure that the selected resources are available for use. Due to churn or failures, the resources found may not be available by the time the application is ready to utilize them. The same resource may also be under consideration by other applications. Hence, a binding has to be established between the resources and the application trying to use them. Binding is particularly important in guaranteed service environments like CASA and GENI to achieve the desired QoS and real-time requirements.
6. *Use* – Utilize the best subset of available resources that satisfy the application requirements and constraints to carry out the application tasks for which resources were acquired. Resource usage and interaction patterns are application specific.
7. *Release* – Release resources when application demand decreases, the task is completed, or binding expires, whichever occurs first. The resource release patterns are also application dependent.

Applications may cycle through these phases as and when they need additional resources to fulfill increased/varying application demands, to take advantage of new resources, or to overcome limitations caused by resources that fail or leave the P2P network. It is possible for different applications and multiple instances of an application to be in different phases at any given time. Thus, the P2P resource allocation solutions have to continually adapt and evolve based on the changing resource availability and dynamic application demands in a robust and scalable manner. The term *resource discovery* typically refers to the first three phases [Al08, Bh04, Ca04, Co09b, Ga04a, Sh07, Su08b, Ta08] whereas the term *resource aggregation* refers to the overall process of advertising, discovering, selecting, matching, and binding resources. Our goal is to characterize the real-world resources and queries, evaluate the fundamental design choices for resource discovery using the learned behavior, and develop new resource and query aware resource aggregation solutions in line with these key phases.

We use existing datasets and collect our own ones to understand the resource and query characteristics (e.g., distributions, correlations, popularities, and temporal evolutions), application requirements,

similarities among P2P communities, and content access patterns of P2P users. Findings from these datasets are used as the basis while evaluating the existing resource discovery solutions, developing tools to generate large synthetic traces of resources and queries, and designing and evaluating new solutions. When applicable, a combination of analytical models and simulation-based studies are used to evaluate the proposed solutions. CASA is the primary application that motivated the proposed research hence algorithms will be mainly designed, developed, and validated in the context of large-scale CASA networks.

Chapter 4

MULTI-ATTRIBUTE RESOURCE AND QUERY CHARACTERISTICS OF REAL-WORLD SYSTEMS AND IMPLICATIONS ON P2P-BASED RESOURCE DISCOVERY

Collaborative Peer-to-Peer (P2P), grid, and cloud computing systems rely on Resource Discovery (RD) solutions to aggregate groups of heterogeneous, multi-attribute, and dynamic resources that are distributed. However, very little is known about the specific characteristics of real-world resources and queries, and their impact on P2P-based RD. We analyze the characteristics of real-world resources and queries, and then use the learned characteristics to evaluate the fundamental design choices for P2P-based RD. First, an equation for the cost of multi-attribute resource advertising and querying is derived. Second, the nature of parameters in the equation under different systems is determined by analyzing datasets from PlanetLab, SETI@home, EGI grid, and a distributed campus computing facility. These datasets exhibit several noteworthy features that affect the performance. These findings are then used to qualitatively and quantitatively evaluate the fundamental design choices for multi-attribute RD based on the cost of advertising/querying, load balancing, and routing table size. Compared to uniform queries, real-world queries are relatively easier to resolve using unstructured, superpeer, and single-attribute-dominated-query-based structured P2P solutions (Section 2.3). However, they introduce significant load balancing issues in all the designs. Cost of RD in structured P2P systems is effectively $O(N)$ as most range queries are less specific. The implications of our findings for improving P2P-based RD solutions are also discussed.

Section 4.1 presents the introduction and motivation. The equation for the cost of RD is derived in Section 4.2. Four datasets are described in Section 4.3 and their characteristics are analyzed in Section 4.4. Different resource advertising and querying options are qualitatively evaluated in Section 4.5. Sections 4.6 and 4.7 present the simulation setup and performance analysis, respectively. Implications and

best practices for future RD solutions are discussed in Section 4.8. Concluding remarks are presented in Section 4.9.

4.1 Introduction

Collaborative Adaptive Sensing of the Atmosphere (CASA), Global Environment for Network Innovations (GENI), P2P clouds, and mobile social networks (see Sections 2.2 and 3.1) depend on some form of collaboration among resources. Therefore, these applications need the ability to locate and aggregate groups of complex resources as and when needed to meet the performance and Quality of Service (QoS) requirements. P2P-based distributed RD is a natural fit for collaborative applications and further enhances their scalability, load balancing, and robustness. Many P2P-based RD solutions are also proposed for conventional applications such as grid, desktop grid, and cloud computing [Ca04, Co09b, Kw10, Sh07], as timely aggregation of complex and distributed resources is becoming increasingly necessary due to the proliferation of parallel applications that utilize multiple and distributed resources. Yet it is nontrivial to discover and utilize heterogeneous, multi-attribute, and dynamic resources that are distributed. Nevertheless, a good RD solution should satisfy several key phases (Fig. 3.2) where they (1) efficiently *advertise* all the resources and their state, (2) *query* to find resources that satisfy application requirements, (3) *match* resources according to application and resource constraints, and (4) *bind* resources and applications to ensure guaranteed service [Ba12b]. Moreover, solutions need to be adaptive, fault tolerant, and should satisfy multiple aspects of scalability such as query resolution latency, number of attributes, number of messages, and routing state. The focus of this chapter is on the first two phases namely *advertising* and *querying*.

Significant progress has been made in multi-attribute RD in grid computing [Ca04, Sh07] and P2P [Al08, Bh04, Co09b, Kw10, Ta08]. However, compared to single-attribute P2P systems such as file sharing, formal characterization of real world, multi-attribute resources and queries received attention only recently [Ba11e, Ba12f, He12]. Characteristics of static attributes of nodes from a set of volunteer computing systems (e.g., SETI@home, Einstein@home, and World Community Grid) are presented in

[He12]. Our preliminary work in [Ba11e] presented both the static and dynamic resource and query characteristics of PlanetLab [Ki11, Pa06] and SETI@home [An06] nodes. In the absence of information, data, and understanding of real life, multi-attribute resource and query characteristics, designs and evaluations of existing RD solutions have relied on many simplifying assumptions. For example, they assume independent and identically distributed (i.i.d.) attributes, uniform or Zipf's distribution of all the resources/queries [Co09b, Sh07, Sh09], and queries specifying a large number of attributes and a small range of attribute values [Ca04, Sh07]. Moreover, the cost of updating dynamic attributes is ignored [Al08, Bh04, Ca04, Sh07, Sh09]. Such assumptions affect both the designs and performance analyses of RD solutions, and consequently their applicability under real workloads [Ba12a]. For example, performance analysis in [Sh09] is limited to point queries in structured P2P systems and extensively relied on the aforementioned assumptions. Furthermore, direct comparison of these solutions is impractical due to the diversity of designs. Therefore, a formal and detailed comparison of fundamental design choices for P2P-based RD with respect to the behavior learned from actual systems is needed. Such an analysis will not only provide a better understanding of the applicability of different design choices but also helps to identify the best practices for resource aggregation in emerging collaborative P2P systems.

We qualitatively and quantitatively evaluate the fundamental design choices for P2P-based, multi-attribute RD using the characteristics learned from real-world systems. Our main contributions are:

- Development of an approximate equation that captures the overall cost of RD in terms of overlay messages. Such an equation helps to identify important parameters that affect the cost of RD and performance bounds for specific RD solutions.
- Characteristics of multi-attribute resources from PlanetLab, SETI@home, EGI grid, and a campus network are then analyzed to determine the nature of parameters that affect the cost of RD in different systems. PlanetLab data are also used to analyze the multi-attribute query characteristics. Such an understanding is useful in designing and evaluating RD solutions and job schedulers.

- A representative subset of design choices based on centralized, unstructured, superpeer, and structured P2P architectures is then qualitatively and quantitatively evaluated using the learned characteristics. Seven design choices are evaluated based on the cost of advertising and querying resources, routing table size, load balancing, and scalability. Simulation-based study using a node and query trace from PlanetLab is used for the quantitative evaluation. These findings help to identify the best practices for resource aggregation.
- The implications of our findings for improving and developing new P2P-based RD solutions are also discussed.

Our findings show attributes of most resources are highly skewed and correlated. Attribute values have different marginal distributions and most of them are too complex/skewed to be described using any of the standard probability distributions. Ones that fit a known distribution satisfy Gaussian, generalized extreme value, and generalized Pareto distributions. While resources are characterized by many attributes, most attributes have only a few distinct set of values. Attribute values changed at different rates and few attributes changed rapidly. Attribute values specified in queries are skewed however do not satisfy a Zipf's-like distribution. Queries are less specific where each query tends to specify only a small subset of the available attributes and large ranges of attribute values. Simulation-based analysis indicates that real-world queries are relatively easier to resolve using unstructured, superpeer, and single-attribute-dominated-query-based structured P2P architectures compared to uniform queries used in conventional studies. Cost of RD in ring-based structured P2P systems is effectively $O(N)$, where N is the number of nodes in the overlay, as most range queries specify large ranges of attribute values. This also increases the overhead of sub-query-based structured P2P solutions by an order of magnitude. The cost of advertising dynamic attributes is significant and increases with the number of attributes hence should not be ignored in performance studies. Furthermore, all the design choices are prone to significant load balancing issues where few nodes are mainly involved in answering the majority of queries and indexing resources. Therefore, existing design choices are applicable only under very specific conditions and perform poorly under realistic workloads. Our work differs from [He12] in several aspects. In [He12], the evolution of static

attributes of multiple volunteer computing systems over five years is analyzed and a model is presented to forecast how the composition of hardware resources will evolve in the future. Conversely, we analyze both the static and dynamic attributes and short-term (ranging from a few minutes to days) changes in dynamic attributes. We also consider four different computing environments. Multi-attribute query characteristics are also analyzed using PlanetLab data. Several other attempts to model grid computing resources are presented in [Ke04, Lu03, Su08a]; however, they also do not capture the dynamic attributes. Understanding the characteristics of dynamic resources and queries across different systems is needed for the design, validation, and performance analysis of RD solutions and job schedulers as their performance is impacted by short-term changes in resources. Our analysis of PlanetLab resources and queries are complementary to the work in [Ki11], which analyzes the behavior of workloads in PlanetLab.

4.2 Cost of Advertising and Querying Resources

We consider the resources that are characterized by multiple attributes, e.g., a computing node is characterized by its CPU speed, memory, operating system, etc. Similar to advertising file names in file sharing, these attributes are advertised to a central database, neighboring nodes, or nodes in Distributed Hash Tables (DHTs). All the attributes of a resource may be advertised together as a vector or separately as a set of scalar values. We analyze the cost of RD in terms of the number of overlay messages involved in advertising such resources and querying them. Our objective is to develop a relatively simple equation to capture the cost of RD. In this process, we identify a set of parameters and their relationship to the cost of advertising and querying. In Section 4.4, we determine the nature of these parameters in real-world systems. Then in Section 4.5, the equation is extended to represent the cost of RD in centralized, unstructured, and structured P2P-based architectures and their performance bounds.

Consider a P2P overlay used to index (i.e., keep track of) resources, where resources are advertised to a distributed set of nodes and queries are resolved by contacting those nodes. Let **R** be the set of resources in the system and **A** be the set of attributes used to describe those resources. We use bold face

symbols to refer to a set and the corresponding italic symbol to refers to its cardinality, e.g., $R = |\mathbf{R}|$. List of symbols is given in Table 4.1. Each resource $r \in \mathbf{R}$ is defined as follows:

$$r = \left(a_1 = v_1, a_2 = v_2, \ldots, a_i = v_i\right) \tag{4.1}$$

Each attribute $a_i \in \mathbf{A}$ has a corresponding value $v_i \in \mathbf{D_i}$ that belongs to a given domain $\mathbf{D_i}$. $\mathbf{D_i}$'s are typically bounded, may be continuous or discrete, or correspond to a set of categories/names. For example, free CPU is continuous, number of CPU cores is discrete, operating system is a category, and a file is represented by a name. A multi-attribute Resource Specification (RS) of a computing node with such a set of attributes may look like the following:

$$RS = \begin{pmatrix} CPUSpeed = 2.4GHz, NumCores = 2, CPUFree = 53\%, \\ MemoryFree = 1071MB, OS = Linux\_2.6.31 \end{pmatrix}$$

Similarly, a radar may be described by its sensing capabilities (e.g., Doppler radar), location, sensor range, and sensing frequency.

Attribute values are classified as *static* (e.g., CPU speed, total memory, operating system, and sensor location) and *dynamic* (e.g., free CPU, memory, bandwidth, and sensing frequency). Let $\mathbf{a_r} \subseteq \mathbf{A}$ be

Table 4.1 – List of symbols.

Symbol	Name
$\mathbf{a_r}$	Attribute set of resource r
$\mathbf{a_r^s}$ / $\mathbf{a_r^d}$	Set of static/dynamic attributes in resource r
$\mathbf{a_q}$	Attribute set of query q
$\mathbf{A}$/A	Set/number of distinct attributes in the system
$\mathbf{A^s}$ / $\mathbf{A^d}$	Set of static/dynamic attributes in the system
c_r^i	Number of copies of i-th attribute of resource r
$\mathbf{D_i}$	Domain of i-th attribute
$h^i_{Advertise}$ / h^i_{Query}	Hops to advertise/query i-th attribute
l_i	Lower bound of i-th attribute
m_q	Desired number of resources in query q
N	Number of nodes in system
$Q(t)$	Set of queries issued in time interval t
$\mathbf{R}$/R	Set/number of resources in the system
t	Time
u_i	Upper bound of i-th attribute
v_i	Value of i-th attribute
λ_r^i	Rate of change of i-th attribute in resource r

the collection of a_i's that is used to describe r. Let $\mathbf{A^s}$ and $\mathbf{A^d}$ be the set of static and dynamic attributes in the system, respectively. $\mathbf{A} = \mathbf{A^s} \cup \mathbf{A^d}$. Similarly, let $\mathbf{a_r^s}$ and $\mathbf{a_r^d}$ be the subsets of static and dynamic attributes of $\mathbf{a_r}$. Dynamic attributes need to be re-advertised whenever their values/states change. Let λ_r^i be the rate at which the change in i-th attribute of $\mathbf{a_r}$ occurs. The cost of advertising dynamic attributes is proportional to their rate of change. In some cases, multiple copies of a_i are placed in different databases, neighboring nodes, or DHTs. Let c_r^i be the number of copies of i-th attribute of $\mathbf{a_r}$.

We are interested in the number of messages that is either sent or forwarded within the overlay network to advertise and query resources. Cost of acknowledgement messages is ignored. Typically, the time required to resolve a query within a node is lower compared to the inter-node communication time. Therefore, if we assume that each overlay link has approximately the same latency, then the query resolution latency can be also represented using the number of overlay hops. Assuming each attribute a_i is advertised separately, the total cost of advertising r is:

$$C_{Advertise}^{r} = \sum_{i \in \mathbf{a_r}} c_r^i h_{Advertise}^i + \sum_{j \in \mathbf{a_r}} c_r^j h_{Advertise}^j \lambda_r^j t = \sum_{i \in \mathbf{a_r}} c_r^i h_{Advertise}^i \left(1 + \lambda_r^i t\right) \tag{4.2}$$

where $h_{Advertise}^i$ is the number of overlay hops that the advertising message corresponding to the i-th attribute travels. First summation captures the initial cost of advertising all the attributes. Second summation captures the recurrent cost of advertising dynamic attributes within time t. Resources need to reenter the system after churn or failure. Moreover, resources indexed in unstructured P2P systems and DHTs typically expire after a predefined timeout. In either case, resource attributes need to be re-advertised. Therefore, even the static attributes can be interpreted as behaving like dynamic attributes, but with much lower turnover. Hence, λ_r^i should be defined considering the rate of change and both the available and unavailable time of r. λ_r^i is lower for static attributes compared to that for dynamic attributes.

Let $\mathbf{Q}(t)$ be the set of all the queries issued within the time interval t. A multi-attribute range query $q \in \mathbf{Q}(t)$ is defined as follows:

$$q = \left(m_q, a_1 \in [l_1, u_1], a_2 \in [l_2, u_2], \ldots, a_i \in [l_i, u_i]\right) \tag{4.3}$$

where, $m_q \in \mathbf{Z}^+$ specifies the required number of resources and $a_i \in [l_i, u_i]$ specifies the desired range of attribute values (l_i and u_i are lower and upper bounds, respectively). A query that requests for five computing nodes with a given CPU speed, free memory, and operating system may look like the following:

$$q = \left(m_q = 5, CPUSpeed \in [2.0GHz, MAX], MemoryFree \in [256MB, 512MB], OS = "Linux\_2.6.32"\right)$$

where MAX_i is the upper bound of the domain $\mathbf{D_i}$. Similarly, a query for a set of radar may look like the following:

$$q = \begin{pmatrix} m_q = 3, Range \in [20km, 50\,km], LocationLati \in [6N, 12N], \\ LocationLongi \in [26W, 51W], Type = "Doppler" \end{pmatrix}$$

The set of attributes specified in a query ($\mathbf{a_q}$) may contain only a subset of the attributes that are used to describe resources (i.e., $\mathbf{a_q} \subseteq \mathbf{A}$). When $l_i = u_i, \forall\, a_i \in \mathbf{a_q}$, q is referred to as a *point query*. In practice, attributes in a query may specify a mixture of point and range values. Unspecified attributes are considered as "don't care". q is referred to as a *single-attribute query* when $a_q = 1$ and it is referred to as a *multi-attribute query* when $a_q > 1$. In practice, it may be necessary to find/discover more than m_q resources, as some of the selected resources may not be available by the time they are required for use due to resource churn/failure. Moreover, RD systems may not index all the attributes used to describe a resource, e.g., only the static attributes are indexed in [Co09b]. In such cases, unadvertised attributes have to be validated by directly contacting the resources. Contacting the resource is also important when a binding has to be established between the resource and the application planning to use it (see Section 3.4). Let $f(m_q, \mathbf{R}, \mathbf{A_{Index}})$ indicates the cost associated with these complex scenarios. $\mathbf{A_{Index}} \subseteq \mathbf{A}$ is the subset of attributes indexed in the P2P overlay. Then the cost of a point query can be given as:

$$C^q_{Query-Point} = \sum_{i \in \mathbf{a_q}} h^i_{Query} + f\left(m_q, \mathbf{R}, \mathbf{A_{Index}}\right) \tag{4.4}$$

h^i_{Query} is the number of overlay hops required to reach the node that indexes the value of i-th attribute defined in q. $f(.)$ captures the cost of directly contacting resources to either validate the attributes that are not

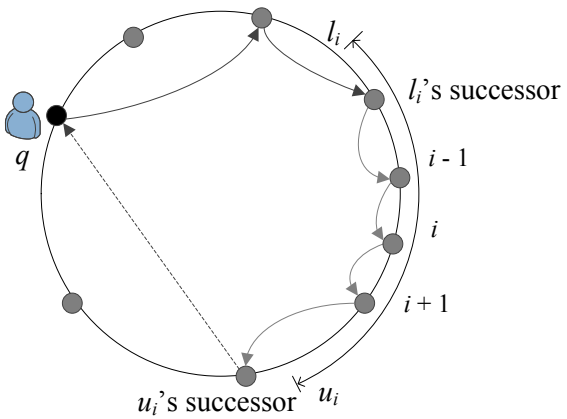


Figure 4.1 – Range query resolution on a ring-like overlay network.

advertised or establish a binding between a resource and the application. After reaching the first node, queries that specify a range of attribute values need to be forwarded further. For example, Fig. 4.1 illustrates how a query will be initially forwarded to the node that indexes the lower bound l_i on a ring-like overlay network. It will then be forwarded to the node responsible for indexing the upper bound u_i through a series of successors (see Fig. 4.1). As in [Al08, Ca04, Sh07], we initially assume the range of attribute values [l_i, u_i] to be uniformly distributed within the overlay. Therefore, the number of additional nodes visited by a range query is proportional to the range ($u_i - l_i$) and number of nodes (N) in the overlay, and is inversely proportional to the size of domain D_i (i.e., range or number of all possible attribute values). Then (4.4) can be extended to indicate the overall cost of a range query as follows:

$$C^q_{Query} = \sum_{i \in \mathbf{a_q}} \left(h^i_{Query} + \left\lceil \frac{(u_i - l_i)}{D_i} N \right\rceil - 1 \right) + f\left(m_q, \mathbf{R}, \mathbf{A_{Index}}\right) \tag{4.5}$$

Then the overall cost of RD is:

$$C_{Total} = \sum_{r \in \mathbf{R}} C^r_{Advertise} + \sum_{q \in \mathbf{Q(t)}} C^q_{Query} \tag{4.6}$$

4.3 Datasets

We use four datasets from PlanetLab, SETI@home, EGI grid, and our campus to identify the nature of parameters that affect the cost of RD. The set of resource attributes that are used in the analysis are described first. Then details on traces used in this study are presented.

4.3.1 Node Model

Though resources are characterized by many static, dynamic, and categorical attributes only a subset of them are essential in RD. Therefore, we consider 11 representative attributes that are essential to characterize a typical node useful in collaborative P2P, grid, and cloud computing systems.

1. *CPUSpeed* – Processor clock speed in GHz. Provides insight on the relative computing power of a node.
2. *CPUArchi* – CPU architecture, e.g., ×86 and SPARC.
3. *NumCores* – Number of processor cores in a node. Indicates how much parallelism in processing is possible. Some processors may count hardware-level threads as separate cores.
4. *CPUFree* – (100% – *CPU utilization*). Indicates to what extent the CPU(s) is available for processing. When multiple cores are available, the average value is given.
5. *CPULoad* – Number of active processes competing or waiting for CPU. It is typically represented as one minute (*1MinLoad*), five minute (*5MinLoad*), and 15 minute (*15MinLoad*) exponentially weighted moving averages of the number of competing processes. *CPULoad* indicates how long a user process has to wait for CPU. Both *CPUFree* and *CPULoad* are complementary to each other, as a large *CPULoad* does not necessarily mean high CPU utilization (e.g., processes could be blocked for I/O).
6. *MemSize* – Size of volatile memory in GB.
7. *MemFree* – Free user-level memory as a percentage (*MemFree%*) or in GB (*MemFree*). Indicates how much memory is available for user processes. Linux ignores the amount of memory consumed by the operating system when determining *MemFree*. Therefore, *MemSize* × *MemFree%* does not indicate the actual amount of free memory in GB. Thus, some systems track both parameters.
8. *DiskFree* – Free disk space in GB.

9. *TxRate* – Average transmission rate in Kbps. In conjunction with the bandwidth limit specified by most nodes, it provides insight on the amount of available bandwidth. It may also give insight on how much bandwidth will be available if an application uses the node exclusively.
10. *RxRate* – Average receive rate in Kbps.
11. *OS* – Type of operating system and version.

CPUSpeed, *NumCores*, and *MemSize* are static, *CPUArchi* and *OS* are categorical, and rest of the attributes is dynamic. RD solutions and job scheduling algorithms for latency sensitive applications are typically interested only in short-term trends. Therefore, we capture statistical characteristics that are valid for several minutes to few weeks.

4.3.2 PlanetLab

PlanetLab [Ki11, Pa06] is a global research network that supports the development of new network services. It provides a versatile platform for testing distributed applications and protocols by aggregating a globally distributed set of nodes. PlanetLab reflects many characteristics of Internet-based distributed systems such as heterogeneity, multiple end users, dynamic nodes, and global presence. Hence, it is being used to evaluate many preliminary P2P protocols and applications. We analyzed data from two PlanetLab tools, CoMon [Ki11, Pa06] and SWORD [Al08], to determine multi-attribute resource and query characteristics. CoMon is a node and slice monitoring system that provides a snapshot of static and dynamic attributes of all the nodes every five minutes. Table 4.2 summarizes the traces used in this study. SWORD is a multi-attribute RD tool for PlanetLab. It enables users to query for multiple groups of nodes

Table 4.2 – Summary of traces.

	CoMon	SETI@home	GCO	CSU	SWORD
No of nodes/queries	~650	234,421	205	387	915
Total no of attributes	64	34	87	27	49
Attributes useful in RD	46	25	17	17	49
Static attributes	12	21	6	8	8
Dynamic attributes	34	4	11	9	41
Sampling interval	5 min	Vary (hours to days)	1 min	1 min	Not tracked
Date(s)	2011/2/1 – 2011/2/14	Active as of 2012/4/30	2012/4/23 – 2012/5/6	2011/12/1 – 2011/12/14	2010/3/12 – 2010/7/20 & 2010/10/2 – 2012/6/5

while specifying inter-group and intra-group latency and bandwidth constraints. Though it was originally designed as a P2P solution, its current deployment is centralized and does not support latency and bandwidth constraints due lack of such data from PlanetLab nodes. Though we were able to collect only 915 queries over the last two years (Table 4.2) due to the light usage and server failure, the dataset provides a unique insight into real world, multi-attribute, and multi-resource queries that are unknown.

4.3.3 SETI@home

BOINC [An09] is a volunteer computing platform (a.k.a. desktop grid) that remotely executes jobs using idle computing resources. SETI@home is one of the largest BOINC deployments and it utilizes a much larger and diverse set of nodes compared to other datasets. While a detailed presentation of attribute values of SETI nodes is given in [An06, He12], their characteristics are not analyzed in the context of RD systems. Therefore, we briefly analyze the distribution of resource attributes collected from SETI@home website [An09]. Except for *DiskFree*, BOINC collects only the static attributes of a node. Attribute values are collected only when a node contacts a BOINC server therefore time between two samples vary from several hours to a few days. Hence, *DiskFree* is not sampled periodically. BOINC also collects CPU performance in terms of Dhrystone (integer) and Whetstone (floating-point) benchmarks. BOINC is a pull-based system where nodes request jobs based on their hardware resources hence do not support a resource query mechanism.

4.3.4 EGI Grid

European Grid Infrastructure (EGI) aggregates multiple grid computing sites across Europe [Ne11]. Data related to job submission, workloads, and computing and storage nodes from EGI sites are collected and published through the Grid Observatory [Ge11a]. We use node traces from Green Computing Observatory (GCO) [Ge11b], a recently launched sub-project within the Grid Observatory, that currently collects static and dynamic attributes of ~200 computing nodes with over 1,000 CPU cores at every one minute. Though the number of nodes in the dataset is much smaller compared to the size of the EGI

grid, GCO dataset provides a unique view of grid computing resources due to its high-resolution sampling in both time and number of attributes. No data are available on specific queries that may have been used to select those nodes for processing.

4.3.5 Campus Dataset

We also collected a dataset within our campus, Colorado State University (CSU). Subset of PCs and servers within the Department of Computer Science and Engineering Faculty was sampled every one minute. The dataset includes both Linux and Windows nodes and reflects an enterprise computing environment. Resource characteristics of such environments are becoming important as we are seeing the emergence of applications such as P2P clouds that aggregate residual computing resources from enterprise environments and homes [Br09]. No explicit query mechanism is provided and users are free to use any unoccupied PC in a lab. Server *CPULoad*s are published on the web enabling students and researchers to pick unloaded servers manually.

4.4 Resource and Query Characteristics

We now describe the characteristic of resources and queries, and observed behavior of variables/parameters identified in Section 4.2. Our discussion is primarily based on the characteristics of PlanetLab as we have both the resource and query datasets. Specific characteristics of other datasets are presented to illustrate the commonalities and differences across different systems.

4.4.1 Resource Characteristics

Figures 4.2 (a) and (b) show the distribution of *CPUSpeed* of PlanetLab and SETI@home nodes. Both distributions can be approximated using Gaussian distributions. Due to many identical nodes, *CPUSpeed* of GCO (Fig. 4.2(c)) and CSU (Fig. 4.2(d)) nodes were highly skewed and did not fit a standard distribution. 68 (11%) PlanetLab nodes had identical static attributes such as *CPUSpeed*, *NumCores*, *MemSize*, and *DiskSize*. This may have been caused by a donation of a set of similar machines to multiple

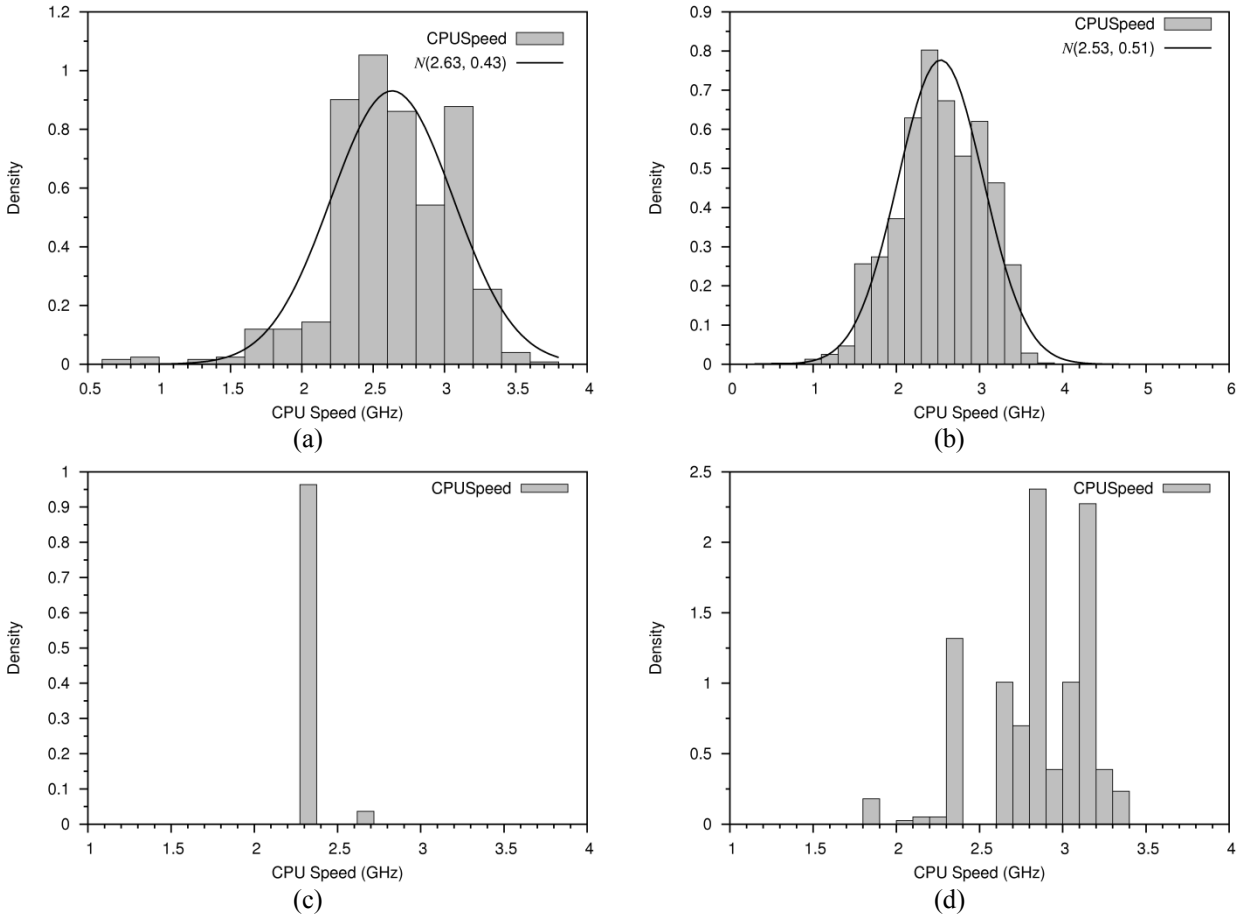


Figure 4.2 – Distribution of CPU speed: (a) PlanetLab; (b) SETI@home; (c) GCO; (d) CSU.

sites. Most sites also had few identical hosts. This is the case even in grid, cloud, and enterprise computing environments where sites may simultaneously deploy or upgrade to a similar set of machines. For example, the largest set of identical nodes in CSU and GCO datasets corresponded to 14% and 95% of the nodes, respectively. Nodes in student labs are typically homogeneous. For example, the four largest sets of identical nodes corresponded to 45% of the nodes in the CSU dataset. Due to such large sets of homogeneous nodes and discrete nature of attribute values, *NumCores* (see Fig. 4.3) and *MemSize* (see Fig. 4.4) of nodes in all the systems do not fit standard distributions. One may argue that we can still attempt to fit data to a set of well-known distributions by applying more complex sub-sampling techniques, rounding attribute values (e.g., rounding *MemSize* to the nearest power of two), or discarding some values (e.g., discarding *NumCores* values that are not powers of two). While we do discard outliers, other techniques

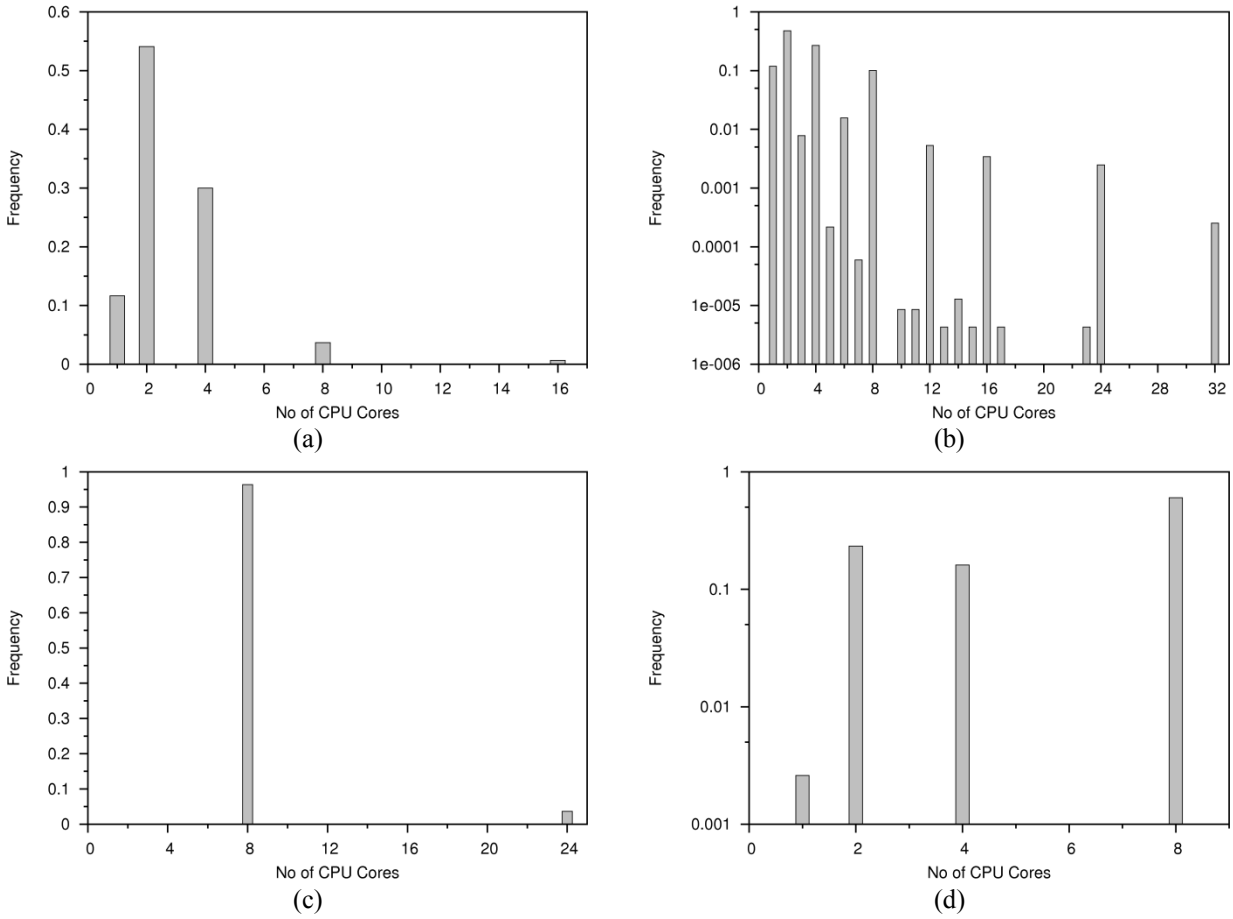


Figure 4.3 – Distribution of number of CPU cores: (a) PlanetLab; (b) SETI@home; (c) GCO; (d) CSU.

are of little value as our objective is to understand the true distributions and behavior of attributes but not to generate synthetic data using them. Such techniques are applied in [He12] for SETI nodes, which are more heterogeneous, as the authors' objective was to generate synthetic data. However, these techniques do not apply well to PlanetLab, GCO, and CSU datasets as they consist of multiple sets of homogeneous nodes. If the objective is to generate synthetic data, in Chapter 5 we demonstrate that it is more accurate to summarize the data using tables of empirical cumulative distributions (ECDF) and then use empirical copulas to generate synthetic data as copulas preserve the complex multivariate distributions and correlations among attributes.

Figure 4.5 shows the variation in average *CPUFree* and *MemFree* values of all the nodes with time. It can be seen that both PlanetLab and GCO experience idle and busy periods. Average resource utilization of GCO nodes (lower average *CPUFree* and *MemFree*) is higher than the other two systems.

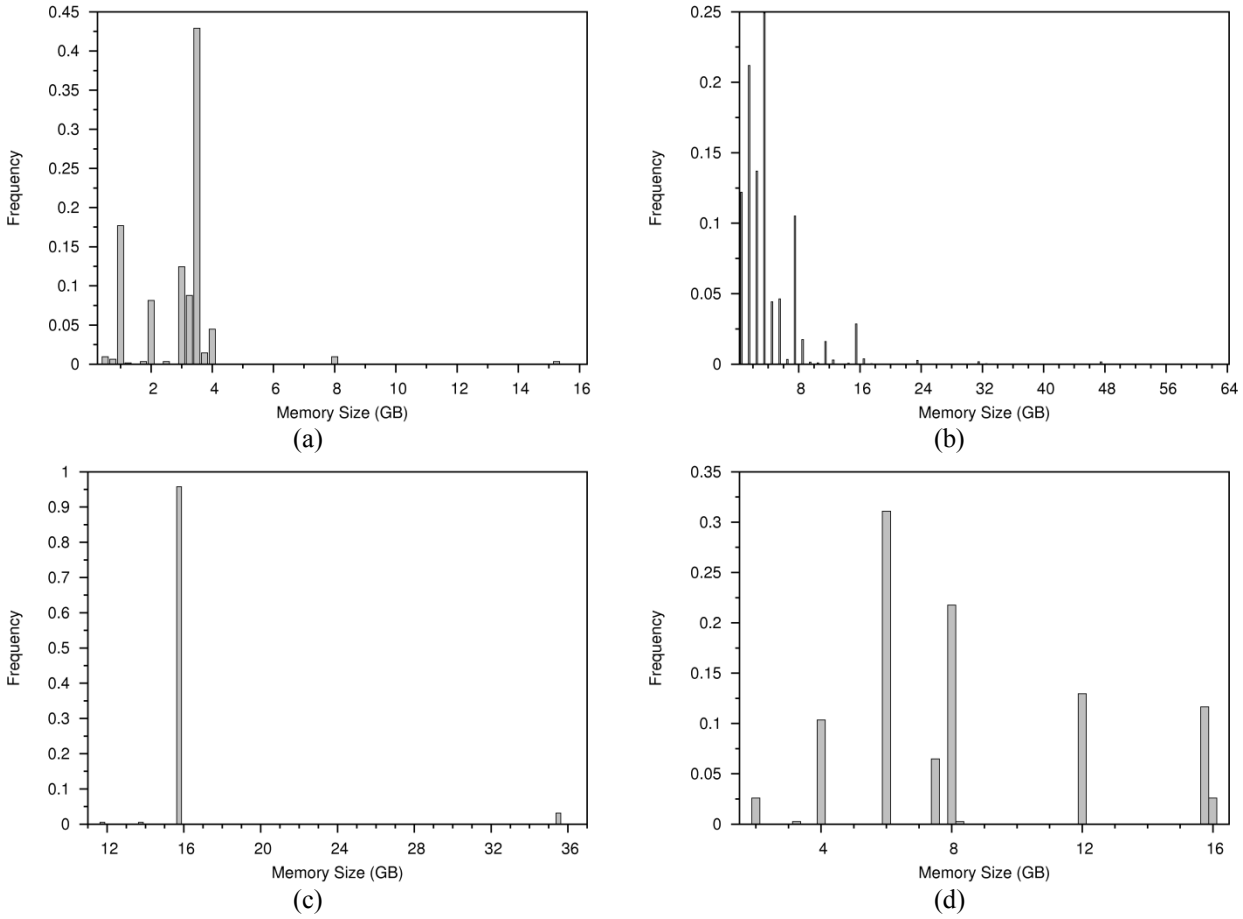


Figure 4.4 – Distribution of memory size: (a) PlanetLab; (b) SETI@home; (c) GCO; (d) CSU.

For example, Fig. 4.6 illustrates the distribution of *CPUFree* and *MemFree* during the peak time of each system. Large fractions of PlanetLab and CSU nodes were idle (high *CPUFree*) even during the peak time whereas GCO nodes were heavily utilized throughout the day. For example, 37% and 71% of the PlanetLab nodes had over 95% of *CPUFree* and 2 GB of *MemFree*, respectively. Similarly, 92% and 97% of the CSU nodes had the same amounts of *CPUFree* and *MemFree*. Even though students used the nodes during peak hours, their workloads do not fully utilize the capabilities of the nodes. However, only 0.5% and 46% of the GCO nodes had similar amounts of *CPUFree* and *MemFree*. Due to the highly skewed distributions, instantaneous values (at a given time *t*) of dynamic attributes do not fit standard distributions. Except for SETI nodes, *DiskFree* also had a similar behavior in other three datasets. SETI nodes had a skewed but monotonically decreasing distribution that can be approximated by a Generalized Pareto Distribution (GPD). Attributes that satisfy known probability distributions are listed in Table 4.3.

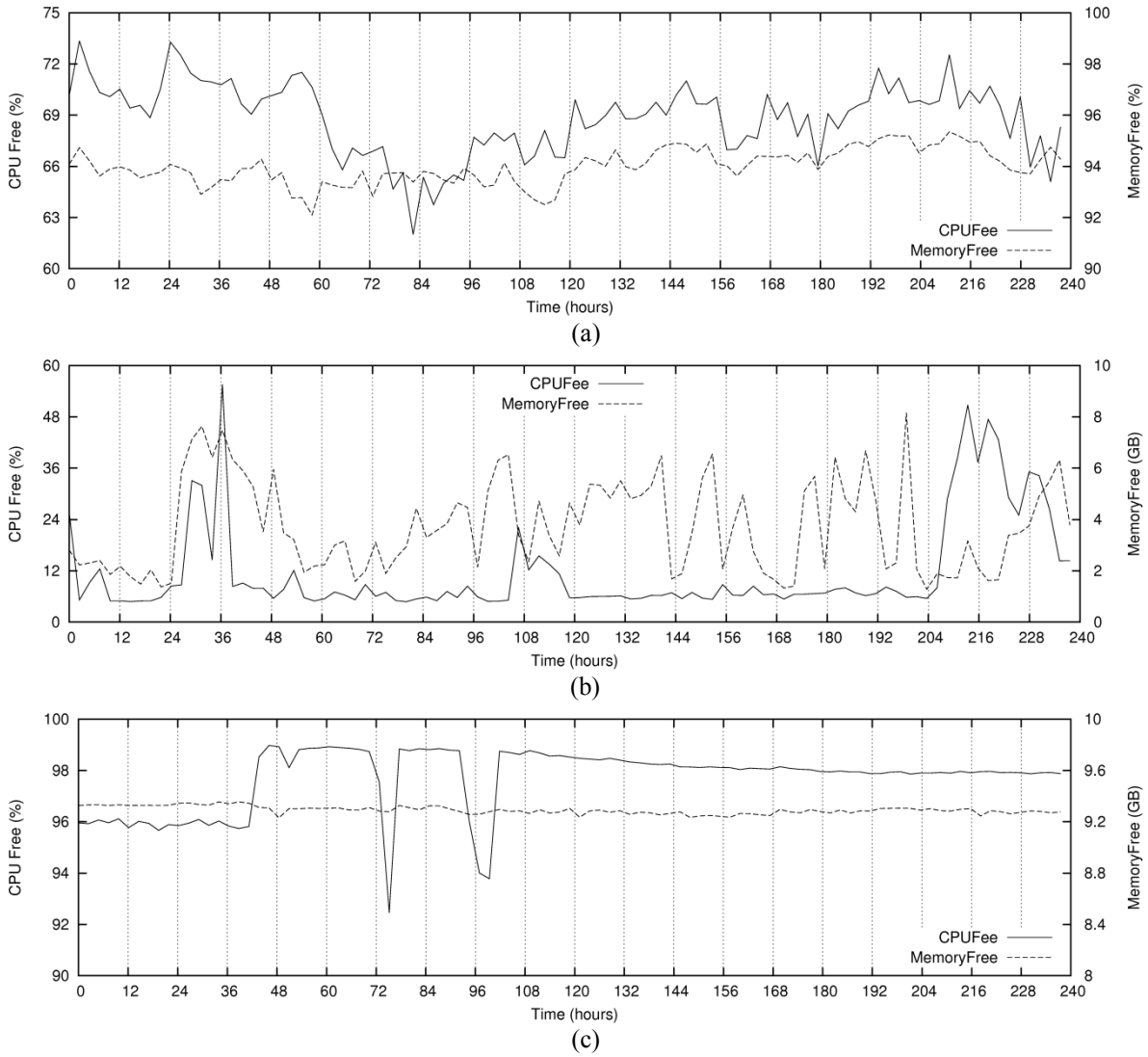


Figure 4.5 – Average resource utilizations of all the nodes with time: (a) PlanetLab; (b) GCO; (c) CSU.

Figure 4.7 shows the distribution of *TxRate* of PlanetLab and SETI nodes, which can be approximated using the Generalized Extreme Value (GEV) distribution. *RxRate* of those nodes can be also approximated using the GEV distribution. Average *TxRate* of SETI nodes is an order of magnitude lower than the average *RxRate* (see Table 4.3). This may be a consequence of the asymmetric bandwidth availability in volunteer nodes. *TxRate* and *RxRate* of GCO nodes did not fit standard distributions as they were skewed. While it was possible to approximate the distribution of *1MinLoad* using GPD, other datasets and *5MinLoad* and *15MinLoad* do not fit standard distributions. It was also observed that integer and floating-point performance of SETI nodes could be approximated by a Weibull distribution. We further observed

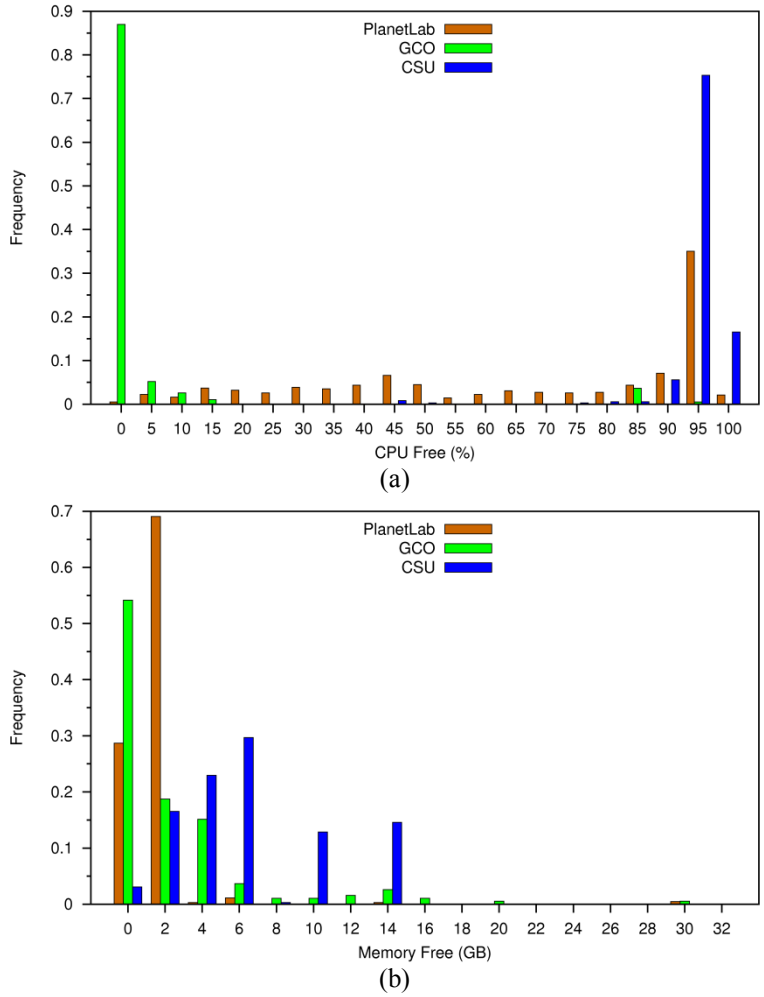


Figure 4.6 – Distribution of dynamic attributes during peak times: (a) Free CPU; (b) Free memory.

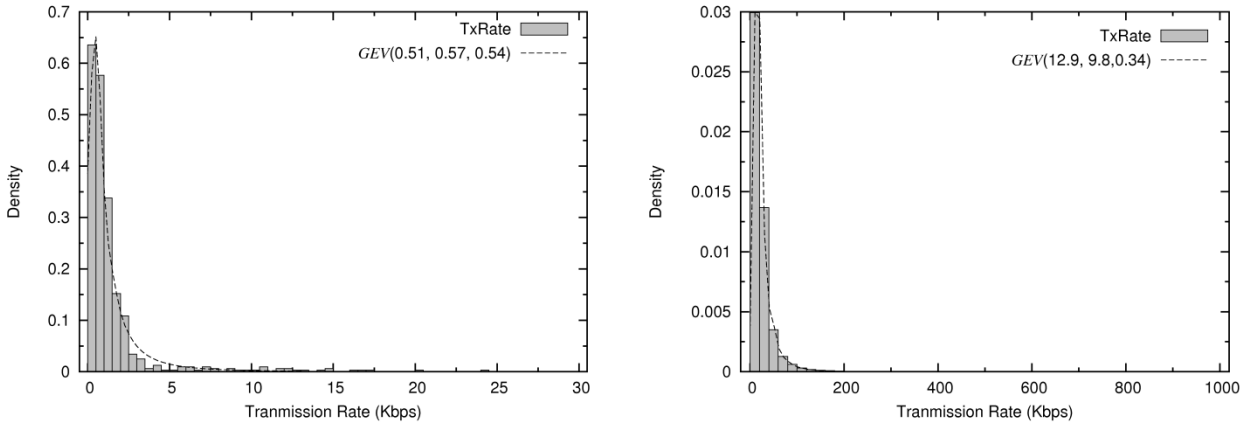
few SETI nodes with very large *NumCores*, *MemSize*, *DiskSize*, *TxRate*, and *RxRate*. However, these nodes were active only for a few weeks. These could be high-performance machines that utilize SETI as a workload to test and measure their performance. Nevertheless, these nodes provide a significant amount of computing, storage, and networking resources for a short time span.

Distribution of categorical attributes was extremely skewed and had only a few distinct attribute values. For example, *CPUArchi* of all the PlanetLab, GCO, and CSU nodes and 99% of the SETI nodes were ×86. Remaining 1% of the SETI nodes corresponded to PowerPC, SPARC, and IA-64 architectures (see Fig. 4.8). All PlanetLab and GCO nodes were Linux but nodes had several different kernel versions. 76% of the CSU nodes ran a Linux variant with different kernel versions. 82.2% of the SETI nodes had a

Table 4.3 – Distribution of attribute values.

Distribution of Attribute Values	PlanetLab*	SETI@home*
CPUSpeed (GHz)	*N*(2.63, 0.43)	*N*(2.53, 0.51)
DiskFree (GB)	No fit	*GPD*(0, 104.9, 0.51)
1MinLoad	*GPD*(0, 6.82, 0.38)	N/A
TxRate (Kbps)	*GEV*(0.51, 0.57, 0.54)	*GEV*(12.9, 9.8, 0.34)
RxRate (Kbps)	*GEV*(0.49, 0.54, 0.54)	*GEV*(111.4, 122.8, 0.84)
Integer performance (MIPS)	N/A	*Weibull*(6,498, 2.2)
Floating-point performance (MIPS)	N/A	*Weibull*(2,507, 3.4)

* *GEV*(μ , σ, k), *GPD*(μ, σ, k), *N*(μ , σ^2), *Weibull*(λ, k), N/A – Not available



(a) (b)

Figure 4.7 – Distribution of transmission rate: (a) PlanetLab; (b) SETI@home. While fitting the curve for SETI@home only the nodes with bandwidth utilization up to 1 Mbps is considered.

Windows variant (44% of them were Windows 7) and 10.6% and 7% of the nodes had variants of MacOS and Linux, respectively (see Fig. 4.9). Remaining 0.2% corresponds to BSD, SunOS, OS/2, AIX, HP-UX, and IRIX. Therefore, categorical attributes are highly skewed and their domains have only a few valid attribute values.

There is a wide variation in how frequently the attribute values change. The number of changes in dynamic attributes over a 24-hour period is observed for two weeks. Figure 4.10 shows the CDF of number of significant changes in several selected attribute values. A fixed threshold is applied to ignore minor variations. *MemFree* in PlanetLab and GCO nodes changed frequently. For example, 54% of the PlanetLab nodes changed *MemFree* by at least 100 MB in every sample taken at 5-minute intervals. 50% of the GCO nodes changed *MemFree* at least 123 times within 24 hours (out of 288 samples taken at 5-minute intervals). While the number of changes in *DiskFree* was insignificant in PlanetLab and CSU, it

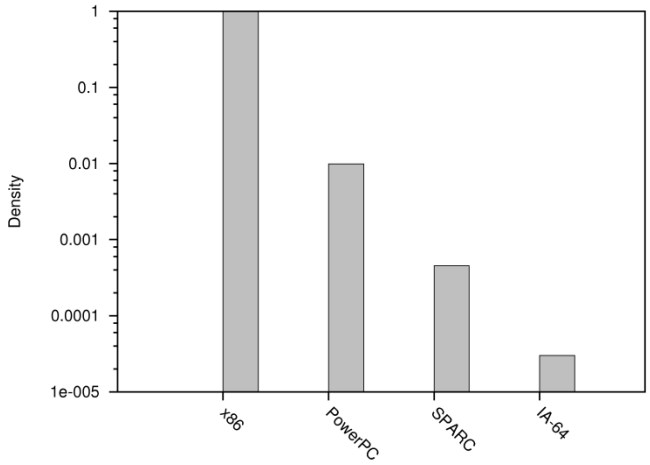


Figure 4.8 – Distribution of CPU architectures of SETI@home nodes.

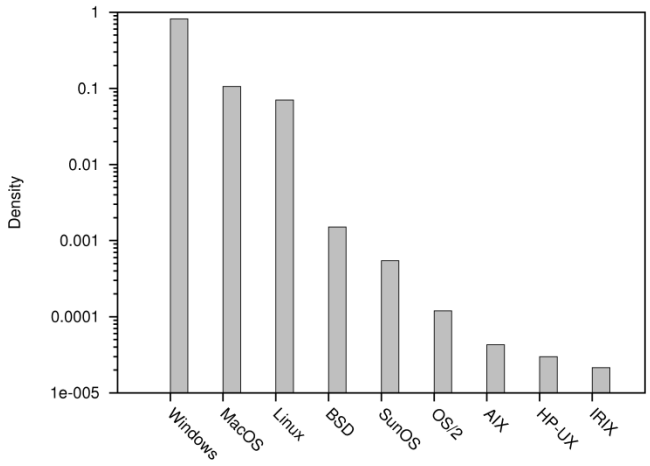


Figure 4.9 – Distribution of operating systems of SETI@home nodes.

moderately changed in GCO dataset. This may be due to the data incentive, high-energy-Physics applications that frequently run on those nodes [Ge11b]. A relatively high rate of change was observed for *1MinLoad* in PlanetLab and CSU datasets and *TxRate* and *RxRate* in GCO dataset. These indicate that the number of processes running on PlanetLab nodes shows a rapid variation with time, which could be either due to the variability in applications' resource usage or execution of small jobs. The number of changes in *1MinLoad* of CSU nodes was moderate; however, *15MinLoad* changed infrequently indicating most processes are short lived. Surprisingly, *TxRate* and *RxRate* do not change rapidly in PlanetLab however average bandwidth consumption of nodes remained relatively high. *CPUFree* moderately changed in GCO dataset, as nodes were mostly busy. The number of process running on nodes was stable with time hence

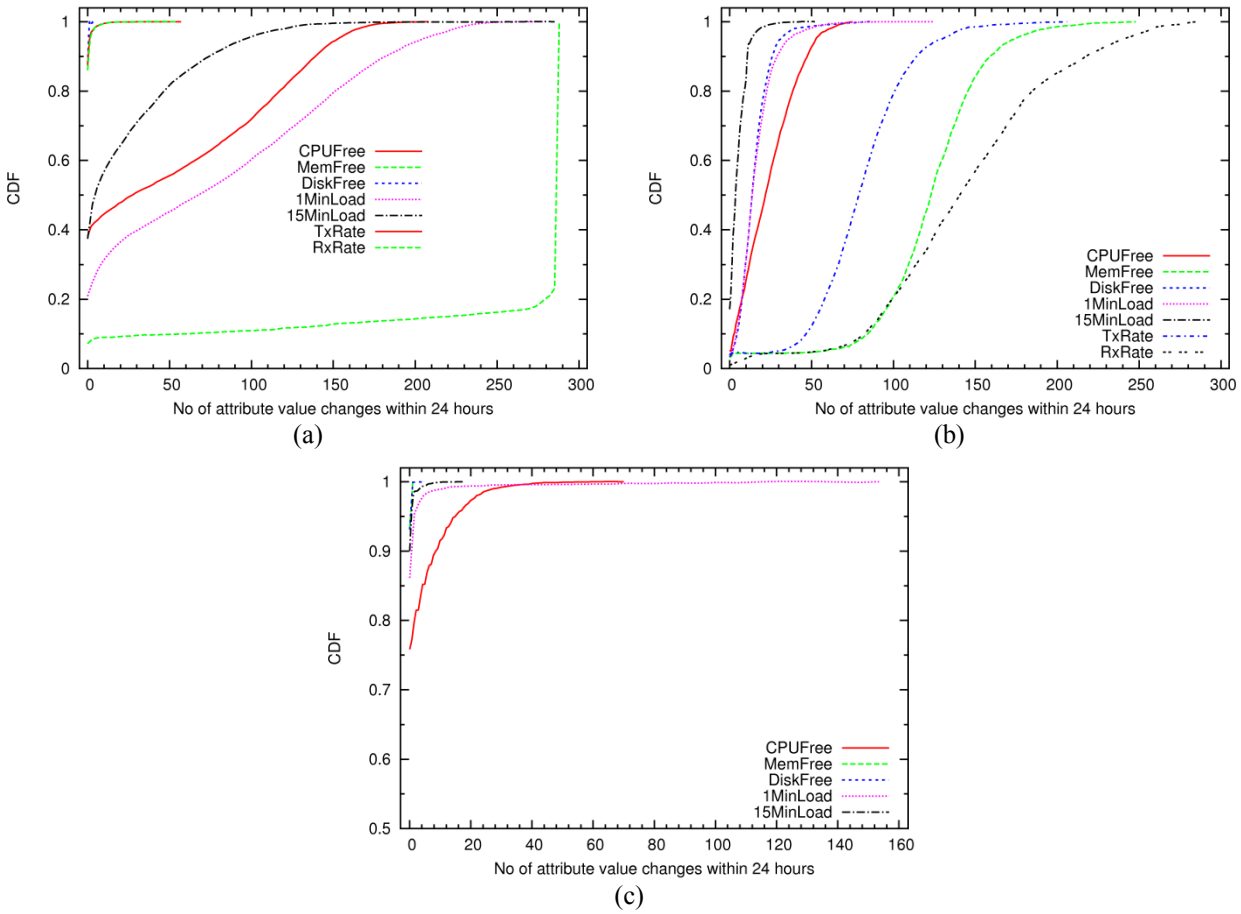


Figure 4.10 – Cumulative distribution of number of attribute value changes within 24-hours: (a) PlanetLab; (b) GCO; (c) CSU. Thresholds: *CPUFree* = ± 10%, *MemFree* = ± 100 MB, *DiskFree* = ± 5 GB, *1MinLoad* = *15MinLoad* = ± 2, *TxRate* = *RxRate* = ± 10 Kbps.

CPULoad changed infrequently. It was observed that the rate of change of some of the attributes could be approximated by several probability distributions such as GPD, GEV, Negative Binomial (NB), and T Location-Scale (TLS) and are listed in Table 4.4. In summary, rate of change λ_r^i depends on the attribute and system.

We further observed the linear and ranked correlation among attributes. Table 4.5(a) lists the linear (Pearson's) correlation among PlanetLab nodes. For the dynamic attributes, we first calculated the correlation among all nodes in a given sample (taken at a given time instance) and then calculated their average across all the samples (taken at every 5 minutes, over two weeks). Correlation values of most of the attribute pairs in different samples had a standard deviation of ≤ 0.1 confirming correlation does not

Table 4.4 – Distribution of number of significant changes in attribute values within 24-hours.

No of Changes in Attribute Values	PlanetLab*	GCO
CPUFree	No fit	*GPD*(0, 34.4, -0.45)
MemFree	No fit	*TLS*(124, 23, 2.7)
DiskFree	No fit	*TLS*(14.2, 6.86, 3.84)
1MinLoad	No fit	*GEV*(11.2, 7.99, 0.06)
15MinLoad	No fit	*GEV*(2.57, 2.93, 0.31)
TxRate	*NB*(0.08, 0.19)	*TLS*(79.7, 22.9, 4.18)
RxRate	*NB*(0.09, 0.19)	*TLS*(142.9, 51.6, 13.7)

* $GEV(\mu, \sigma, k)$, $GPD(\mu, \sigma, k)$, $NB(r, p)$, $TLS(\mu, \sigma, v)$

Table 4.5 – Correlation among attributes of PlanetLab nodes:

(a) Pearson's correlation coefficient.

Index*	1	2	3	4	5	6	7	8	9	10
2	-0.10									
3	-0.01	0.46								
4	-0.03	-0.03	-0.16							
5	-0.03	-0.02	-0.15	0.98						
6	0.02	0.43	0.30	-0.03	-0.03					
7	0.06	0.25	0.35	-0.12	-0.11	0.36				
8	-0.11	0.54	0.37	-0.09	-0.08	0.59	0.30			
9	-0.11	0.56	0.41	-0.09	-0.09	0.58	0.30	0.99		
10	0.06	-0.06	-0.12	-0.04	-0.04	0.00	0.01	0.00	-0.03	
11	0.05	-0.05	-0.11	-0.03	-0.03	0.00	0.01	0.00	-0.03	0.87

(b) Spearman's ranked correlation coefficient.

Index*	1	2	3	4	5	6	7	8	9	10
2	0.06									
3	-0.09	0.68								
4	0.08	-0.20	-0.62							
5	0.08	-0.18	-0.62	0.94						
6	-0.01	0.45	0.35	-0.34	-0.35					
7	0.20	0.61	0.51	-0.03	-0.03	0.69				
8	-0.22	0.62	0.49	-0.30	-0.29	0.49	0.63			
9	-0.21	0.66	0.56	-0.42	-0.42	0.44	0.60	0.95		
10	0.03	0.00	-0.21	0.10	0.10	0.10	0.28	0.04	-0.09	
11	0.02	0.02	-0.21	0.11	0.11	0.11	0.29	0.07	-0.07	0.95

* 1 – *CPUSpeed*, 2 – *NumCores*, 3 – *CPUFree*, 4 – *1MinLoad*, 5 – *15MinLoad*, 6 – *MemSize*, 7 – *MemFree%*, 8 – *DiskSize*, 9 – *DiskFree*, 10 – *TxRate*, 11 – *RxRate*

significantly change with time. It can be seen that (*15MinLoad*, *1MinLoad*), (*DiskFree*, *NumCores*), (*DiskFree*, *MemSize*), and (*RxRate*, TxRate) are positively correlated. *1MinLoad* and *15MinLoad* are correlated because they are the one-minute and 15-minute averages of *CPULoad*. It seems that a node with a

Table 4.6 – Correlation among attributes of GCO nodes:

(a) Pearson's correlation coefficient.

Index*	1	2	3	4	5	6	7	8	9	10
2	1.00									
3	0.26	0.26								
4	0.40	0.40	-0.68							
5	0.42	0.42	-0.67	0.96						
6	1.00	1.00	0.26	0.40	0.42					
7	0.42	0.42	0.61	-0.31	-0.30	0.42				
8	0.97	097	0.26	0.38	0.40	0.97	0.41			
9	0.96	0.96	0.29	0.34	0.35	0.95	0.48	0.99		
10	0.03	0.03	0.01	0.04	0.04	0.03	-0.01	0.03	0.03	
11	0.12	0.12	0.03	0.08	0.07	0.12	-0.06	0.11	0.10	0.25

(b) Spearman's ranked correlation coefficient.

Index*	1	2	3	4	5	6	7	8	9	10
2	1.00									
3	0.34	0.34								
4	0.21	0.21	-0.50							
5	0.22	0.22	-0.48	0.80						
6	0.88	0.88	0.30	0.19	0.19					
7	0.20	0.20	0.27	-0.21	-0.19	0.20				
8	0.94	0.94	0.32	0.19	0.20	0.83	0.19			
9	0.32	0.32	0.18	-0.10	-0.10	0.28	0.64	0.31		
10	0.05	0.05	0.26	-0.06	-0.06	0.03	-0.05	0.06	-0.10	
11	0.07	0.07	0.27	-0.06	-0.09	0.06	-0.08	0.08	-0.11	0.82

* 1 – *CPUSpeed*, 2 – *NumCores*, 3 – *CPUFree*, 4 – *1MinLoad*, 5 – *15MinLoad*, 6 – *MemSize*, 7 – *MemFree*, 8 – *DiskSize*, 9 – *DiskFree*, 10 – *TxRate*, 11 – *RxRate*

large disk space also tends to have a large number of CPU cores and memory (correlation between *NumCores* and *MemSize* is 0.43). Time series of *TxRate* and *RxRate* are highly correlated as PlanetLab nodes tend to simultaneously transmit and receive data. Spearman's ranked correlation ρ among attributes is listed in Table 4.5(b). Spearman's ρ measures how well the correlation between two attributes can be described using a monotonic function. The correlation between (*CPUFree*, *NumCores*), (*MemFree*, *NumCores*), and (*MemFree*, *MemSize*) have increased. This indicates that when a node has a large *NumCores*, *CPUFree* tends to increase (see Fig. 4.11(a)) as it is the average of all the CPUs and some of them may be idle. Figure 4.11(b) also shows that when nodes have more CPU cores their memory capacity also tends to increase. (*1MinLoad*, *CPUFree*) and (*15MinLoad*, *CPUFree*) are negatively correlated as

Table 4.7 – Correlation among attributes of SETI@home nodes:

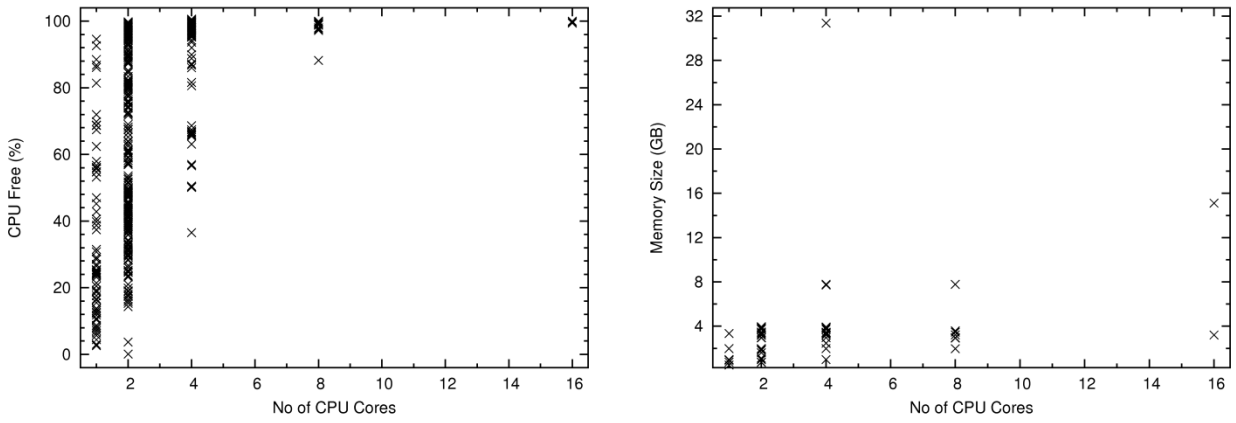
(a) Pearson's correlation coefficient.

Index*	1	2	3	4	5	6	7	8
2	0.27							
3	0.25	0.59						
4	0.28	0.31	0.26					
5	0.00	0.00	0.00	0.00				
6	0.00	0.00	0.00	0.00	0.00			
7	-0.01	0.00	0.00	0.00	0.00	0.00		
8	0.51	0.42	0.39	0.34	0.00	0.00	0.00	
9	0.60	0.34	031	0.35	-0.01	0.00	-0.01	0.79

(b) Spearman's ranked correlation coefficient.

Index*	1	2	3	4	5	6	7	8
2	0.36							
3	0.37	0.71						
4	0.26	0.44	0.51					
5	0.24	0.36	0.38	0.85				
6	0.15	0.22	0.26	0.18	0.17			
7	0.06	0.10	0.22	0.14	0.12	0.51		
8	0.51	0.62	0.68	0.42	0.36	0.22	0.16	
9	0.61	0.48	0.56	0.43	0.35	0.19	0.17	0.80

* 1 – *CPUSpeed*, 2 – *NumCores*, 3 – *MemSize*, 4 – *DiskSize*, 5 – *DiskFree*, 6 – *TxRate*, 7 – *RxRate*, 8 – *Integer performance (MIPS)*, 9 – *Floating-point performance (MIPS)*



(a) (b)

Figure 4.11 – Number of CPU cores of PlanetLab nodes vs.: (a) Free memory; (b) Memory size.

CPUFree reduces when the number of processes using the CPU increases. Increased Spearman's ρ values show that more complex correlation patterns exist among attributes that are not necessarily linear (Fig. 4.11). Linear and ranked correlations of GCO nodes are listed in Table 4.6 where many attributes were

correlated due to homogeneous nodes and heavy utilization of nodes (when a node is heavily utilized, *CPUFree* and *MemFree* reduce while *CPULoad* increases). Analysis of Spearman's ρ also indicates *TxRate* and *RxRate* are positively correlated with a coefficient of 0.82. Similar behavior is also observed for SETI (Table 4.7) and CSU (Table 4.8) nodes.

4.4.2 Query Characteristics

Table 4.9 summarizes the number of attributes (a_q), resources (m_q), and groups of resources specified in PlanetLab queries. 2% queries requested for several resources without specifying any attribute. Figure 4.12 shows the distribution of number of attributes specified in a query. 78% queries specified at most two attributes while 0.4% queries specified up to 10 attributes. Thus, queries tend to specify a lower number of attribute values (small a_q). 30% queries requested ten resources while 52% requested 50 or more resources. 19% queries requested all the resources in the system. Thus, there is a tendency to request a large number of resources (i.e., large m_q). Figure 4.13 shows the popularity of attributes specified in the queries. Dynamic attributes are the most popular. 26 attributes (out of 49) were never queried. Many que ries requested both *CPUFree* and *CPUSpeed* (or *MemFree* and *MemSize*). This may have been because the percentage values (PlanetLab represents both *CPUFree* and *MemFree* as percentages) are inadequate to represent the actual availability of resources. We observed the query popularity by clustering identical

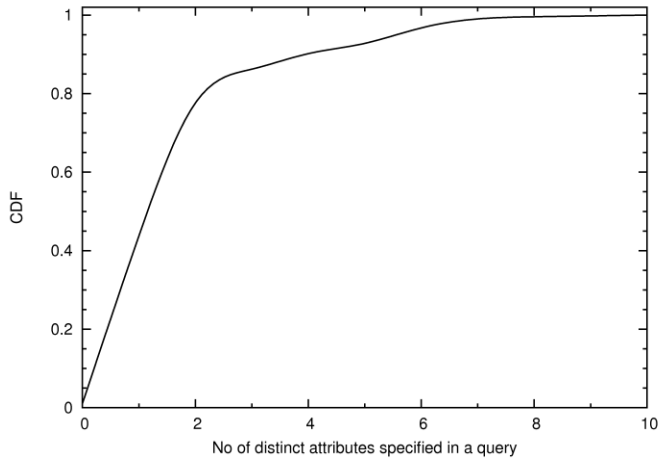


Figure 4.12 – Distribution of the number of distinct attributes specified in a query.

Table 4.8 – Correlation among attributes of CSU nodes:

(a) Pearson's correlation coefficient.

Index*	1	2	3	4	5	6	7	8
2	0.10							
3	-0.08	0.01						
4	0.07	0.05	-0.54					
5	0.07	0.06	-0.56	0.97				
6	-0.22	0.51	-0.05	0.07	0.07			
7	-0.27	0.42	-0.06	0.07	0.07	0.99		
8	0.38	0.14	0.07	0.25	0.26	-0.47	-0.60	
9	0.38	0.14	0.07	-0.11	-0.11	-0.47	-0.60	1.00

(b) Spearman's ranked correlation coefficient.

Index*	1	2	3	4	5	6	7	8
2	0.15							
3	0.00	0.09						
4	0.09	0.06	-0.37					
5	0.09	0.07	-0.39	0.73				
6	-0.34	0.45	-0.29	0.06	0.06			
7	-0.35	0.47	-0.30	0.07	0.07	0.98		
8	0.52	0.05	0.24	0.03	0.04	-0.47	-0.49	
9	0.22	0.06	0.53	-0.21	-0.21	-0.49	-0.51	0.70

* 1 – *CPUSpeed*, 2 – *NumCores*, 3 – *CPUFree*, 4 – *1MinLoad*, 5 – *15MinLoad*, 6 – *MemSize*, 7 – *MemFree*, 8 – *DiskSize*, 9 – *DiskFree*

Table 4.9 – Composition of PlanetLab queries.

	Attributes (a_q)	**Resources (m_q)**	**Groups**
Average	2.13	63.66	1.09
Minimum	0	1	1
Maximum	10	All	3
Median	2	65	1
Mode	1	All	1
Standard deviation	1.62	380	0.3

queries together. Similarly, we clustered attributes with the same range of values. 24% of the queries and 13% of the attributes specified in queries appeared only once. Figure 4.14 shows the distribution of cluster size. Number of identical queries in the largest query cluster was 31 and the largest attribute cluster had 117 queries. Though the popularity distribution is skewed, it does not satisfy a Pareto (or Zipf's-like) distribution as some of the queries are equally popular (the lines are not straight). It was further observed that the ranges of attribute values $[l_i, u_i]$ specified in queries are somewhat large. As seen in Fig. 4.15,

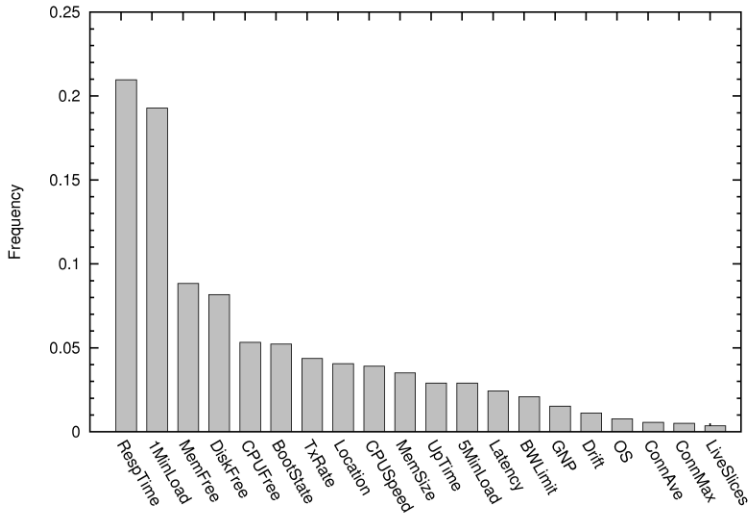


Figure 4.13 – Popularity of attributes specified in ueries. Only the first 20 is shown.

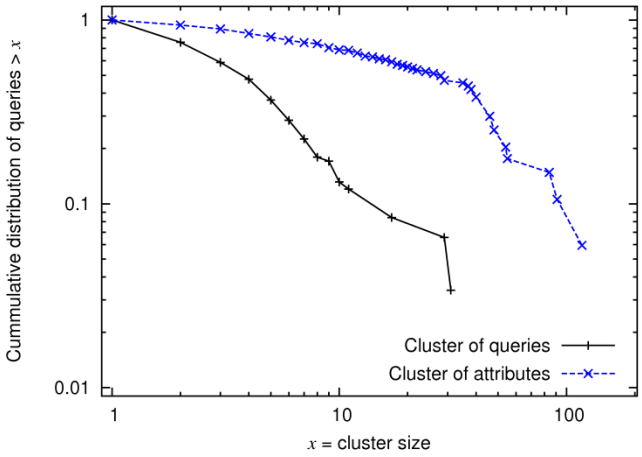


Figure 4.14 – Popularity distribution of queries and attributes specified in queries.

89% of the queries that specified *CPUFree* requested values of 40-100%. Similarly, 86% queries requested *DiskFree* values of 5-1000 GB. Therefore, most queries are less specific as they specify lower a_q and large $[l_i, u_i]$. This could be due to several reasons: (1) the nature of applications that run on PlanetLab that may not require fine-grained resource selection, (2) users' inability to identify detailed resource requirements, or (3) principle of least effort [Br05] where users are willing to specify only the simplest queries that satisfy their resource requirements.

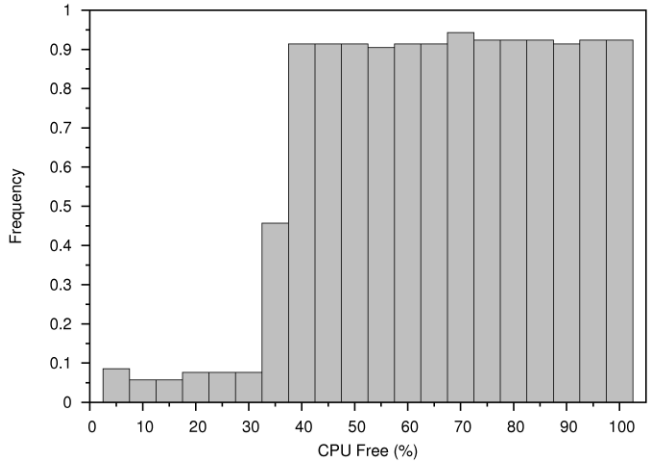


Figure 4.15 – Range free CPU values specified in queries.

4.4.3 Summary of Findings

Analysis of four datasets confirms that attribute values are correlated, their marginals satisfy different probability distributions, and in most cases are highly skewed and too complex to be represented using well-known probability distributions. Queries are skewed and less specific, as they specify small a_q, large $[l_i, u_i]$, and large m_q. These trends are likely to remain valid even in collaborative applications where users may not be informed enough to issue very-specific queries or due to the principle of least effort. Iosup and Epema [Io10] also observed that while most of the jobs in grids use a single CPU, few jobs tend to use a large number of CPUs (128 CPUs or more) hence m_q tends to be large even in grids. These findings invalidate the commonly used assumptions such as i.i.d. attributes, uniform/Zipf's distribution of all the attributes [Bh04, Ca04, Sh07], and queries specifying a large number of attributes and small ranges of attribute values [Ca04, Sh07]. Correlations between attribute pairs indicate that resources are not uniformly distributed throughout the multi-attribute space (domain of all $\mathbf{D_i}$s). Domain $\mathbf{D_i}$ of categorical attributes such as *CPUArchi* and *OS* are small as they have only a few distinct attribute values. Various hardware constraints favor having *NumCores* and *MemSize*s that are powers of two. Therefore, these attributes occupy only a small fraction of their domain $\mathbf{D_i}$. Moreover, large numbers of identical resources in PlanetLab, Grid, Cloud, and enterprise computing environments appear as sets of clusters within the multi-attribute space. Therefore, non-uniform and clustered distribution of attribute values force nodes to

index and/or answer a disproportionate number of resources and queries leading to load balancing issues. For example, DHTs that assume uniform distribution of resources [Lu04] will fail to provide static load balancing in these systems.

Both a_r and A are large, as the resources are described using tens of attributes (see Table 4.2). Dynamic attributes change at different rates and some of them change frequently (high λ_r^i). A user can decide when to run a BOINC client. Hence, SETI nodes are active only 81% of the time on average [An06]. Even though a node may be active, it does not execute jobs unless idle/residual computing resources are available. Thus, average job execution time was only 84% of the active time (overall availability of 68%). Nodes were reachable through the network 83% of the active time [An06]. Another study [He09] observed that 50% of the nodes have effective job execution time of less than 40% and only 5% of the nodes are available for job execution over 80% of the time. Alternatively, most nodes in PlanetLab, GCO, and CSU datasets were avaialable throught the two-week period. Thus, availability of nodes in volunteer-computing environments is significantly lower compared to more stable environments like PlanetLab, grids, and clouds. Consequently, λ_r^i increases further due to frequent arrival and departure of nodes. It is also observed that grids tend to experience sudden arrivals of a set of jobs (a.k.a. bag of tasks) [Io10]. Such arrivals suddenly change the dynamic attributes of many resources (e.g., Fig. 4.5(b)). Higher λ_r^i values and sudden changes in system resources significantly increase the advertising cost of RD solutions hence should not be ignored.

4.5 Design Choices in P2P-Based Resource Discovery

To ease the RD, resources are typically advertised and indexed at specific or random locations within the RD system. Resources are then located using various querying mechanisms. Resource advertising and querying options of centralized, unstructured P2P, and structured P2P solutions are analyzed using the characteristics learned in the previous section. Equation (4.6) is also extended to reflect the specific behavior of each solution.

4.5.1 Centralized Designs

Resources can be advertised and indexed at a well-known central location. Centralized indexes are utilized in PlanetLab, GENI, and grid and cloud computing. Latency and cost of resource advertising and querying are minimum as nodes can directly communicate with the central node. Hence, the cost per message is $O(1)$. Resolving queries with large $[l_i, u_i]$ and m_q do not introduce additional routing overhead. Thus, the RD cost depends on the initial advertisement cost, recurrent cost of advertising static and dynamic attributes, and query cost. Then (4.6) can be modified as:

$$C_{Total} = \sum_{r \in \mathbf{R}} \left(1 + \sum_{j \in \mathbf{a_r}} \lambda_r^j t \right) + Q(t) \tag{4.7}$$

$c_r^i = 1$, as resources are advertised only to the central location. If the query arrival rate is constant, total cost of RD is bounded by $O(RA\,\lambda_r^i)$. In the worst case, all the resources may have all the attributes. Therefore, cost of indexing resources at the central node is $O(RA)$. This approach is not scalable (in terms of both the number of messages and index size) as A, R, and λ_r^i are typically large in production systems. It also leads to a single point of failure.

Hierarchical indexes are proposed to overcome these limitations where separate indexing nodes are assigned to different geographic regions, sites, or organizations [El09]. Then a higher-level node(s) is assigned to keep track of aggregated resources from these nodes. This approach could lead to conflicts while querying and binding resources. Partial failures in sub-regions of the system are also problematic. Moreover, aggregation along the hierarchy reduces the resolution at which resources are advertised. Such hierarchies are not desirable while indexing highly dynamic attributes such as *MemFree* and bandwidth.

4.5.2 Unstructured P2P-Based Designs

Unstructured P2P systems [Lu04] are attractive as they distribute the resource information across many nodes in the system while providing resilience and load balancing. They are utilized in file sharing, mobile social networks, and ad-hoc networks. These systems use either flooding or random walks [Ta08]

to disseminate information about resources and/or resolve queries. Though resources are guaranteed to be found with flooding, it is extremely costly. Random-walk-based solutions use agents for advertising and/or querying. Agent lifetime is controlled using a Time To Live (*TTL*) value that tries to balance the cost of advertising/querying and query success rate. As queries are forwarded to individual nodes, their availability is known. Hence, no additional overhead is introduced. Assuming a new agent is generated for each change in attribute value or query, RD cost is:

$$C_{TOTAL} = \sum_{r \in \mathbf{R}} \left(\sum_{i \in \mathbf{a_r}} TTL + \sum_{j \in \mathbf{a_r}} \lambda_r^j t TTL \right) + \sum_{q \in \mathbf{Q(t)}} h_{Query}^q \tag{4.8}$$

h_{Query}^q is typically $O(TTL)$. Therefore, the total cost is bounded by $O(RA\, \lambda_r^i\, TTL)$. However, random walks are not guaranteed to find resources, and it will become even harder if one reduces *TTL* with the intention of reducing the cost of RD. It has been shown that the hitting time (i.e., expected number of hops to reach any node staring at any node) of a random walk on an arbitrary finite graph is $O(N^3)$ [Ik09]. Hence, *TTL* has to scale with N^3 to increase the success rate of queries. Therefore, unstructured P2P systems are suitable only for moderate scale, best effort, and highly dynamic environments that can tolerate large delays. Resource indexing cost per node is $O(RA)$ as all the nodes can eventually get to know about all the resources in the system. Though advertising resources to other nodes speeds up the query resolution, state of the selected resources may be stale.

More-scalable version of this approach uses a two-layer overlay where resource rich nodes, namely *superpeers*, form a separate overlay while acting as proxies for rest of the nodes. Similar to hierarchical solutions, superpeers manage multiple resources. Resources can communicate with their superpeers using one overlap hop. When local resources are insufficient to resolve a query, superpeers may query other superpeers using flooding or random walks. Therefore, $h_{Query}^q = O(N_{Superpeer})$ or $h_{Query}^q = O(TTL)$, where $N_{Superpeer}$ is the number of superpeers in the system. Cost of RD is:

$$C_{TOTAL} = \sum_{r \in \mathbf{R}} \left(1 + \sum_{i \in \mathbf{a_r}} \lambda_r^i t \right) + \sum_{q \in \mathbf{Q(t)}} h_{Query}^q \tag{4.9}$$

As $N_{Superpeer} \ll N$, *TTL* can be set to a relatively lower value than in unstructured P2P systems. However, superpeers are also not guaranteed to find resources with random walks. Indexing load on a superpeer is proportional to A and the number of resources assigned/connected to it. If resources are uniformly assigned to superpeers, index size is $O(RA/N_{Superpeer})$.

4.5.3 Structured P2P-Based Designs

Structured P2P systems are appropriate for large-scale implementations due to high scalability and some guarantees on performance [Lu04]. These systems use a DHT to index resources. Each DHT node or a resource has a unique identifier called a *key*. Each resource's contact information is indexed (i.e., stored) at a node having a close by *key* in the key space. Resources are advertised and queried using messages that are forwarded to appropriate nodes using a deterministic overlay network. Chord, Kademlia, CAN, and Pastry (see Section 2.1) are some of the well-known solutions that are used to build such an overlay. These solutions typically keep *pointers* to nodes that are spaced at exponentially increasing gaps in the key space enabling messages to be routed with a bounded path length of $O(\log N)$. DHTs are designed to index resources that are characterized by a single attribute. As it is, they are not efficient for simultaneous selection of multiple and multi-attribute resources. A representative subset of solutions that extend DHTs to support multi-attribute resources and queries are discussed next.

Multiple Address Spaces

One of the simplest solutions is to maintain a separate DHT for each attribute in A [Bh04]. Figure 2.10 illustrates a design based on three DHTs with a circular address space, also referred to as a *ring*. Each resource r advertises either each attribute value to the corresponding ring or all the attribute values to all the rings responsible for $\mathbf{a_r}$. In the former case, a multi-attribute query q is first split into a set of *sub-queries* where each sub-query searches for one of the attributes in $\mathbf{a_q}$. Sub-queries are then simultaneously forwarded to appropriate rings. Query results have to be then combined at the application using a join operation like in databases. A range query is resolved by forwarding each sub-query to a series of nodes responsible for indexing attribute values in the range $[l_i, u_i]$ (see Fig. 4.1). Thus, RD cost is:

$$C_{TOTAL} = \sum_{r \in \mathbf{R}} \left(\sum_{i \in \mathbf{a_r}} h^i_{Advertise} + \sum_{j \in \mathbf{a_r}} h^j_{Advertise} \lambda^j_r t \right) + \sum_{q \in \mathbf{Q(t)}} \left\{ \sum_{i \in \mathbf{a_q}} \left(h^i_{Query} + \left\lceil \frac{(u_i - l_i)}{D_i} N \right\rceil - 1 \right) \right\} \quad (4.10)$$

Each advertising message can be sent in $O(\log N)$ hops thus advertising cost is $O(RA\,\lambda^i_r \log N)$. Worst-case query cost is $O(N)$ as $h^i_{Query} = O(\log N)$. Even the average cost is large, as queries tend to specify large $[l_i, u_i]$. Indexing cost of all the rings is $O(RA)$ or $O(R)$ per ring.

In the latter case, q can be resolved using one of the rings, as each ring is aware of all the attribute values of a resource. This also enables queries to be terminated as soon as the desired number of resources is found. Query resolution cost can be further reduced by forwarding q to the ring corresponding to the *most selective attribute* (i.e., attribute with the lowest $(u_i - l_i)/D_i$). For example, as seen in Fig. 2.10(a), the query travels 3-hops in *CPUSpeed* and *bandwidth* rings while it travels only 2-hops in the *MemSize* ring. Thus, q is issued only to the *MemSize* ring. This approach is called *Single-Attribute Dominated Querying* (SADQ) and is used in [Al08, Bh04, Ca04]. Total RD cost is:

$$C_{TOTAL} = \sum_{r \in \mathbf{R}} \left(\sum_{i \in \mathbf{a_r}} h^i_{Advertise} + \sum_{j \in \mathbf{a_r}} a_r h^j_{Advertise} \lambda^j_r t \right) + \sum_{q \in \mathbf{Q(t)}} \left(h^k_{Query} + \left\lceil \frac{(u_k - l_k)}{D_k} N \right\rceil - 1 \right) \quad (4.11)$$

where $k \in \mathbf{a_q}$ is the most selective attribute. In the worst case, each change in attribute values has to be advertised to all the rings hence advertising cost is $O(RA^2\,\lambda^i_r \log N)$. Worst-case query cost is still $O(N)$. Real-world queries specify only a few attributes hence not many options are available while choosing the more selective attribute. Therefore, even the average query cost is relatively high. Indexing cost of all the rings is $O(RA^2)$ or $O(RA)$ per ring. This approach has a lower query cost than using multiple sub-queries. However, it has a higher advertising cost due to multiple copies where $c^i_r = a_r$ (recall a_r tends to be large). Therefore, it is suitable only if the query rate is higher than λ^i_r. New rings can be added to support additional attributes. However, excessive routing entries associated with multiple rings (typically $O(A \log N)$ make it less scalable. Some rings experience higher load due to skewed distribution of resources and queries. This is particularly a problem in SADQ where highly selective and skewed attributes such as *CPUArchi*, *NumCores*, and *OS* may get a large number of queries overloading the nodes that index them.

Moreover, domain $\mathbf{D_i}$ of these attributes is small, and hence may not be suitable to be placed on a separate ring/overlay.

Partitioned Address Space

Figure 2.10(b) shows a ring that is partitioned into several segments where each segment corresponds to a separate range of attribute values. Prefix bits of the overlay *key* are used to represent the attribute type and suffix bits represent the attribute value. A Locality Preserving Hash (LPH) [Al08, Ca04, Sh07] function is used to assign the suffix bits, which preserves the locality of attribute values. Advertising and querying schemes are similar to the case of multiple address spaces. However, each partition has approximately N/A nodes therefore worst-case query cost is $O(N/A)$. Moreover, these solutions maintain a much lower number of routing entries, typically $O(\log N)$. Some partitions experience higher load due to skewed distribution of resources and queries while most of the remaining partitions are rarely utilized.

Overlapped Address Space

Another alternative is to map all the attribute values to the same ring (see Fig. 2.10(c)) using a separate LPH function for each attribute type [Ca04]. Cost of RD and indexing is same as multiple address spaces because the same advertising and querying mechanisms are applicable here. Some nodes experience higher load due to skewness in resources and queries. However, routing state is $O(\log N)$ as only a single ring is used.

d-torus

Figure 2.12(b) illustrates an alternative design where each dimension of a d-torus represents an attribute (i.e., $d = A$). Resources are mapped to the torus according to their attribute values. A multi-attribute range query q encloses a hyper-box on the torus. This approach is used in [Co09b] where dimensions of the torus reflect only the static attributes (i.e., $d = A^s$). For routing, d-torus is logically partitioned into an increasing set of *levels* and each level is further partitioned along each dimension d forming a set of *cells*. Each node keeps a set of pointers to all the other nodes in the same cell and to a node in each partition at level l and dimension d. In contrast to prior architectures, resources are not explicitly advertised. Instead, a gossip scheme is used to identify a random node in each (l , d) pair. These pointers are used to

route q to the desired query region on the torus using depth-first search (see Fig. 2.12(b)). Then neighboring nodes are used to traverse from one node to another during which dynamic attributes in q may be evaluated. RD cost is:

$$C_{TOTAL} = \sum_{q \in \mathbf{Q(t)}} h^{q}_{Query} + m_q - 1 \tag{4.12}$$

It can be shown that h^{q}_{Query} is $O(\ln 2^d / \ln d)$. After reaching the first node that satisfies q, remaining $m_q - 1$ resources need to be found. Hence, worst-case query cost is $O\left(\ln 2^d / \ln d + m_q + 1\right)$, ignoring the overlay maintenance cost. As the propagation of q is based on depth-first search and query visit individual nodes (one at a time), query resolution latency is much higher. This routing scheme is inefficient when queries are less specific, which significantly increases the volume of the hyper-box to query. Moreover, it cannot route queries with only the dynamic attributes, which accounts for a large fraction of queries in real-world systems (see Fig. 4.13). Summary of all the solutions are listed in Table 4.10.

4.6 Simulation Setup

Next, we quantitatively evaluate the fundamental design choices for P2P-based RD. We simulated seven representative architectures for RD (listed in Table 4.10) against the same set of resources and queries derived from PlanetLab. Use of realistic data preserves the complex distribution of attributes, dynamic and correlated changes in attribute values, and users' interest in resources. To simplify the performance analysis and eliminate any bias due to node failure, we replayed a trace with only the PlanetLab nodes that were continuously available for three days starting 2010/11/08. There were 527 such nodes. As our query dataset was small, a large number of synthetic queries were generated using the empirical distributions of a_q, popularity of attributes, $[l_i, u_i]$, and m_q. To capture the correlation among attributes and attribute value ranges, conditional probabilities of attribute occurrences are also taken into account. The query generation solution presented in Chapter 5 was not used, as it was not designed by the time of these simulations. Queries were issued only after the network was stabilized. Performance of ring-based structured P2P solutions were also evaluated under different numbers of nodes ranging from 250 to 1,000.

Table 4.10 – Summary of resource discovery architectures.

Architecture	Routing Mechanism	Advertising		Querying		Indexing	
		Mechanism	Cost	Mechanism	Cost	Mechanism	Cost
1. *Centralized*	Direct	To central node	$O(1)$	Query central node	$O(1)$	At central node	$O(RA)$
2. *Unstructured* – Random overlay [Ta08]	Random walk	Optional	$O(TTL)$	Query random nodes	$O(TTL)$	Index locally	$O(A)$ or $O(RA)$ if advertised
3. *Superpeer* – Random overlay among superpeers	Random walk among superpeers	To superpeer	$O(1)$	Query superpeers	$O(TTL)$	At superpeer	$O(RA/N_{Superpeer})$
4. *Multi-ring* [Bh04] – Separate ring-like overlay for each attribute type	Chord	To relevant ring(s) based on attributes	O(log N)	1. Multiple sub-queries 2. Single attribute dominated query	$O(N)$	1. At relevant rings 2. At all rings	1. $O(RA)$ 2. $O(RA^2)$
5. *Partitioned-ring* [Al08, Sh07] – Each attribute type is assigned a different segment of overlay ring	Chord or Cycloid	To relevant partition(s) based on attributes	$O(\log N)$	1. Multiple sub-queries 2. Single attribute dominated query	$O(N/A)$	1. At relevant partitions 2. At all partitions	1. $O(RA)$ 2. $O(RA^2)$
6. *Overlapped-ring* [Ca04] – All attribute types are mapped to same ring-like overlay	Chord	To relevant nodes in the ring	$O(\log N)$	Single attribute dominated query	$O(N)$	At relevant nodes	$O(RA^2)$
7. *d-Torus* [Co09b] – d-torus partitioned into a set of cells	Depth-first search	Not required	–	Visit cells that overlap with query hyper-box	$O\left(\frac{\ln 2^d}{\ln d}\right)$	Index locally	O(A)

Large number of nodes beyond 527 was generated using our correlation persevering, multi-attribute resource generation tool presented in Chapter 5.

Both the unstructured and superpeer-based networks were generated using the B-A scale-free network generator [Ge07] with a minimum node degree of two. Advertising dynamic attributes is not that useful in unstructured P2P and superpeer (from one superpeer to another) architectures, as state of resources may be different by the time they are queried. Therefore, only the query agents are used in these two solutions. The number of superpeers is set to 20. Maximum number of hops to forward a random walk (i.e., *TTL*) is set to 100 and 10 hops in unstructured and superpeer architectures, respectively. Those *TTL* values were sufficient to achieve ~70% query hit rate with different number of attributes. Multi-ring,

partitioned-ring, and overlapped-ring architectures are based on the Chord overlay [St03]. According to [Co09b], number of cell levels of the *d*-Torus is set to three. Minimum update interval for resource attributes is 5 minutes, which is the sampling interval of PlanetLab nodes. A fixed threshold is applied to ignore minor variations. Each node issued queries based on a Poisson distribution with a mean inter-arrival time of 2.5 minutes (i.e., two queries per sampling interval per node). Results are based on eight samples with different random seeds. Additional details on simulators are given in Appendix II.1.

4.7 Performance Analysis

Figure 4.16 shows the total cost of advertising and querying resources using only the 12 static attributes of PlanetLab nodes. This enables us to validate the performance of different architectures against their prior performance studies (when such a study exists), which did not consider the cost of advertising dynamic attributes. As expected, centralized architecture has the lowest overall cost while unstructured P2P architecture has the highest cost. Superpeer-based architecture has the second lowest cost. Less specific queries (large $[l_i, u_i]$ and small a_q) make it easier to find required resources by visiting a few superpeers that index multiple resources. For example, though random walks are not granted to find all the resources, superpeer architecture was able to resolve 96% of the queries. Therefore, most of the random walks were terminated within a few hops while reducing the overall cost. However, the unstructured P2P solution was able to resolve only 73% of the queries as they visit one node at a time. Partitioned-ring-based architecture has the third lowest cost as query cost is $O(N/A)$. Cost of multi-ring and overlapped-ring-based architectures are higher compared to the partitioned-ring, as their query cost is $O(N)$. *d*-torus has the second largest overall cost due to less-specific queries that dramatically increase the volume of the query hyper-box. It is not considered for rest of the discussion, as it cannot route queries with only the dynamic attributes, which are the most popular types of attributes defined in real-world multi-attribute queries.

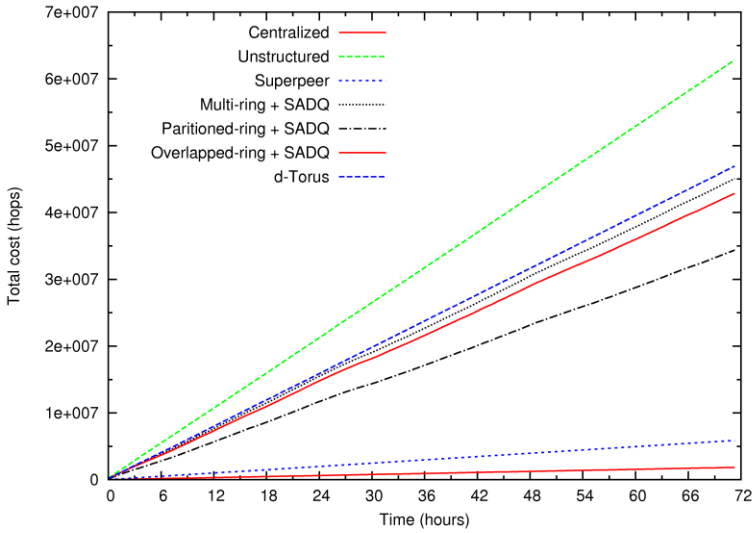


Figure 4.16 – Total cost of advertising and querying static attributes. $N = 527$.

We analyze ring-based architectures in detail, as they are considered applicable in large-scale applications due to scalability and some guarantees on performance. Figure 5.17(a) shows the per-node advertising cost of ring-based architectures while varying the number of attributes. Advertising cost increases as the dynamic attributes are introduced (first 12 attributes are static). Resources need to be re-advertised whenever their attribute values change significantly. It is typically assumed that DHT entries will expire after a predefined timeout. However, our analysis shows that it is nontrivial to determine an appropriate timeout given the diversity of attributes and their rate of change λ_r^i. Therefore, the old attribute values need to be explicitly removed from the DHT to maintain a consistent resource index. We considered the cost of removing those old indexes as part of the advertising cost. Both advertise and remove messages can be delivered within $O(\log N)$ as they are sent to specific nodes. SADQ requires all the attributes of a resource to be advertised to each ring/partition corresponding to each attribute $a \in \mathbf{a_r}$. Therefore, resources need to be re-advertised to all the rings/partitions even when a single attribute is changed. Figure 5.17(a) confirms this behavior where advertising cost of architectures that utilize SADQ is significantly higher and increases linearly with the number of attributes. We introduced dynamic attributes according to their popularity where attribute 13 is the most popular, 14 is the second most popular, and so on. Attributes 13-15 that correspond to response time of a node (*RespTime*), *1MinLoad*, and *MemFree* are

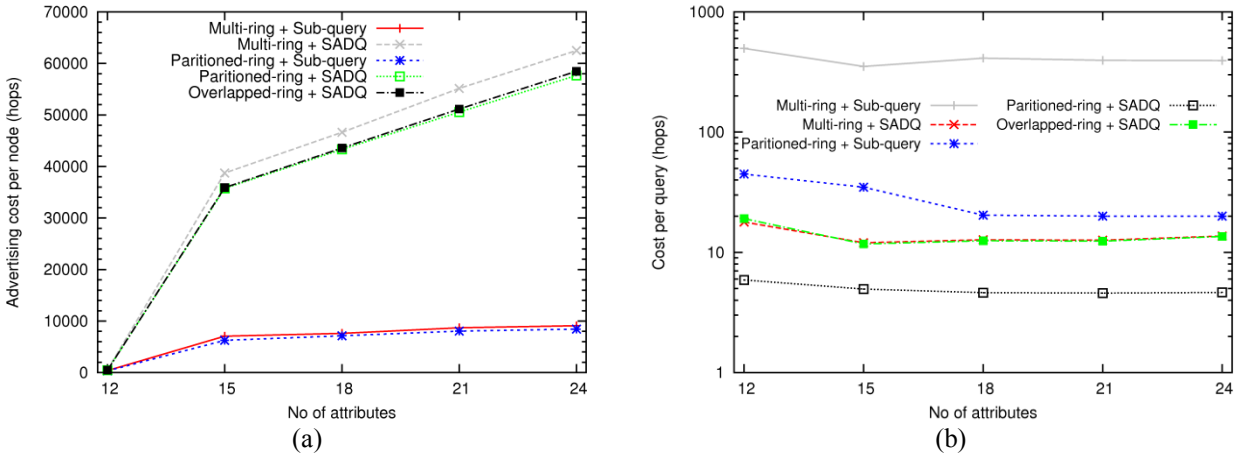


Figure 4.17 – Cost of ring-based architectures: (a) Advertising cost; (b) Query cost. $N = 527$.

updated frequently. This is the reason for the significant increase in advertising cost between attributes 12 and 15. The rate of update λ_r^i of rest of the dynamic attributes was relatively lower.

Figure 4.17(b) shows the cost per query. Architectures based on SADQ have much lower query cost, as they use only the most selective attribute and queries are terminated as soon as m_q resources are found. However, overall cost (advertising and querying) of SADQ-based architectures will be acceptable only if queries are more frequent than advertisements. Alternatively, sub-queries need to search in multiple rings/partitions and have to search the entire range of attribute values specified in each sub-query hence have a higher cost. Queries that specified attributes 13-14 (*RespTime* and *1MinLoad*) tend to be more specific (i.e., small $(u_i - l_i)/D_i$); therefore, can be resolved by forwarding to a lesser number of nodes. Furthermore, these were the two most popular attributes hence appeared in many queries consequently reducing the overall query cost. Therefore, query cost drops when the number of attributes is 15. Query distribution and range of attribute values get balanced as rest of the dynamic attributes are introduced. Consequently, query cost tends to stabilize. Though new attributes were introduced, PlanetLab queries specified only 1-10 attributes and one or two attributes were specified 78% of the time. This explains why the query cost seems to be independent of the number of attributes in the system.

The best design choice from each of the architectures is compared in Fig. 4.18. Unstructured-P2P-based architecture, which has the highest cost per query, is not shown to simplify the graph. Increase in advertising cost of centralized and superpeer architectures due to dynamic attributes is insignificant, as the cost is $O(1)$. Moreover, their query cost is independent of a_q, $[l_i, u_i]$, and m_q. Therefore, they have the lowest cost per query. Cost of ring-based architectures tends to increase linearly. In Table 4.11, we compare the three best ring-based designs with varying number of nodes N. Both the average cost per query

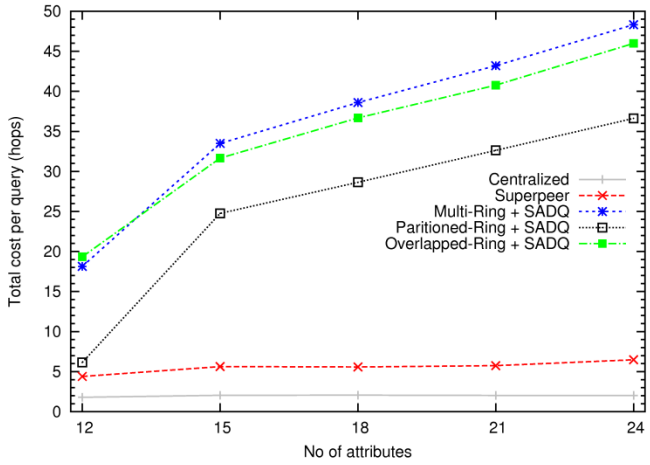


Figure 4.18 – Total cost (advertising and query) per query vs. number of attributes. $N = 527$.

Table 4.11 – Query cost of ring-based designs under varying number of nodes ($A = 24$).

N	Multi-Ring + SADQ			Partitioned-Ring + SADQ			Overlapped-Ring + SADQ		
	Min	Ave	Max	Min	Ave	Max	Min	Ave	Max
250	0	9.2	239.1	0	3.7	19.4	0	9.1	238.4
527	0	13.7	509.0	0	4.6	27.6	0	13.5	506.0
750	0	16.2	719.1	0	4.9	36.6	0	16.5	719.9
1000	0	19.8	975.5	0	5.3	45.3	0	20.4	963.8

Table 4.12 – Query cost, query load, and index size ($N = 527$, $A = 24$).

Architecture	Total Cost per Query		Query Load				Index Size	
			Min		Max		Min	Max
	SWORD	Uniform	SWORD	Uniform	SWORD	Uniform		
Centralized	2.03	2.03	950,000	950,000	950,000	950,000	527	527
Unstructured	69.5	94.8	4,859	1,272	268,497	37,824	1	1
Superpeer	6.5	9.5	81,021	22,390	289,626	87,209	17	36
Multi-ring + SADQ	48.3	69.0	0	0	178,492	22,943	0	527
Multi-ring + Sub-queries	398.8	120.8	0	0	624,837	57,518	0	230
Partitioned-ring + SADQ	36.6	37.0	0	0	185,972	15,840	0	527
Partitioned-ring + Sub-queries	40.7	16.4	0	0	432,859	46,946	0	527
Overlapped-ring + SADQ	46.0	67.2	0	0	391,738	57,524	0	527

and maximum query cost tend to increase linearly with N confirming the analysis in Section 4.5.3. Therefore, query cost is bounded by $O(N)$. Cost of partitioned-ring-based architecture is lower as it is bounded by $O(N/A)$.

Table 4.12 presents the query cost as well as per node query load and index size. For comparison, according to [Ca04], we also generated range queries based on the uniform distribution of attributes and attribute values where each attribute in a query specified 10% of the possible range of attribute values (i.e., $(u_i - l_i) = 0.1\ D_i$). Resource attributes were not changed. Performance of RD architectures is different when queries are formulated by selecting attributes and ranges of attribute values uniformly at random (irrespective of the actual resources). Not all random queries match resources in the system. Therefore, both the unstructured and superpeer architectures have to keep forwarding the queries until the *TTL* expires. Consequently, the overall cost of RD increases. Furthermore, uniform queries cannot significantly benefit from SADQ, as the minimum range of any attribute is always 10% of the domain. Whereas in real-queries (collected from SWORD), at least few attributes tend to be very specific (e.g., *RespTime*, *1MinLoad*, and *NumCores*). Therefore, real-world queries can be resolved more efficiently using SADQ. Alternatively, given that the range of most of the attribute values in real-world queries tend to be large, aggregated query cost significantly increases when multiple sub-queries are used.

Centralized solution had to index all the resources and answer all the queries issued within the system (950,000 queries were issued during the simulation). Unstructured and superpeer architectures have a smaller index size as either resources index themselves or superpeers index only a subset of the resources in the system. Furthermore, their query load is relatively balanced due to the use of random walks. Uniform queries are issued to different overlay rings/partitions with the same probability hence distribute the query load among many nodes. Alternatively, due to the skewed distribution of attributes and attribute values in PlanetLab queries, a few rings/partitions and subset of the nodes within those rings/partitions are used to answer most of the queries. This is the reason that maximum number of SWORD queries answered by a node is 3.3 to 11.7 times higher than when answering uniform queries. The query load on a node that supports SADQ is relatively low, as the queries are resolved using the most

selective attribute. Moreover, queries also terminate as soon as the desired number of resources is found. Hence, each node has to handle a relatively smaller number of queries. Figures 4.16-4.18 confirm that partitioned-ring with SADQ outperform all other design choices. Figure 4.19(a) illustrates the query load distribution of architectures supporting SADQ. It can be seen that load distribution is skewed and few nodes had to answer majority of the queries. It is particularly worse in the partitioned-ring where 74% of the nodes did not answer any query and one of the nodes answered ~20% of all the queries. Such an im-balanced load distribution is not acceptable when the query rate is higher.

Because of the correlation and skewed distribution, resources are not uniformly spread across the attribute space. Therefore, resources are indexed in only a small subset of the nodes in the ring while a large fraction of nodes does not index any resources. Multiple indexing used with SADQ and overlapped-ring also force nodes to index many resources corresponding to different attributes. Moreover, some of the attributes have only a few valid attribute values and they are highly skewed, e.g., *CPUArchi* and *NumCores*. Such attribute values are indexed in a few nodes and some of those nodes have to index a large fractions of resources with the same attribute value. Index size distribution of solutions supporting SADQ is shown in Fig. 4.19(b). In the partitioned-ring architecture, nodes that were mapped to less popular or unused partitions were never utilized. Therefore, it suffers from significant load balancing issues, though it has the lowest RD cost among structured P2P architectures. In conclusion, load balancing is a critical issue in all the designs, as popularity of resources/queries is skewed and queries are less specific.

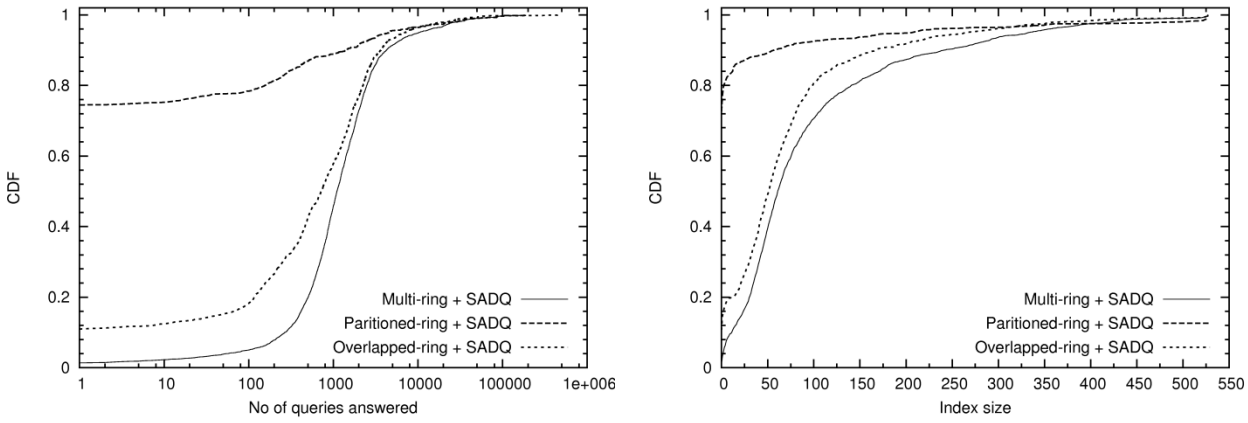


Figure 4.19 – Distribution of load: (a) Query load; (b) Index size. $N = 527$.

4.8 Discussion

None of the design current choices for RD simultaneously provide efficiency, scalability, and load balancing under real workloads. Centralized architecture has the lowest cost per advertisement and query. A single message can be used to either advertise or query for a resource(s) irrespective of a_r, a_q, [l_i, u_i], and m_q. Lower advertising costs enable dynamic attributes to be advertised whenever their attribute values change while increasing the accuracy of indexed resources. As the central node is aware of all the resources in the system, it is also suitable for complex tasks such as matching multiple resources (for inter-resource bandwidth and latency), establishing a binding between a resource and an application that is interested in using it, and enforcing various incentives, trust, and security policies. However, a single node may not be able to handle all these messages. It also leads to a single point of failure and privacy issues, as the central node can monitor usage patterns of resources. Nevertheless, centralized solutions are becoming more feasible, affordable, and reliable due to the recent advancements in distributed datacenter technologies. Therefore, when applicable/feasible, centralized solution is still a desirable option.

Superpeer architecture is relatively efficient in resolving real-world queries that are less specific. Moreover, both the query load and index size are well balanced and independent of a_r, a_q, and [l_i, u_i]. Resources can afford to advertise dynamic attributes whenever they change as the superpeer can be reached within one overlap hop. However, it does not provide a best-fit type solution, where select resources fit the minimum requirements of a query. Hence, applications that actually require a large number of high-capacity resources may not be able to find them, as they have been already allocated to applications with much lower resource demand. Trying to enforce best-fit type matching will increase the query cost. Therefore, superpeer architecture is more suitable for dynamic, best-effort environments such as mobile social networks and ad-hoc networks. Due to high query cost and relatively low query hit rate, unstructured P2P architecture is not suitable for most applications. However, its high resilience to random node failures makes it suitable for highly dynamic environments such as mobile social networks. Advertising agents are of little use, as some dynamic attributes change very frequently making the indexed resources

stale. While it has been proposed to use aggressive timeouts to invalidate resources after a while, given the diversity in attributes and their rate of range it is nontrivial to set a timeout.

While less-specific queries do not affect the centralized architecture and increase the hit rate of superpeer architecture, they increase the overhead of structured P2P architectures. Though SADQ reduces the query cost, it significantly increases the advertising cost. Advertising cost is effectively doubled as old attribute values need to be removed from the rings to maintain consistency, as it is nontrivial to set a timeout. All the structured P2P architectures are prone to significant load balancing issues due to the small number of valid attribute values and their skewed distributions. Dynamic attributes are more important while predicting the performance of latency-sensitive applications (e.g., CASA and mobile social networks) and when resources are shared across multiple applications (e.g., CASA, GENI, grids, and clouds). Therefore, applicability of *d*-Torus is limited, as it cannot resolve queries with only the dynamic attributes. Though it seems most of the benefits of DHTs are lost under real workloads, they can still provide guaranteed RD, bounded performance, and are distributed. Moreover, as the locality of attribute values is preserved they can also find resources that fit the minimum requirements of a query. Hence, it is important to overcome their deficiencies to gain their benefits.

DHT designs typically assume that domain of attributes is much larger than the number of nodes in the DHT (i.e., $D_i \gg N$) and resources are uniformly spread within D_i. Therefore, it is preferable to add all the nodes to the DHT(s) hoping each node will index $\sim R/N$ resources. However, attributes such as *CPUArchi*, *OS*, and *NumCores* are not only skewed but also their domain sizes (i.e., number of distinct values in the domain) are much smaller (i.e., $D_i \ll N$). Even though the domain size of attributes such as *CPUFree*, *1MinLoad*, *DiskFree*, and *TxRate* are infinite, it is not useful to advertise them at a very high resolution, as users are not interested in finding very specific values. For example, advertising *DiskFree* at the granualarity of few Megabytes is not suitable as 86% queries requested *DiskFree* values of 5-1000 GB. Similarly, advertising *TxRate* at bps resolution is not useful as users are likely to query attribute ranges in tens to thousands of Kbps. Specifying attributes at a relatively low resolution is desirable (as far as it satisfies a query with the lowest $(u_i - l_i)$), as advertising cost can be reduced by ignoring minor

changes in attribute values. It was observed that highly dynamic attributes that are queried using moderate to large ranges of attribute values contributed to more than 90% of the advertising cost in SADQ based designs. Therefore, it is desirable to advertise attributes at a lower resolution (by applying a fixed or dynamic threshold) while reducing the effective domain sizes (i.e., $D_i \ll N$). Consequently, many nodes in the ring will not be able to index resources or answer queries. Moreover, adding all the nodes to a ring will unnecessarily increase the query cost, as both the average and worst-case query costs are proportional to N. Instead, it is desirable to prune nodes that do not index any resources or answer any queries (see Chapter 6) [Ba12c]. However, this does not solve the problem of few nodes having to index a large number of resources due to skewed distributions (e.g., when *CPUArchi* of 99% nodes are ×86). One alternative is to append few random bits to a *key* (i.e., hash of an attribute value) such that identical resources will be mapped to different but adjacent nodes in the ring [Al08]. While this helps to distribute the index size better, it increases the query cost and does not balance the query load, as queries have to always start from the node corresponding to l_i. In Chapter 6, we present an alternative design where large indexes are split across multiple nodes that are added orthogonal to the ring. Adding nodes orthogonal to the rings does not increase the average and worst-case path length along the ring hence the query cost is also reduced. The same concept can be also used to balance the query load distribution (see Chapter 6). Another alternative is to explore hybrid designs where desirable properties such as lower advertising cost and load distribution in centralized and superpeer architectures are coupled with rings e.g., ring of superpeers. Performance of ring-based architectures can be approximated to $O(\log N)$ by using queries that are more specific. However, in practice, it is hard to determine specific resource requirements of an application. By specifying very specific queries, users also run into the risk of not finding any useful resource. For example, a user may need only 500 MB of disk space. If his/her query specified *DiskFree* $\in$ [500 MB, 1000 MB], the query is very likely to fail as modern machines have much higher free disk space. Therefore, users are compelled to specify a large range of attribute values. It may be possible to achieve close to $O(\log N)$ query performance, if such complexities could be incorporated into the RD solution while enabling users to provide only the abstract details about their application requirements.

As the number of attributes increases, performance of all the solutions degrade due to additional memory/storage requirements and increase in advertising cost. This is a concern in heterogeneous systems like CASA and GENI, which aggregate multitude of diverse resources. One alternative is to represent multi-attribute resources using a few composite attributes. For example, BOINC uses micro-benchmark to rate nodes based on their integer and floating-point performance [An09]. Cloud computing nodes are typically rated as high-memory, high-CPU, and cluster instances. Few attributes are attractive as they simplify and reduce the cost of RD. However, it is not good at predicting performance of arbitrary applications and is too abstract to be used in latency sensitive applications such as CASA. Alternatively, applications that depend on more than one attribute cloud pick a minimum set of primary attributes that can accurately represent a resource. Dimension reduction techniques are also of interest. However, it would be challenging to resolve real-world queries that specify only a few attributes, as dimension reduction techniques are typically designed for queries that specify all the attributes. Less informative attributes such as percentage of *CPUFree* and *MemFree* may be avoided.

As the existing solutions are applicable under very specific scenarios, novel RD solutions are needed to overcome the performance and QoS issues posed by real workloads. Hybrid approaches that combine the desirable features of centralized, superpeer, and ring-based architectures while taking into account the complex resource and query characteristic have the potential to provide better solutions. It is also important to evaluate their performance using real or synthetic traces that are derived using the real-world resources and queries (see Chapter 5).

4.9 Summary

Fundamental design choices for resource discovery are evaluated using the characteristics learned from four different real-world systems. Findings show that real world, multi-attribute resource and query characteristics diverge substantially from conventional assumptions. While real world, less-specific queries are relatively easier to resolve, they introduce significant load balancing issues due to skewed resources and queries. Dynamic attributes contribute to high resource advertising cost, and their behavior is

attribute-type and system specific hence should not be ignored in performance studies. These findings indicate the need for more efficient, scalable, and robust resource discovery solutions and the importance of taking into account the specific characteristics of real-world resources and queries while designing and analyzing such solutions. Hybrid approaches that combine the desirable features of centralized, superpeer, and ring-based architectures have the potential to provide better solutions.

Chapter 5

RESQUE: MULTI-ATTRIBUTE RESOURCE AND RANGE QUERY GENERATOR

Modeling and simulation of multi-attribute resources and range queries are essential in application design, validation, and performance analysis of many distributed application domains. Novel mechanisms are presented to generate realistic synthetic traces of multivariate static and dynamic attributes of computing resources and multi-attribute range queries. The methodology is demonstrated using the resource and query traces from PlanetLab, SETI@home, EGI grid, and a distributed campus computing facility. First, random vectors of static attributes are generated using empirical copulas that capture the entire dependence structure of multivariate distribution of attributes. Second, time series of dynamic attributes are randomly drawn from a library of multivariate time-series segments extracted from the node traces. These segments are identified by detecting the structural changes in time series corresponding to a selected attribute. Time series corresponding to rest of the attributes are split at the same breakpoints to preserve their contemporaneous correlation. Finally, multi-attribute range queries are generated using a Probabilistic Finite State Machine (PFSM) that preserves the popularity of attributes and correlations among attribute values. Furthermore, a tool is developed to automate the synthetic resource and query generation process and its output is validated using statistical tests.

Section 5.1 presents the introduction and motivation. Temporal behavior and correlation of resources and queries are further analyzed in Section 5.2. Static attribute generation is presented in Section 5.3 while the dynamic attribute generation is presented in Section 5.4. Section 5.5 presents the multi-attribute, range query generation. The design of the tool that generates synthetic traces of resources and queries and its validation are presented in Section 5.6. Section 5.7 presents concluding remarks.

5.1 Introduction

Models characterizing resources, resource attributes, and demand on resources at computing/storage nodes and end hosts are vital for the design, validation, and performance analysis of many distributed application domains. Such analysis is of particular interest in collaborative Peer-to-Peer (P2P) systems [Ba12b], volunteer computing [He12], and grid [El11] and cloud computing [La12] that utilize large numbers of heterogeneous, distributed, and dedicated/voluntary resources. For example, BOINC [An09] is a volunteer computing platform that remotely executes jobs using idle computing resources. BOINC schedules jobs based on the *static attributes* (e.g., CPU speed, total memory, and presence of hardware accelerators and Graphic Processing Units (GPUs)) of nodes, as the jobs are expected to run for several hours and the system is optimized for throughput. In contrast, performance, Quality of Service (QoS), and Quality of Experience (QoE) of latency sensitive applications such as Collaborative Adaptive Sensing of the Atmosphere (CASA) [Mc09] and community cloud computing [Br09] also depend on the *dynamic attributes* (e.g., CPU utilization, free memory, and bandwidth). Collaborative P2P data fusion provides an attractive implementation choice for real-time radar data fusion, weather monitoring, and hazard prediction in CASA, as multiple data volumes are constantly being generated, processed, and pushed and pulled among groups of radars, storage, and processing nodes. CASA depends on efficient discovery and utilization of heterogeneous, dynamic, and distributed resources that are characterized by multiple attributes. Therefore, its resource discovery and scheduling algorithms must take into account both the static and dynamic attributes of resources to ensure that data are generated, processed, and delivered to end users within 30 seconds. Dynamic attributes are also becoming important in cloud computing while scheduling a Virtual Machine (VM) based on the resources utilized by other VMs running on a given physical node. For example, current service-level agreements in Infrastructure-as-a-Service (IaaS) cloud computing environments are typically based on the static attributes such as the number of allocated CPU cores and memory. However, QoS perceived by a VM is affected by the behavior of other VMs running on the same node. For example, because of the shared I/O busses, disk access and bursty traffic patterns of VMs affect each other's QoS. Hence, it is becoming important to take into account such dynamic

attributes to take better scheduling decisions. Community cloud computing [Br09] aggregates residual computing resources in Internet end hosts to build virtual cloud systems. Given that such systems rely on residual resources, respective resource discovery systems and job schedulers must take into account the dynamic attributes of hosts while scheduling latency sensitive, cloud-based applications (e.g., collaboration tools, multimedia applications, and scientific algorithms) to enhance both the QoS and QoE. Therefore, understanding the characteristics and modeling of large-scale computing platforms and nodes are essential to correctly design, validate, and analyze the performance of resource discovery solutions, job schedulers, and distributed applications.

Formal characterization of nodes and queries has received attention only recently [Ba11e, Ba12f, He12]. Characteristics of static attributes of nodes from several BOINC deployments are presented in [He12]. In Chapter 4, we presented the characteristics of both the static and dynamic resources and queries from four real-world systems. It was observed that the attributes of both the resources and queries are highly skewed [Ba11d, Ba12d] and correlated [Ba11e, Ba12f, He12]. Attribute values have different marginal distributions and change at different rates [Ba11e, Ba12f]. Queries are less specific where each query tends to specify only a small subset of the available attributes and large ranges of attribute values [Ba11e, Ba12f]. Based on the analysis of static attributes, [He12] developed a forecasting model for Internet hosts while taking into account their marginal distributions, linear correlations, and long-term evolution of static attributes (e.g., how the ratio between single-core to multi-core processors changes with time). While the static attributes are useful in evaluating systems such as BOINC, they are insufficient for evaluating latency sensitive systems that are affected by dynamic attributes and their temporal changes. Several other attempts to model computing resources are presented in [Ke04, Lu03, Su08a]; however, they also do not capture the behavior of dynamic attributes. In the absence of real-world traces and tools to generate large synthetic resource and query datasets, existing performance studies have relied on many simplifying assumptions such as independent and identically distributed (i.i.d.) attributes, uniform or Zipf's distribution of all the resources/queries, and queries specifying a large number of attributes and a small range of attribute values. For example, [Bh04, Co09b] generated static attribute values based on a

set of independent uniform distributions while [Sh07] used a set of independent bounded Pareto distributions. Moreover, [Al08, Ca04] generated a large set of nodes by replicating small datasets from computing clusters. Furthermore, multi-attribute range queries were formulated under the assumption that a query must specify all or a large subset of attributes used to describe a resource. Moreover, both the queries and attribute value ranges specified in a query were generated based on independent uniform [Ca04, Co09b, Sh07] or Zipf's distributions [Al08, Bh04]. Therefore, it is important to develop new tools to generate large synthetic datasets while preserving the statistical properties of real-world systems.

To evaluate applications and protocols for scalability beyond what is available, it becomes necessary to consider node configurations with higher number of nodes and attributes. Yet, it is still necessary to adhere to the statistical characteristics, dependencies, and temporal patterns exhibited by real-world systems. It is impractical to gather large traces with sufficient resolution and duration even for existing systems. Therefore, our idea is to gather representative information about the available traces and generate synthetic trace arrays of larger dimensionality in number and time, to meet the required goals.

We present novel mechanisms to generate random traces of nodes with both static and dynamic attributes and multi-attribute range queries. Such traces are useful in evaluating the performance of large-scale resource discovery solutions [Ba12a, Ba12c], job schedules, and distributed applications. The presented methodology is applicable to any multivariate resource and/or query dataset, and the four real-world traces used in Chapter 4 are utilized as examples. First, temporal behavior and correlation of nodes and queries are further analyzed. Our findings show that attributes of resources exhibit complex correlation patterns and time series of dynamic attributes are nonstationary. These characteristics make it non-trivial to generate random node and query traces with multiple attributes. Second, vectors of static attributes are generated using empirical copulas that capture the entire dependence structure of multivariate distribution of attributes. Third, time series of dynamic attributes are randomly drawn from a library of multivariate-time-series segments extracted from node traces. These segments are determined by identifying the structural changes in time series corresponding to a selected attribute. We present two mechanisms to identify structural changes in time series. Time series corresponding to rest of the attributes are split at

the same breakpoints and randomly drawn together to preserve their contemporaneous correlation. Finally, multi-attribute range queries are generated using a PFSM that preserves the popularity of attributes and correlations among attribute values. Furthermore, a tool is developed to automate the synthetic data generation process and its output is validated using statistical tests. Both the tool and preprocessed datasets are publicly available at [CNRL]. As the tool is independent of the set of attributes and datasets, users may also use their own datasets as the basis to generate synthetic traces.

5.2 Characteristics of Resources and Queries

We extend the analysis in Chapter 4 by further analyzing several characteristics of resources and queries that are important while generating synthetic traces. Figure 5.1 shows the cumulative distributions

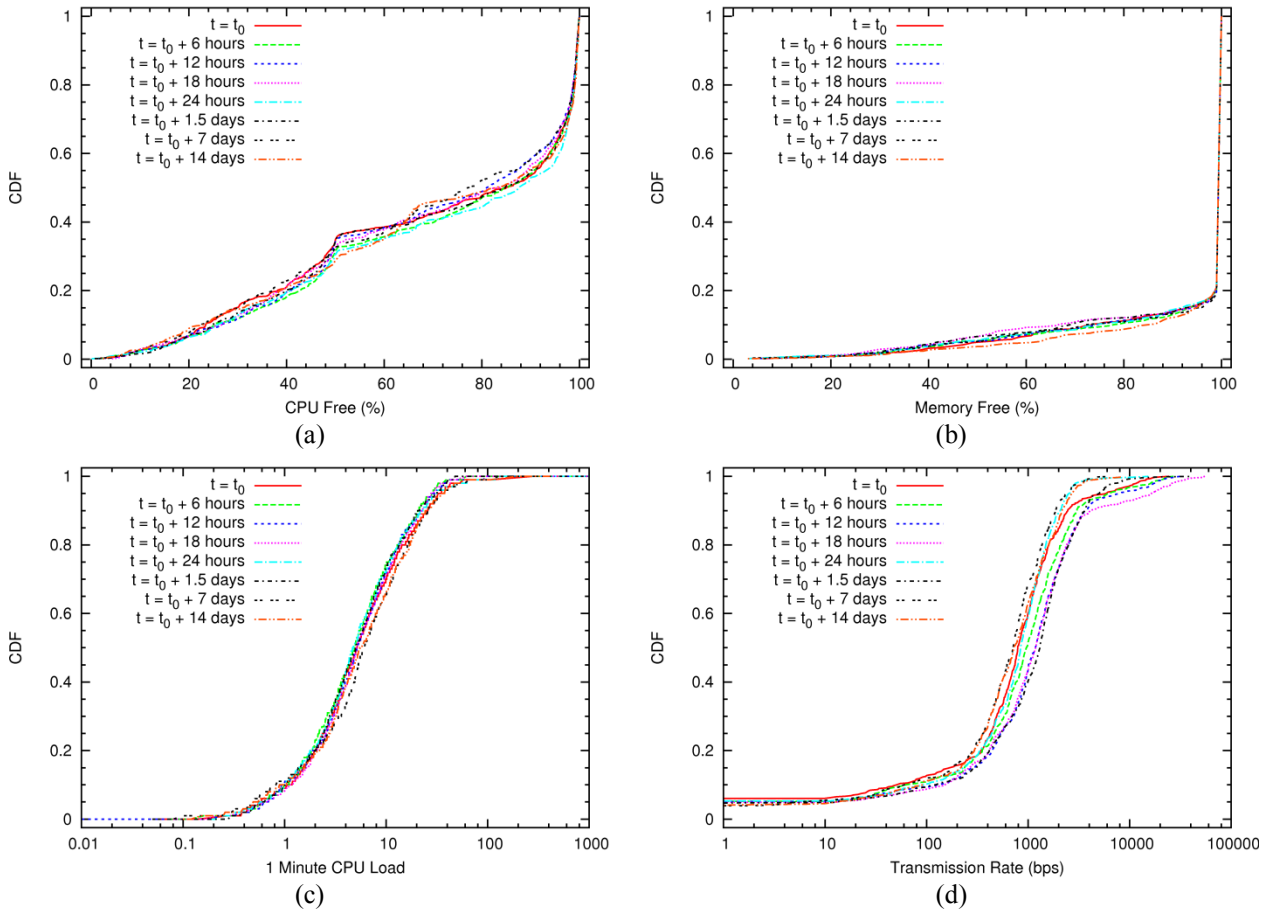


Figure 5.1 – Cumulative distributions of dynamic attributes of PlanetLab nodes sampled at different time instances: (a) Free CPU; (b) Free memory; (c) One minute CPU load; (d) Transmission rate. Starting time t_0 = 2011/02/01 5:00 UTC.

(CDFs) of four dynamic attributes of PlanetLab nodes at different time instances. It can be seen that CDFs of different samples (taken at different time instances relative to a given starting time t_0) are somewhat similar. Similar behavior is also observed (see Fig. 5.2) for the nodes within our campus (CSU dataset). Therefore, distributions derived from a particular sample of PlanetLab and CSU nodes remain valid for several days to weeks. However, as seen in Fig. 5.3, distributions of grid computing nodes (GCO dataset) show a much wider variation across samples. This could be due to the recurring busy and idle periods that are known to occur in grid computing systems [Io10]. Our goal is to generate nodes with similar overall characteristics to evaluate the impact of dynamic attributes over a moderate time span ranging from several minutes to a few weeks. Therefore, long-term trends (ranging from weeks to years) are not considered.

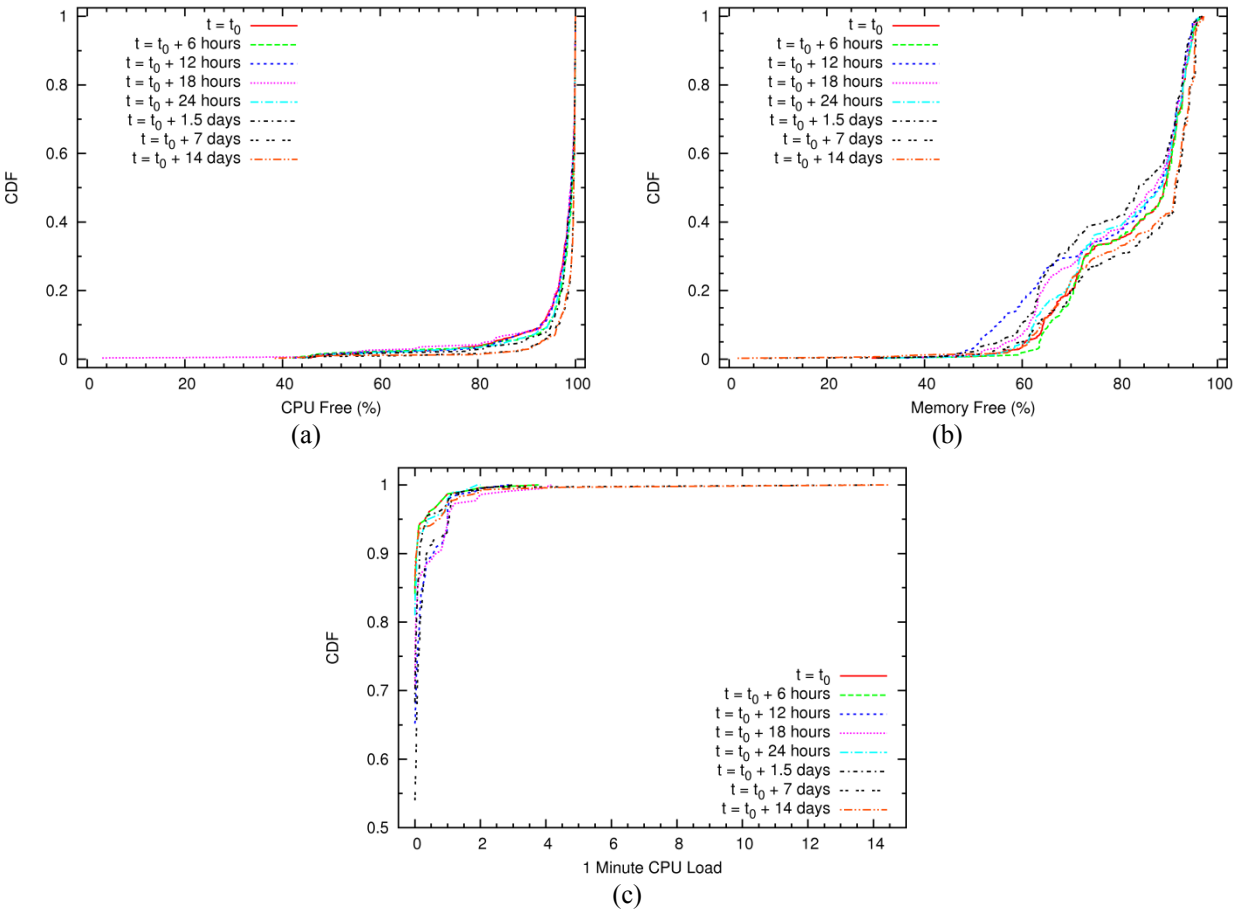


Figure 5.2 – Cumulative distributions of dynamic attributes of CSU nodes sampled at different time instances: (a) Free CPU; (b) Free memory; (c) One minute CPU load. Starting time t_0 = 2011/12/01 7:00 UTC.

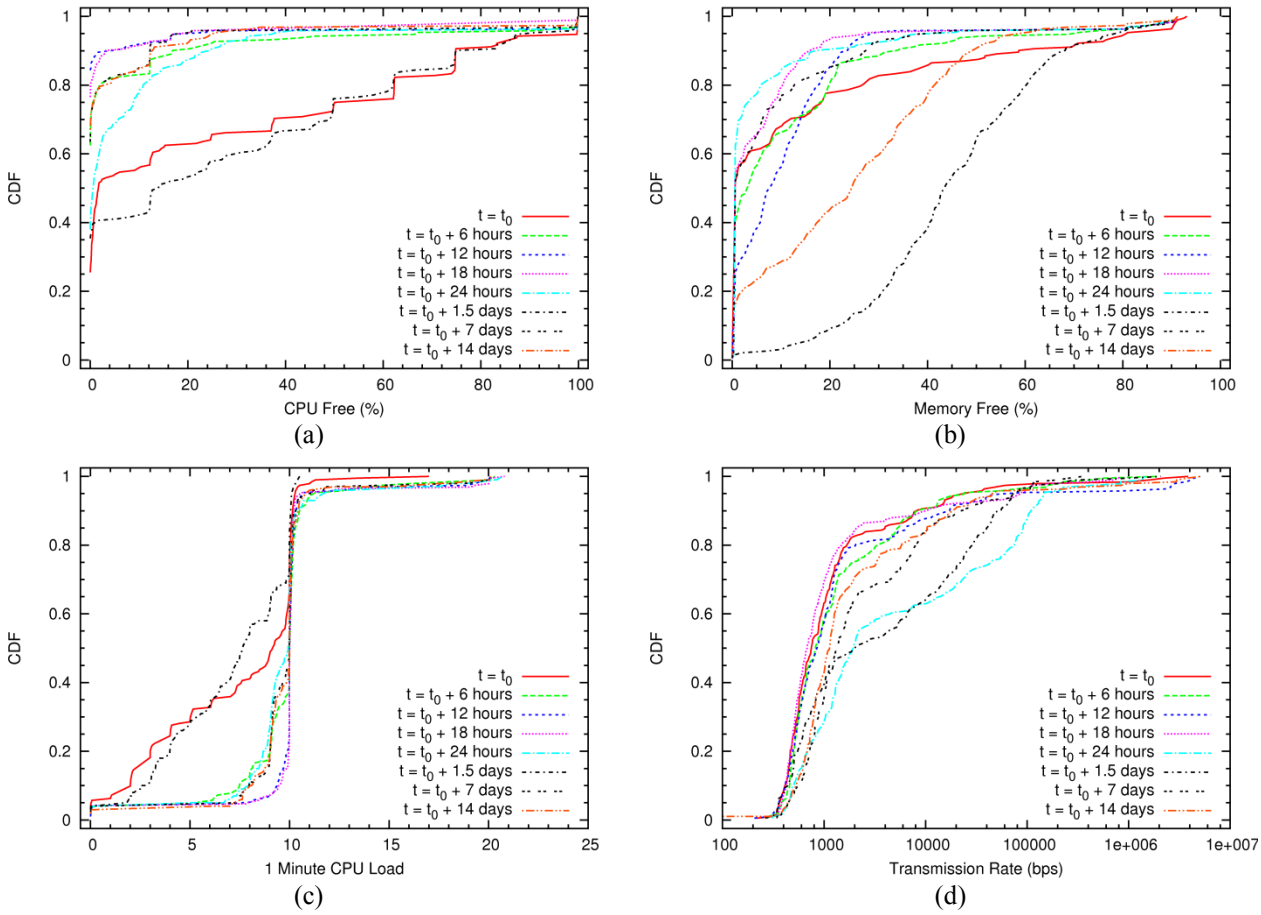


Figure 5.3 – Cumulative distributions of dynamic attributes of GCO grid computing nodes sampled at different time instances: (a) Free CPU; (b) Free memory; (c) One minute CPU load; (d) Transmission rate. Starting time t_0 = 2012/04/23 00:00 UTC.

Figure 5.4 shows time series corresponding to dynamic attributes of a selected PlanetLab node. It can be seen that the attribute values tend to change together with time, e.g., *CPUFree* reduces while *1MinLoad* increases, and *TxRate* and *RxRate* change together. This behavior is called *contemporaneous correlation* [Wo04] where observations of one time series are correlated with the observations of another time series during the same time interval. Similar behavior is also observed for dynamic attributes of GCO and CSU nodes (see Fig. 5.5 and 5.6). Note the distinct pattern in memory free (*MemFree*) time series of PlanetLab node and its structural changes. Similar patterns were observed in more than 30% of the PlanetLab nodes. GCO node in Fig. 5.5 also shows periods of high and moderate memory consumption. Nodes in CSU dataset were mostly idle (Section 4.4.1) and did not exhibit specific temporal patterns

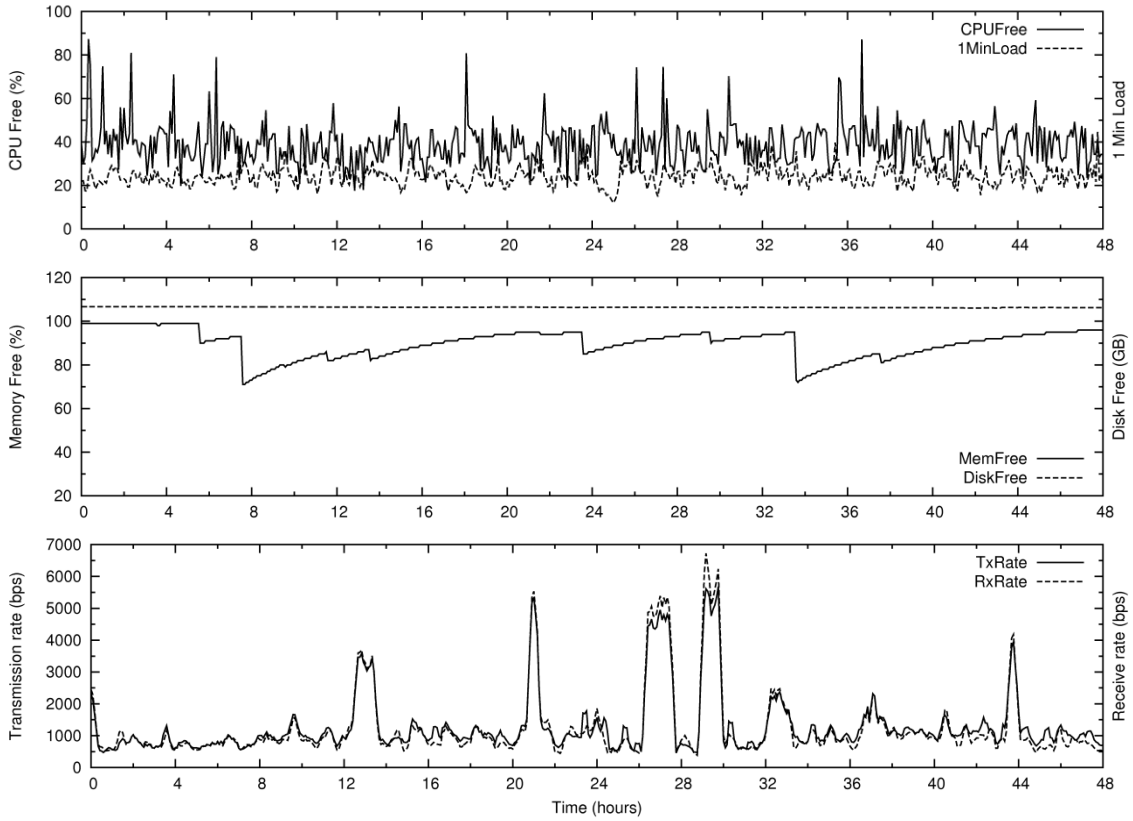


Figure 5.4 – Time series of dynamic attributes of a selected PlanetLab node. Starting time t_0 = 2011/02/01 5:00 UTC.

Table 5.1 – Normalized frequency of occurrence of attribute pairs in PlanetLab queries.

	CPUSpeed	**CPUFree**	**MemSize**	**MemFree**	**DiskFree**	**1MinLoad**
CPUFree	0.061					
MemSize	0.058	0.053				
MemFree	0.059	0.080	0.030			
DiskFree	0.060	0.098	0.058	0.099		
1MinLoad	0	0	0	0.003	0.027	
TxRate	0.059	0.085	0.053	0.075	0.084	0

except for some occasional variability in *MemFree*. Such temporal patterns in attributes also need to be preserved to accurately represent the behavior of a node.

Table 5.1 lists the frequency of occurrences of attribute pairs in PlanetLab queries. Attribute pairs such as (*MemFree*, *CPUFree*), (*DiskFree*, *CPUFree*), (*DiskFree*, *MemFree*), (*TxRate*, *CPUFree*), and (*TxRate*, *DiskFree*) appear more frequently than other possible attribute pairs. This is not surprising as these attribute pairs tend to characterize the performance and free resources of a node. Similarly, certain

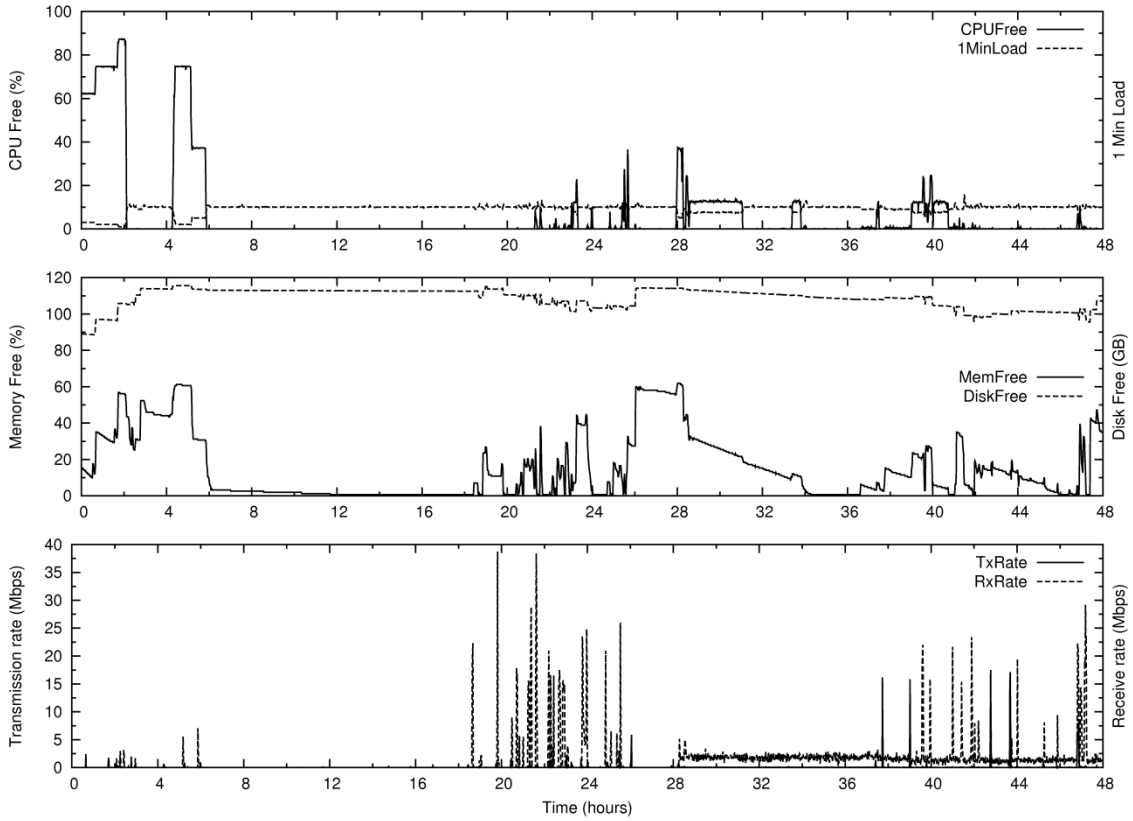


Figure 5.5 – Time series of dynamic attributes of a selected GCO node. Starting time t_0 = 2012/04/23 0:00 UTC.

combinations of three, four, five, and six attributes appeared frequently. For example, (*CPUSpeed*, *CPUFree*, *MemSize*, *MemFree*, *DiskFree*, and *TxRate*) appeared in 6% of the queries. Therefore, while generating multi-attribute range queries, we need to take into account the popularity of individual attributes (Fig. 4.13), attribute value ranges (Fig. 4.15), number of resources requested by a query (Fig. 4.12), and attributes that frequently appear together.

5.2 Generating Random Vectors of Static Attributes

Because of the strong correlation between some of the attribute pairs and specific structure in time series, attribute values of random nodes cannot be drawn from independent distributions. Therefore, we have to rely on the joint distribution of attributes. Static and dynamic attributes are handled separately,

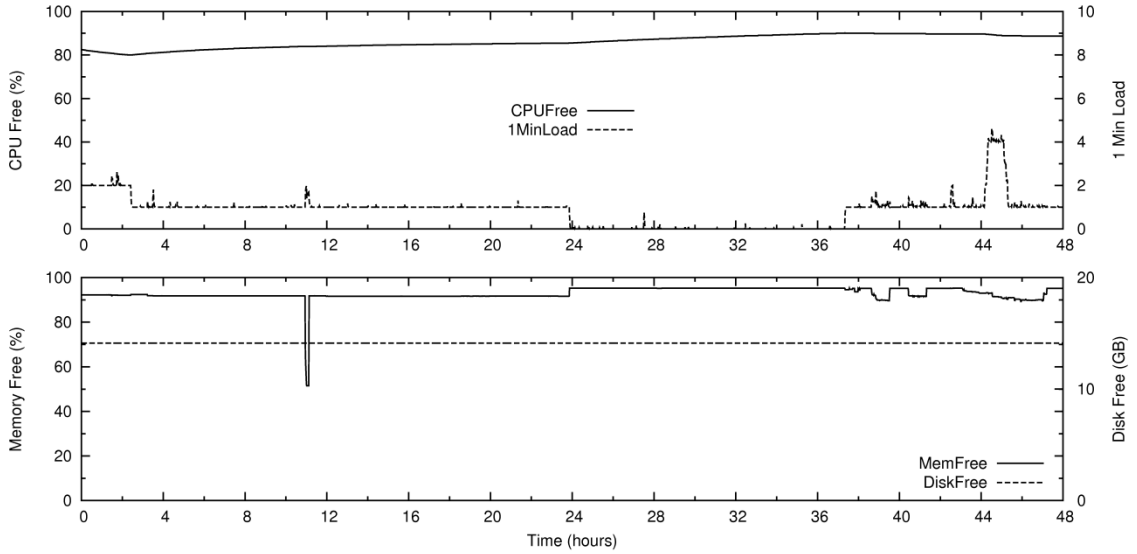


Figure 5.6 – Time series of dynamic attributes of a selected CSU node. Starting time t_0 = 2011/12/01 7:00 UTC.

as the time series of dynamic attributes are nonstationary and have specific temporal structures, as exemplified by Figures 5.4 and 5.5.

As the correlation between attribute pairs is nonlinear (see Tables 4.5 to 4.8) and complex (see Fig. 4.11), it is insufficient to use the matrix of Pearson's correlation coefficients to establish the dependence among random variables. Alternatively, copulas [Ne06] can be used to capture the entire dependence structure of multivariate distributions. Copulas are functions that couple the multivariate distribution functions to their marginal distributions. A copula $C(u)$ is a multivariate joint distribution defined on the d-dimensional unit cube $[0, 1]^d$, $(u_1, \ldots, u_n) \in [0, 1]^d$, such that every marginal distribution u_i is uniform on the interval [0, 1]. Let F denote the d-dimensional distribution function (CDF) with marginals $F_1, \ldots, F_d$. Then a copula C exist such that for all real $u = (u_1, \ldots, u_d)$:

$$F(u) = C\big(F_1(u_1), \ldots, F_d(u_d)\big) \tag{5.1}$$

Several well-known copula families are available, e.g., Gaussian and Archimedean copulas [Ne06]. However, these copulas tend to be symmetric along the axis of correlation. Alternatively, empirical copulas are useful while analyzing data with complex and/or unknown underlying distributions [Ne06,

St09]. Empirical copula also supports any number of dimensions and its bivariate frequency function is given by:

$$C_n\left(\frac{i}{n},\frac{j}{n}\right)=\frac{\text{No of pairs}(x,y)\ \text{ s.t.}\ x\le x_{(i)}\ \text{and}\ y\le y_{(j)}}{n} \tag{5.2}$$

where $1 \le i, j \le n$, $x_{(i)}$ is the ordered statistics of x, and n is the number of data points. It is proven that the empirical copula converges uniformly to the underlying copula [De78]. After deriving the copula, dependent random numbers can be generated. Those numbers can be transformed into original marginal distributions using inverse transforms.

We use empirical copulas to generate vectors of static attributes, as the joint distribution is unknown and complex. Use of empirical copulas enables us to use the empirical data directly while generalizing our approach to any multivariate dataset regardless of its dependence structure. In contrast, [He12] manually fitted probability distributions to attributes in each dataset and used the matrix of Pearson's correlation coefficients to establish the dependency between them. Moreover, [He12] had to sub-sample data, round attribute values (e.g., rounding *MemSize* to the nearest power of two), and discard samples (e.g., discarding *NumCores* values that are not powers of two) to obtain a good fit to a known probability distribution.

First, all the active nodes at a given time instance is sampled for their static attributes. Second, marginal distribution of each attribute is then transformed to a uniform random variable $\sim U(0, 1)$, e.g., using Kernel smoothing density estimation. Third, empirical copula is calculated using the multivariate version of (5.2). Fourth, dependent random numbers are then generated from the multivariate copula. Finally, random numbers are transformed back to desired marginal distributions using inverse transformation techniques, e.g., using estimated empirical distribution functions. See [St09] for additional details. If the attribute value is continuous, linear interpolation can be used to generate in-between values while performing the inverse transformation. Empirical distribution functions can be used for discrete valued attributes (e.g., *NumCores*). Figures 5.7 and 5.8(a) show the actual and generated data for PlanetLab nodes obtained using the *pwlCopula* [St09] MATLAB tool. As a comparison, data generated using the

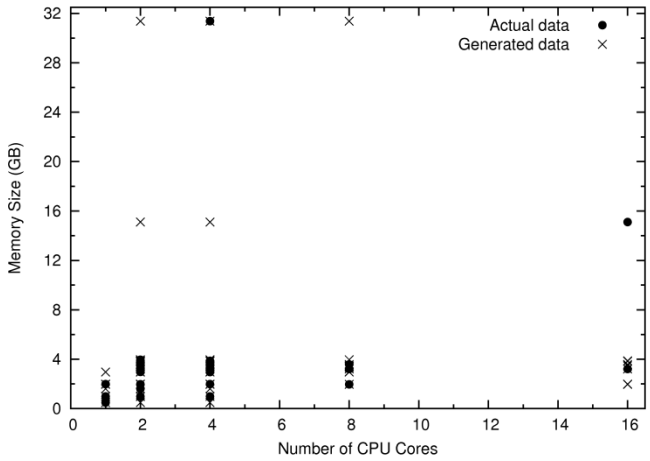


Figure 5.7 – Number of CPU cores vs. memory size of 500 random nodes generated using empirical copula. Original data based on a random sample of 100 PlanetLab nodes.

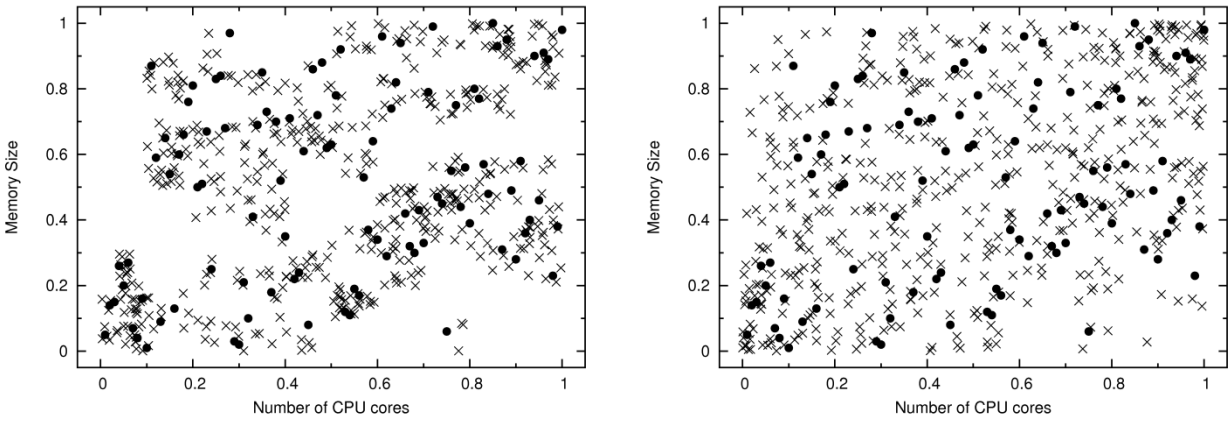


Figure 5.8 – Number of CPU cores vs. memory size of 500 random nodes generated using: (a) Empirical copula; (b) Matrix of Pearson's correlation coefficients. Data in unit scale. Original data based on a random sample of 100 PlanetLab nodes. Circles indicate the actual data while crosses indicate the generated data.

matrix of Pearson's correlation coefficients is also shown in Fig. 5.8(b). It can be seen that data generated using copula closely match the actual data. Figure 5.9 also shows the generated and actual data are also in good agreement for GCO and CSU datasets. We will statistically quantify the similarity between actual and generated attributes in Section 5.5. If only the instantaneous values of dynamic attributes are of interest, empirical copula can be simultaneously applied to both static and dynamic attributes sampled at a given time instance.

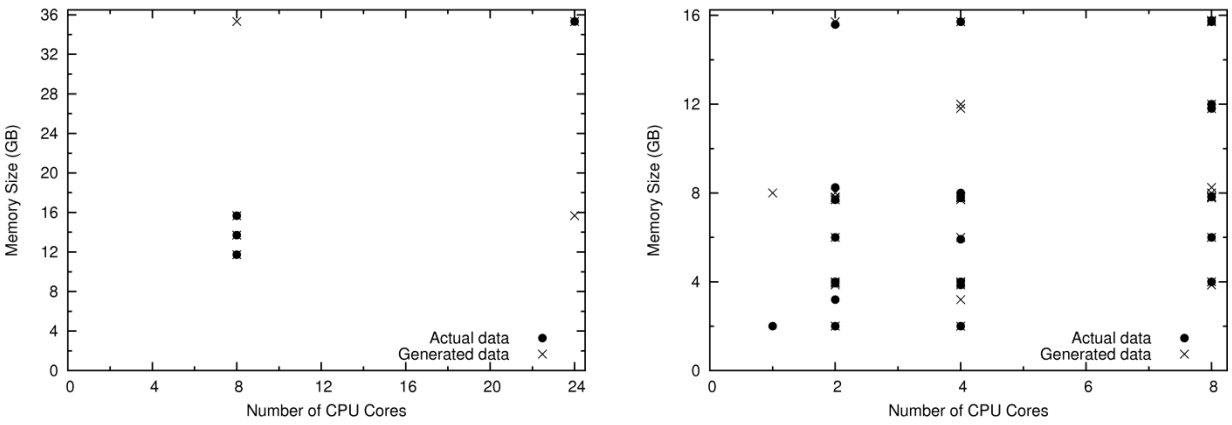


Figure 5.9 – Number of CPU cores vs. memory size of 500 random nodes generated by applying empirical copula to: (a) GCO dataset; (b) CSU dataset.

5.3 Generating Dynamic Attributes

Time varying dynamic attribute values cannot be drawn randomly from marginal distributions as the time series of some of the dynamic attributes have a specific structure (e.g., *MemFree* in Fig. 5.4 and 5.5). Failing to capture such behavior could result in over or underestimating the number of changes in attribute values over a given period. Moreover, the contemporaneous correlation between two time series needs to be preserved. Therefore, a time series of a dynamic attribute cannot be generated independently from time series of rest of the attributes. Furthermore, many structural changes in these multivariate time series make it nontrivial to model them using regression. Though it may be possible to fit a model for piecewise stationary time series (e.g., [El11] presented a method for a single time series), such an approach provides only a minor enhancement as our goal is not to predict the future behavior of nodes but to generate nodes with similar overall characteristics. Moreover, the model will not be valid over long time durations and will be specific to the mixture of applications executed in that node. Instead, it is more useful to come up with a general mechanism that can be applied across multiple computing systems and different mixtures of applications and hardware resources. Therefore, we build a library of time-series segments by identifying specific temporal patterns exhibited by dynamic attributes. This is sufficient, as our goal is to preserve the temporal variation of an attribute and its contemporaneous correlation.

We pick one of the time series to identify the structural changes, as it is nontrivial to use all the attributes simultaneously to identify structural changes. To determine a time series to use as the basis for identifying the structural changes we analyzed their autocorrelations. Figure 5.10 shows the autocorrelation of dynamic attributes of a selected node. *MemFree* and *DiskFree* time series show a much higher autocorrelation. High autocorrelation in *MemFree* is due to its specific structure. Moreover, some periodic behavior can be observed in Fig. 5.10 (a) and (b). *DiskFree* does not change drastically within a moderate

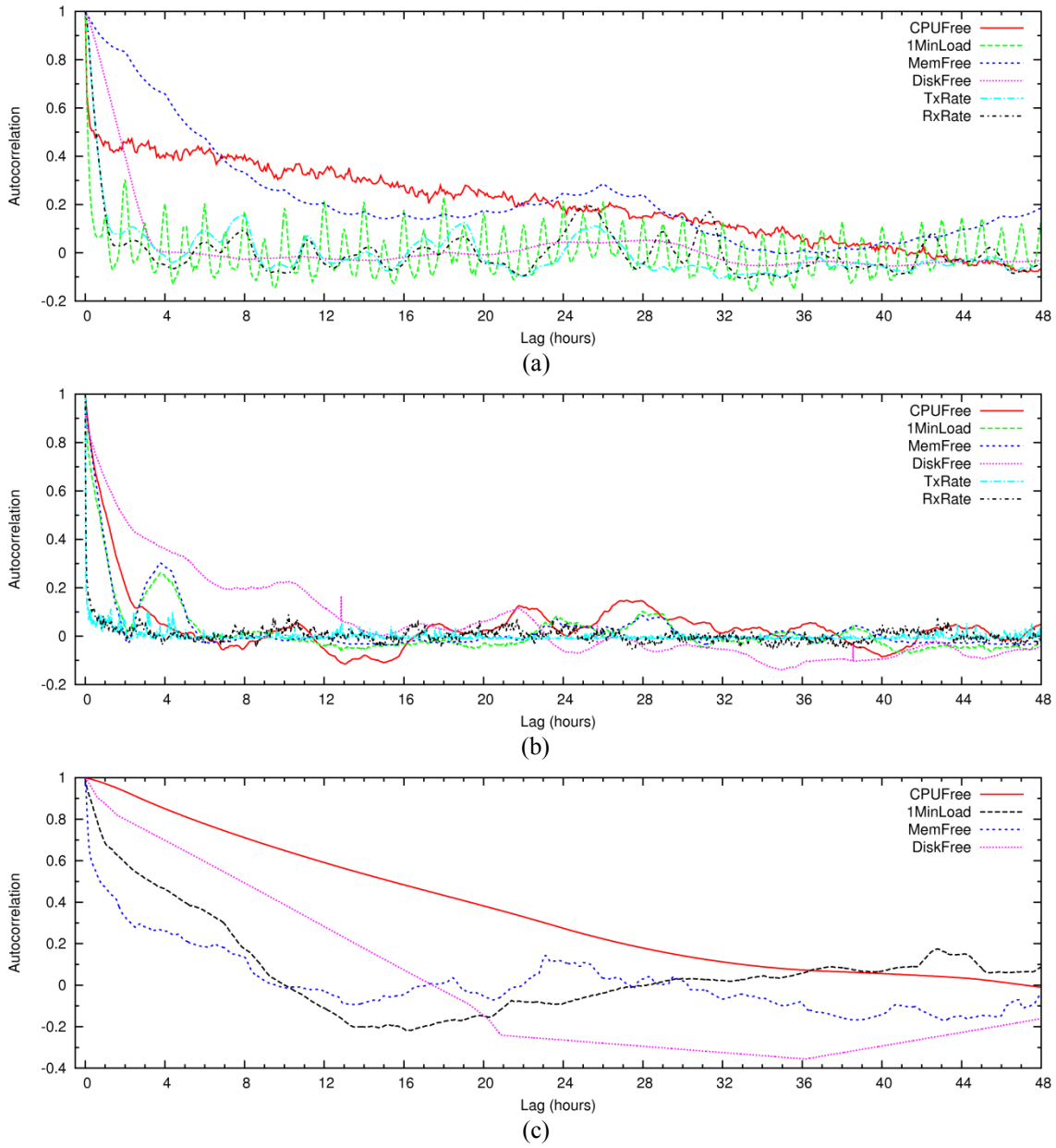


Figure 5.10 – Autocorrelation of attributes of a selected node: (a) PlanetLab; (b) GCO; (c) CSU.

time span, and consequently the autocorrelation tends to be high. Autocorrelation of other attributes is lower as they exhibit random variations. Therefore, we selected *MemFree* as the attribute based on which to partition the time series due to its distinguishable pattern. Next, we present two mechanisms to split a time series based on its structural changes. In the first approach, we check for the changes in regression coefficients while the second approach uses a derivative filter.

5.3.1 Splitting Time Series Based on Changes in Regression Coefficients

Consider the standard linear regression model:

$$y_i = x_i^T \beta_i + u_i \quad i = 1, \ldots, n \tag{5.3}$$

where at time i, y_i is the dependent variable, x_i is the vector of regressors, β_i is the vector of regression coefficients, and u_i is an i.i.d. error term. We assume that the time series has $m \geq 0$ structural changes where the regression coefficients change from one stable segment/region to another. Then (5.3) can be rewritten as:

$$y_i = x_i^T \beta_j + u_i \quad i = i_{j-1} + 1, \ldots, i_j, \; j = 1, \ldots, m+1 \tag{5.4}$$

where j is the segment number (there are $m + 1$ segments) and i is the sample index within the j-th segment. Then $\beta_j \neq \beta_{j+1}$, when $j \leq m$. We determine the structural changes by testing the null hypothesis that regression coefficients remain constant over a given segment (i.e., $H_0 : \beta_{i_j} = \beta_{i_{j-1}+1}$). Optimum number of structural changes m and their positions (a.k.a. breakpoints) can be determined using the *strucchange* package for *R* [Ze03], which uses a dynamic programming algorithm to compute the breakpoint estimates that are global minimizers of the residual sum of squares. Figure 5.11 illustrates the breakpoints obtained for the *MemFree* time series of PlanetLab, GCO, and CSU nodes. While the structural changes of the GCO and CSU nodes are properly captured, structural changes of the PlanetLab node is not so accurate. The regression coefficients are sensitive to gradual changes in the time series, as the method is designed to capture level shifts (also called steps and edges) in the time series. Consequently, this method captures

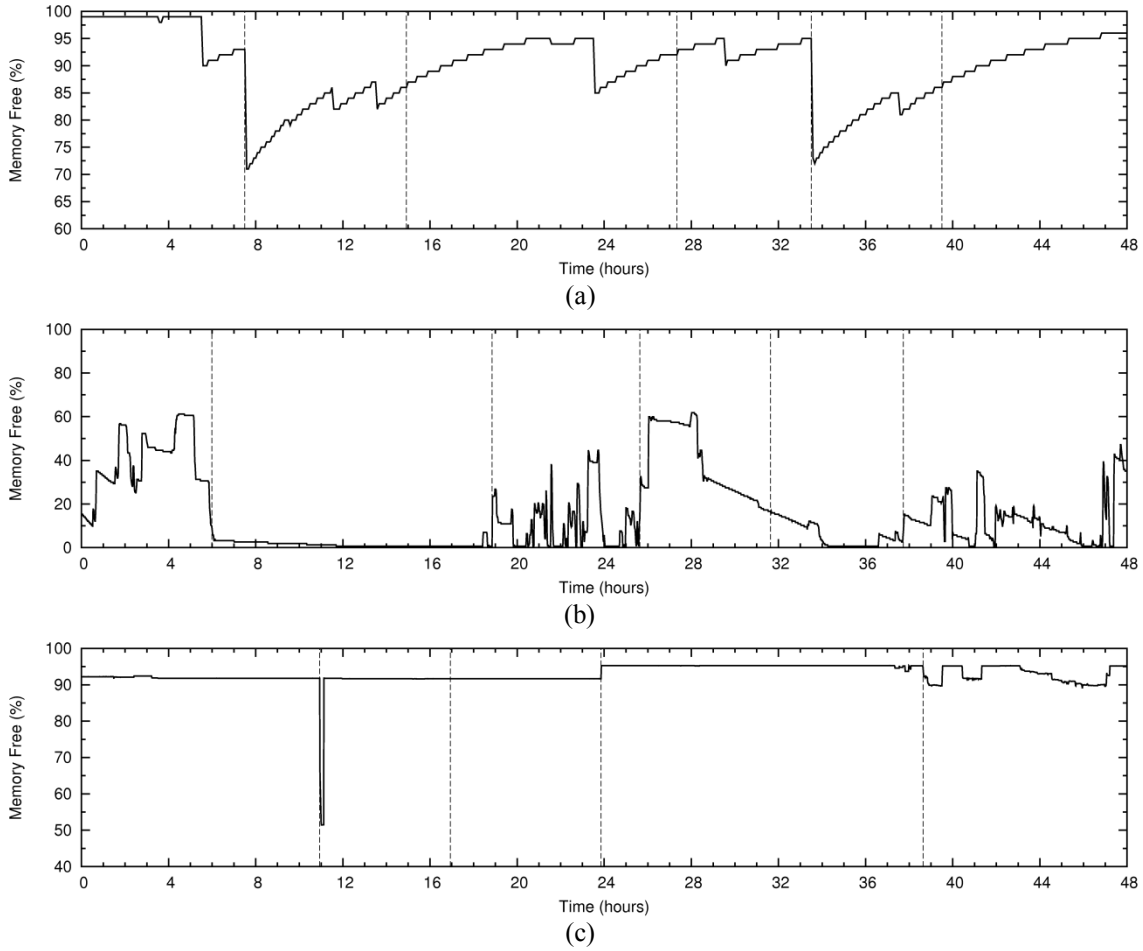


Figure 5.11 – Breakpoints identified for free memory time series of a node using the test for regression coefficients: (a) PlanetLab; (b) GCO; (c) CSU. Scattered lines indicate the breakpoints.

both gradual (e.g., the second breakpoint in Fig. 5.11(a)) and sharp (breakpoints in Fig. 5.11 (b) and (c)) level shifts in a time series. Next, we present a derivative filter that overcomes this problem.

5.3.2 Splitting Time Series Using a Derivative Filter

Derivative filters [Ma01] can be used to detect rapid changes in a time series. However, they are highly sensitivity to noise. Therefore, we first smooth the *MemFree* time series by applying a Finite Impulse Response (FIR) filter (see Fig. 5.12(b)). Coefficients of the FIR filter are set using a Hamming window [Bl58] because of its ability to minimize the maximum (nearest) side lobe and its high resolution. Then the derivative filter is applied. The absolute values of the resulting time series are shown in Fig.

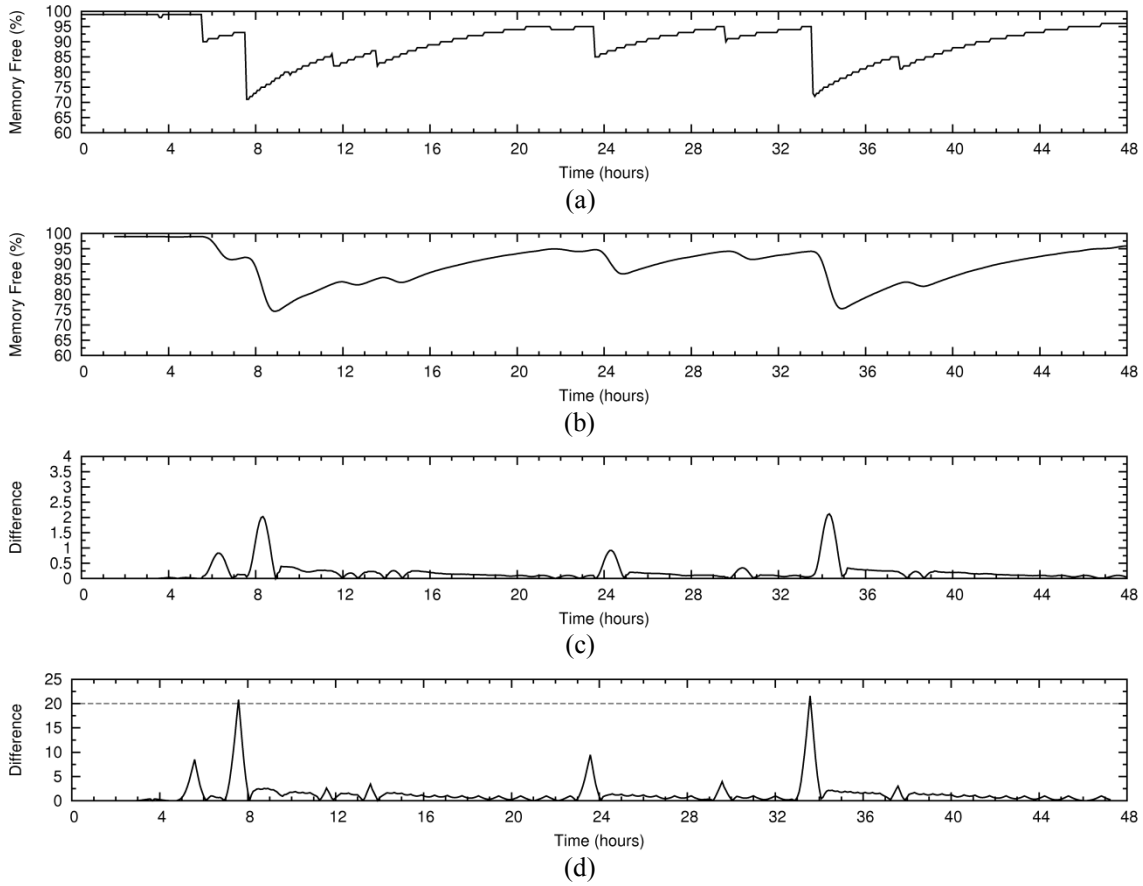


Figure 5.12 – Breakpoints identification using the derivative filter: (a) Original time series; (b) Time series after applying the FIR filter (window size 20); (c) After applying the derivative filter to time series in (b); (d) Time series after applying two sides of the FIR filter window separately and then taking the difference between the resulting time series. Scattered line indicates the threshold.

5.12(c). While the peaks are prominent, their positions are somewhat shifted from the original time series due to smoothing. This problem can be overcome by first applying the two-halves of the Hamming window separately and then taking the difference between the two smoothed time series. Figure 12(d) shows the resulting time series, where peaks align with the major structural changes in the original time series and are much sharper than the peaks in Fig. 12(c). Major structural changes in the time series can be then detected by applying a suitable threshold as shown in Fig. 5.12 (d). We empirically determined the window size and threshold for each dataset and the values are listed in Table 5.2. Structural changes detected by applying the two-halve-window-based derivative filter are shown in Fig. 5.13. It can be seen that the filter was able to capture the structural changes in PlanetLab and CSU datasets better than the

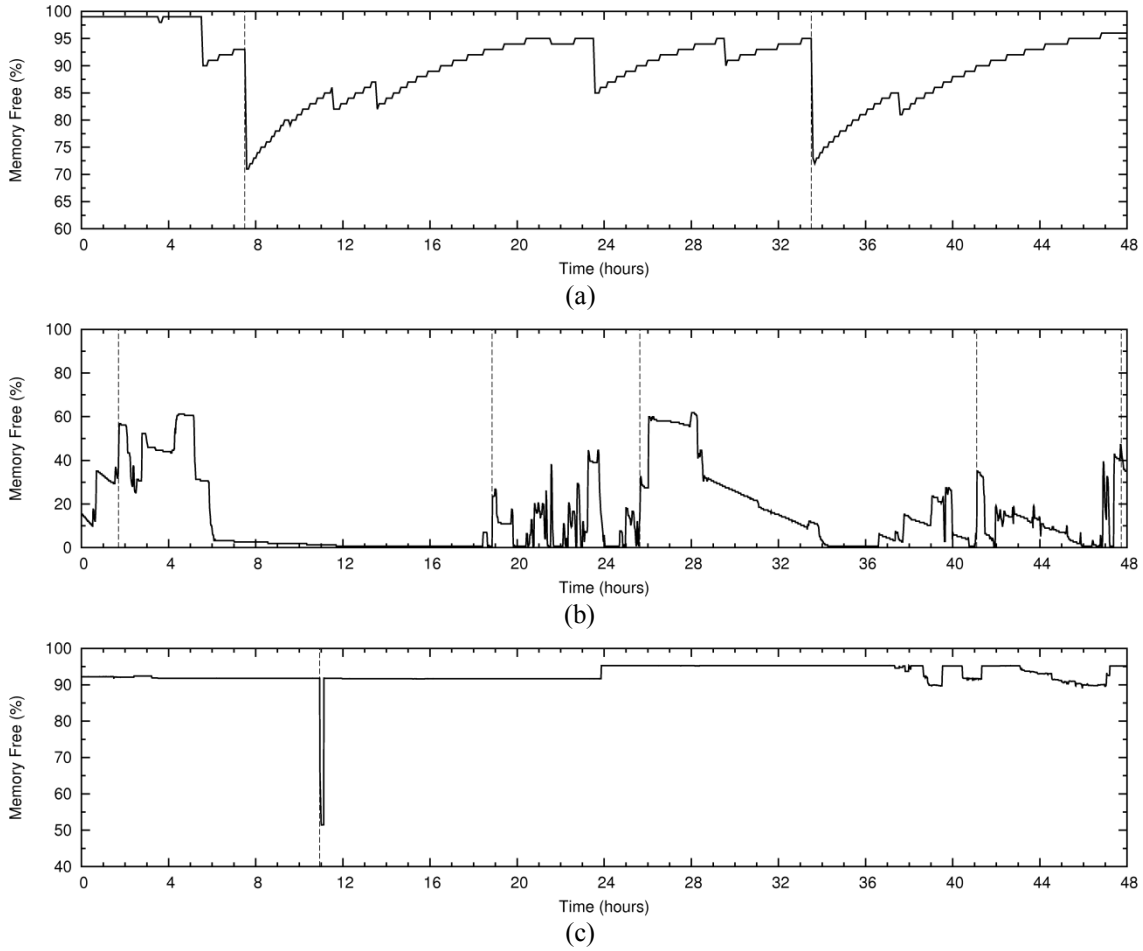


Figure 5.13 – Breakpoints identified for memory free time series of a node using the proposed two-halve-window-based derivative filter: (a) PlanetLab; (b) GCO; (c) CSU.

Table 5.2 – Windows sizes and thresholds used while splitting time series.

	PlanetLab	**GCO**	**CSU**
Window size (no samples)	19	19	19
Threshold	20%	1 GB	20%
Minimum gap between two breakpoints	12 hours	12 hours	12 hours

regression-coefficients-based method, which was more suitable for the time series from the GCO dataset (Fig. 5.11(b)). However, filter-based method requires manually determining a suitable window size and a threshold, while the solution based on the regression coefficients can calculate the optimum number of breakpoints *m* automatically. Therefore, depending on the structural properties of a time series and application requirements different methods may be used to capture their structural changes.

5.3.3 Generating Dynamic Attributes Using the Library of Time Series Segments

Once the structural changes in the *MemFree* time series are identified, time series corresponding to rest of the dynamic attributes are split at the same breakpoints. Resulting multivariate time-series segments are then collected to form a library. If desired, a stationary time series can also be split after a specific duration to increase the number of segments in the library. However, one needs to be careful not to introduce unnecessary variability by splitting the time series after a short duration as time series are concatenated randomly during the time series generation.

Dynamic attribute values are generated by randomly drawing multivariate time-series segments from the library. Longer sequences are generated by concatenating one randomly drawn segment to another. Breaking all the time series of a node at the same point and replaying them together preserve the contemporaneous correlation among attributes. Randomly mixing time-series segments corresponding to busy and idle periods is acceptable in systems such as PlanetLab and CSU where distribution of attributes tend to be stable over several hours to a few weeks. However, such random mixing is not suitable in grid and cloud computing where the entire system or a large fraction of it oscillates between busy and idle periods (Fig. 4.5). In practice, one may want to build a synthetic trace where the system is busy during given time ranges, moderately busy in another set of time ranges, and idle in the remaining times. For example, one may want to build a traces where the system is idle within the first 6 hours, it then remain busy during the next 12 hours (e.g., due to arrival of a bag of tasks), and then becomes moderately busy for another 6 hours. Such traces are useful in determining the adaptability of resource discovery solutions [Ba12c]. Such constraints can be accomplished by grouping the time-series segments in the library according to a given attribute. For example, based on the average *CPUFree*, *MemFree*, and/or *TxRate* values time-series segments can be labeled as *idle* or *busy*. Depending on the user requirements, time segments can be randomly drawn from only the subset of segments that are labeled as *idle* or *busy*. Moderate loads can be generated by randomly drawing time-series segments marked as *idle* or *busy* (if the number of *idle* and *busy* samples is similar, otherwise weights may be adjusted to get a similar number of samples from each group).

As the static and dynamic attributes are correlated, it is essential to establish the dependency between them. For example, a node with a large *NumCores* typically has higher *CPUFree* values (Fig. 4.11). Therefore, time-series segments in the library are grouped according to the *NumCores* of the corresponding node. Consequently, given the *NumCores* generated from empirical copula, the dependency between static and dynamic attributes can be established by randomly drawing time-series segments from the corresponding group. This is sufficient to establish the correlation, as correlations between other static and dynamic attributes are not strong (e.g., between *CPUSpeed* and *MemFree*, see Table 4.5).

5.4 Generating Multi-Attribute Range Queries

We use the query model described in Section 4.2 (Eq. 4.3). No noticeable correlation was observed between the number of resources (m_q) requested by a query and attributes or ranges of attribute values $[l_i, u_i]$ specified in a query. This enables us to independently generate m_q and the attributes in a query. Therefore, m_q is generated using the empirical distribution derived from PlanetLab (SWORD) queries. However, it is not straightforward to generate the attributes and ranges of attribute values in a query. Suppose following three multi-attribute queries are given as the basis to generate synthetic queries (attribute values are ignored to simplify the discussion):

Q_1 = {*CPUSpeed*}
Q_2 = {*MemFree, 1MinLoad*}
Q_3 = {*MemFree*, *CPUSpeed, TxRate*}

Suppose Q_2 appeared twice and Q_1 and Q_3 each appeared once. Large synthetic query traces can be generated using the empirical distributions derived from the number of attributes in a query and popularity of attributes, and conditional probabilities of attribute co-occurrences. It is relatively straightforward to derive the empirical distributions for the number of attributes in a query (e.g., single-attribute queries occurs with probability ¼, two-attribute queries with probability ½, and so on) and popularity of attributes (e.g., *MemFree* occur in 75% of the queries). Moreover, it is necessary to capture conditional probabilities such as P(*1MinLoad* | *MemFree*) and P(*TxRate* | *MemFree, CPUSpeed*). However, capturing the conditional

probabilities of co-occurrence of attributes becomes difficult as the number of attributes in a query and possible ranges of attribute values increase. For example, 7% of the PlanetLab queries specified more than five attributes, see Fig. 4.12. We overcome these issues by building a Probabilistic Finite State Machine (PFSM) while interpreting attributes as a set of states and attribute co-occurrences as state transitions weighted by their frequency of occurrences.

We selected a PFSM because of its ability to capture the structure in a given set of queries and assign probabilities to that structure. It is also proven that a PFSM can represent the same distributions as those modeled by the hidden Markov model [Vi05]. Thus, a PFSM can capture the distributions of number of attributes in a query and popularity of attributes, as well as conditional probabilities of attributes co-occurrences. Probabilities assigned to state transitions allow us to describe the behavior of a PFSM as a random process, which can be used to generate a random query. PFSMs have been applied to generate random workloads in web-based applications [Ba11b] where the user behavior is modeled as transitions among a set of states such as login into a web page, viewing the calendar, adding a new entry to the calendar, and logout. Similarly, attributes in a query can be represented as a set of states and their co-occurrences can be modeled as state transitions. However, multi-attribute queries do not have well defined *START* and *END* states as web-based applications (e.g., login and logout). Therefore, we assumed virtual *START* and *END* states, and then interpreted the first attribute specified in a query as a transition from the *START* state and the last attribute in the query as a transition into the *END* state. Figure 5.14 depicts the corresponding PFSM for the above three queries. Following distinct queries can be generated using this PFSM:

q_1 = {*CPUSpeed*} 1/8
q_2 = {*CPUSpeed*, *TxRate*} 1/8
q_3 = {*MemFree*, *1MinLoad*} 1/2
q_4 = {*MemFree*, *CPUSpeed*} 1/8
q_5 = {*MemFree*, *CPUSpeed*, *TxRate*} 1/8

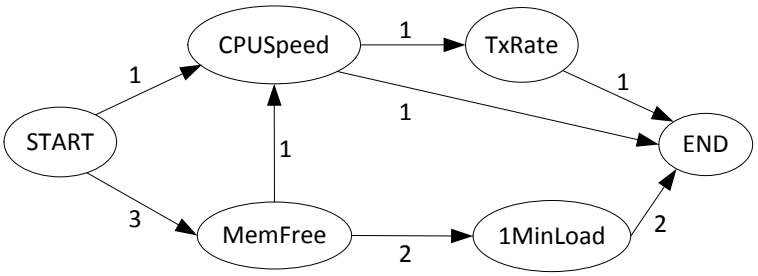


Figure 5.14 – Probabilistic finite state machine for queries Q_1, Q_2, and Q_3.

Their probability of occurrences are indicated on the right. PFSM generates two queries q_2 and q_4 that were not among the original queries. There queries are also valid as there is a possibility of specifying *CPUSpeed* with *TxRate* and *MemFree*. Therefore, by applying a PFSM we can also generate many queries that are likely to occur in practice. Ranges of attribute values defined in queries can be represented as a set of sub-states. For example, two queries with *CPUSpeed* ∈ [1.5, 3.0] and *CPUSpeed* ∈ [2.0, MAX] can be defined as two sub-states within the main state *CPUSpeed*.

Suppose the attributes in the original query Q_2 is swapped as Q_2 = {*1MinLoad*, *MemFree*}. This is possible as it is not necessary to specify attributes in a particular order. Then the corresponding PFSM is given in Fig. 5.15. This PFSM is slightly different from Fig. 5.14 and produces the following set of distinct queries:

q_1 = {*CPUSpeed*}	1/8
q_2 = {*CPUSpeed*, *TxRate*}	1/8
q_3 = {*MemFree*, *1MinLoad*}	1/3
q_4 = {*MemFree*, *CPUSpeed*}	1/24
q_5 = {*MemFree*, *CPUSpeed*, *TxRate*}	1/24

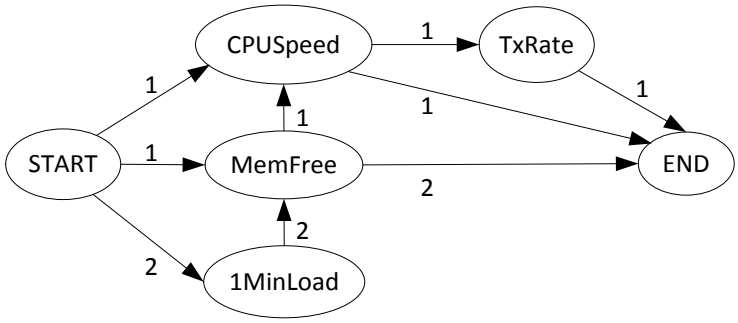


Figure 5.15 – Probabilistic finite state machine for queries when attributes in Q_2 is swapped.

q_6 = {*MemFree*} 1/6

q_7 = {*1MinLoad*, *MemFree*, *CPUSpeed*} 1/12

q_8 = {*1MinLoad*, *MemFree*, *CPUSpeed*, *TxRate*} 1/12

It produces three more queries q_6 to q_8 in addition to the ones generated by the PFSM in Fig. 5.14. This is a consequence of not having well defined *START* and *END* states in queries. Therefore, resulting PFSM is sensitive to how the states are coded. While this could result in generation of queries with invalid combinations of attributes, it also offers the opportunity to generate different mixtures of queries by using different coding conventions. For example, in addition to coding attributes based on the order they appear in queries, attributes in a query may be shuffled randomly or sorted in the ascending or descending order before building the PFSM. The problem of generating invalid queries can be handled by either ignoring those queries ones they are generated or modifying the PFSM to prevent the generation of such queries. Standard practice of preventing such illegal states is to break a conflicting state(s) into multiple states. Suppose it is invalid to generate q_6 with only *MemFree*. Then the state transition from *MemFree* to *END* can be removed by representing *MemFree* as two different states (one that has a transition from *1MinLoad* and another from *START*). Then the updated PFSM based on Fig. 5.15 is given in Fig. 5.16 where *MemFree* is broken into two states as *MemFree* and *MemFree*$'$. Similarly, suppose q_8 is invalid. Even then, *MemFree* is the common state in original queries Q_2 and Q_3 that lead to the generation of q_8. Hence, by breaking MemFree into two states as in Fig. 5.16 q_8 can be also avoided. We may prevent the generation of new query combinations (which are valid) while trying to avoid one of the invalid queries (e.g., while trying to avoid q_6 we also prevent the generation of q_8). This can be overcome by reorganizing

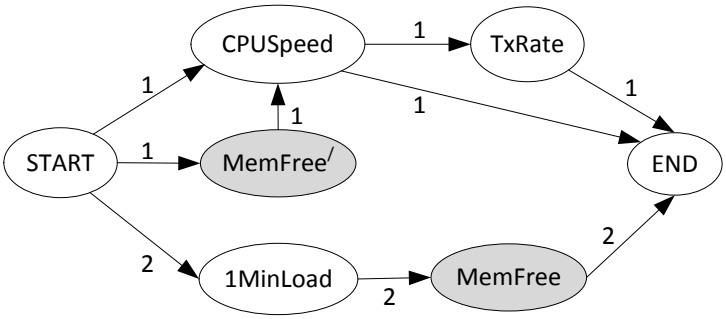


Figure 5.16 – Probabilistic finite state machine modified to avoid invalid query q_6 in Fig. 5.15.

the state diagram such that only the invalid state is removed while preserving other potential states. However, in practice, it would be difficult to modify a large state diagram manually and identify all possible valid state transitions. Hence, for practical reasons, it may be easier to discard invalid queries after they are generated.

5.5 ResQue – Resource and Query Generator

A tool named ResQue (**Res**ource and **Que**ry generator) has been developed to automate the synthetic resource and query generation process. ResQue can generate a set of random nodes/resources with a subset of static and dynamic attributes (see Fig. 5.17). Dynamic attribute values can be generated up to a given time (typically between a few minutes to weeks), and the sampling interval must be an integer multiply of sampling interval of data used as the basis. Moreover, if desired, users can also specify during which time intervals the generated traces should reflect a system that is busy, moderately busy, or idle. Figure 5.18 illustrates the process of generating resources by combining the empirical-copula-based static

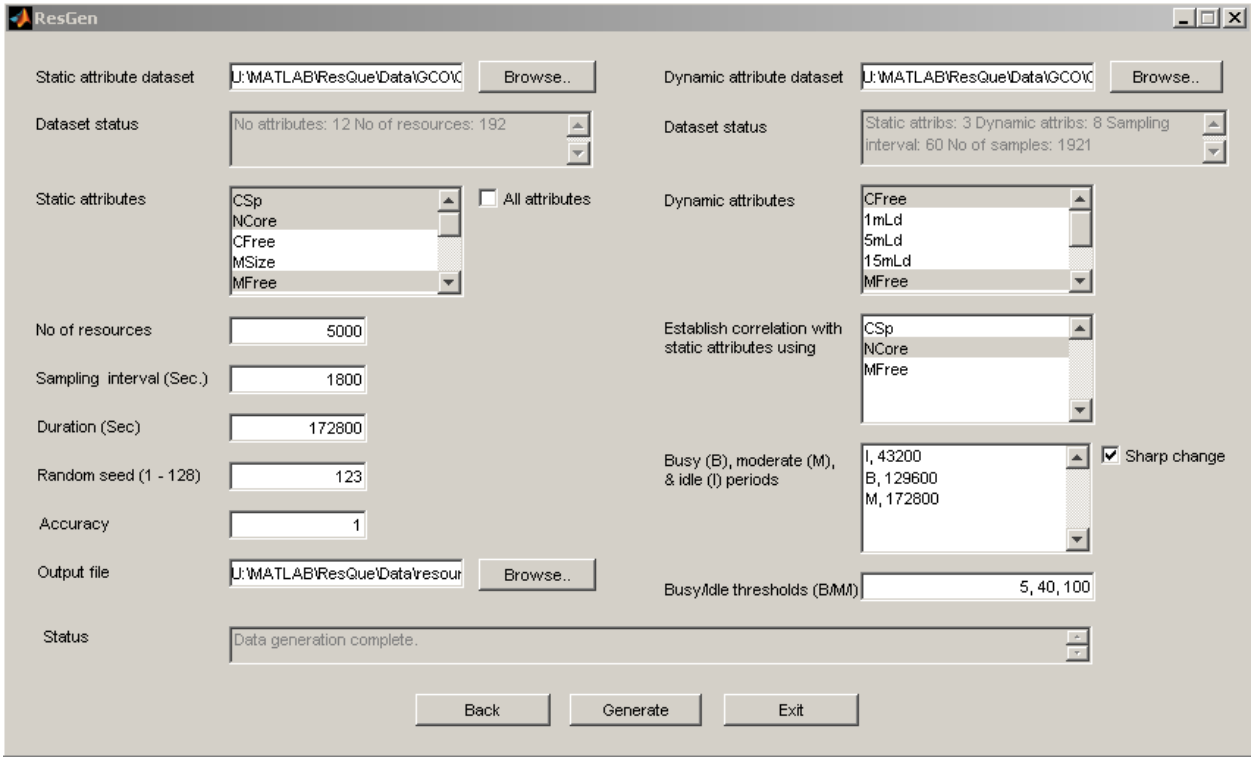


Figure 5.17 – Screenshot of ResQue's multi-attribute resource generator.

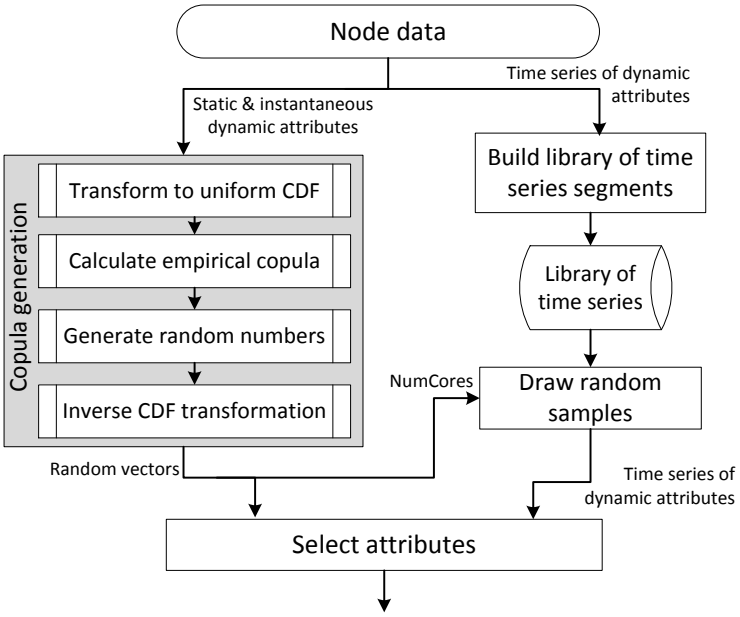


Figure 5.18 – Flowchart of random resource generation.

attribute generation and time-series-library-based dynamic attribute generation. First, all the active nodes at a given time instance is sampled for their static and instantaneous values of dynamic attributes (optional). The instantaneous values of dynamic attributes are supported as they are useful in evaluating certain scheduling algorithms. Second, marginal distribution of each attribute is then transformed to a uniform random variable ~$U(0, 1)$, e.g., using Kernel smoothing density estimation. Third, empirical copula is calculated. Fourth, dependent random numbers are then generated from the multivariate copula. Finally, random numbers are transformed back to desired marginal distributions using inverse transformation techniques, e.g., using estimated empirical distribution functions. If the attribute value is continuous, linear interpolation may be used to generate in-between values while performing the inverse transformation. Empirical distribution functions may be used for discrete valued attributes. *NumCores* from copula is feed to the random sampling module to establish the dependence between static and dynamic attributes as the correlation between *NumCores* and *MemFree* was high (ResQue also supports other attributes such as *CPUSpeed* and *MemSize*).

Screenshot of the query generator is shown in Fig. 5.19. Number of resources requested by a query m_q (see Section 4.2) are generated independent of the attributes in a query as they are not correlated. m_q

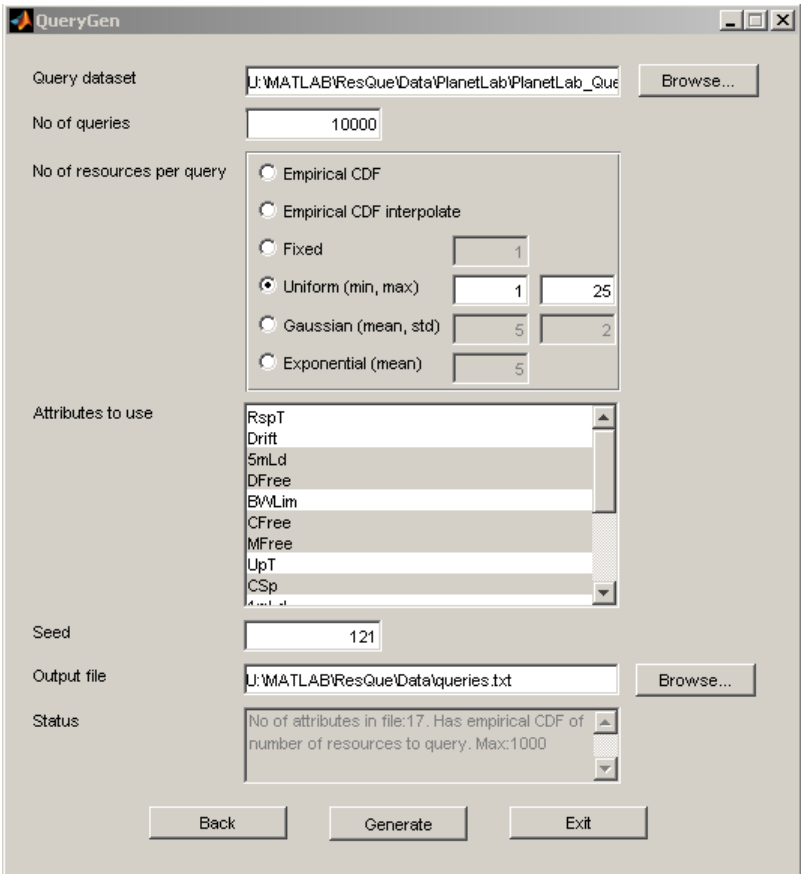


Figure 5.19 – Screenshot of ResQue's multi-attribute range query generator.

can be generated based on the empirical distribution extracted from an actual query dataset or using a set of random number generators, e.g., uniform, Gaussian, exponential, etc. Multi-attribute range queries are generated using the PFSM, which takes a set of state transitions and their frequencies as the input. Users may select which subset of the states to use while generating the queries. Both ResQue and the four pre-processed datasets that are fed into the tool as the basis to generate data are publicly available at [CNRL]. These datasets include additional attributes such as *5MinLoad*, *15MinLoad*, and response time. As our approach is independent of the dataset, users may use more recent datasets or use their own datasets extracted from other systems. Multi-attribute time series library may be built using the two methods described in Section 5.3 or users may use their own techniques. Several utilities are also provided to simplify the pre-processing of new datasets.

5.6 Validation

Statistical properties of synthetic data generated by ResQue are validated as follows. Attribute values of PlanetLab, GCO, and CSU nodes that were active over a week is used as the input to the tool. We generated 5,000 random nodes with static and dynamic attributes over a two-week period. While transforming the random numbers generated using copula to original distributions, linear interpolation was used for *CPUSpeed* and *MemSize*, and empirical distribution was used for *NumCores*. Stationary time series are split every 24 hours to create more segments. However, it did not significantly vary the distribution of number of attribute changes over a given time interval. To prevent the addition of many small segments to the time-series library, gap between two structural changes is set to at least 6 hours while using both the regression-coefficients-based (*strucchange* package) and derivative-filter-based methods.

Figure 5.20 plots the distribution of a selected set of attributes from PlanetLab nodes and the attributes generated using ResQue. It can be seen that the attributes of generated nodes closely match the distributions observed in Section 5.2. Mean (μ) and standard deviation (σ) of *CPUSpeed* (included in the figure) derived using copula is similar to the actual data. Even for the *CPUFree*, *MemFree*, and *TxRate* error in μ is 3-18% which is expected as the distribution of time series varies between samples as seen in Fig. 5.1. The Kolmogorov-Smirnov test (KS-test) with a significance level of 0.05 further confirmed that the synthetic data satisfy the distributions of original data. In addition to meeting μ and σ, synthetic traces also mimic the true variations/patterns inherent in time series. While the derivative-filter-based method was more accurate in capturing the structural changes in PlanetLab nodes than the regression-coefficients-based method, no significant difference was observed between the distributions of instantaneous attribute values (see Fig. 5.20 and 5.21). Figure 5.22 shows a similar behavior for CSU dataset. *CPUSpeed* of generated SETI@home nodes is shown in Fig. 5.23.

Though GCO nodes were mostly busy, Fig. 4.5(b) and 5.3 also show a wide variation in attribute values with time. Therefore, attribute value distributions that are consistent with the actual data cannot be

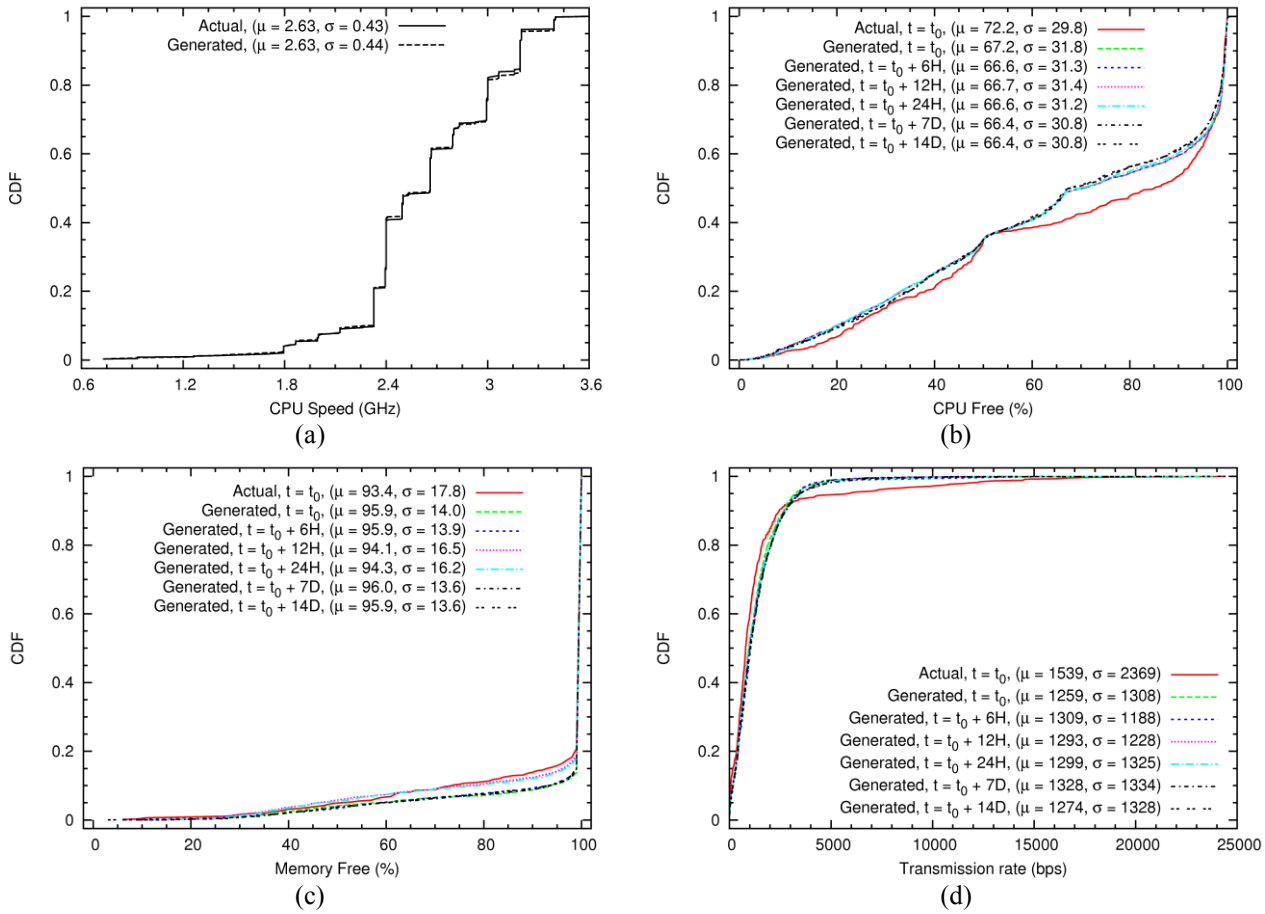


Figure 5.20 – Comparison of attributes of PlanetLab nodes and nodes generated using ResQue: (a) CPU speed; (b) Free CPU; (C) Free memory; (d) Transmission rate. Time series split using the regression-coefficients-based method. Starting time t_0 = 2011/02/01 5:00 UTC. *H* – Hours and *D* – Days.

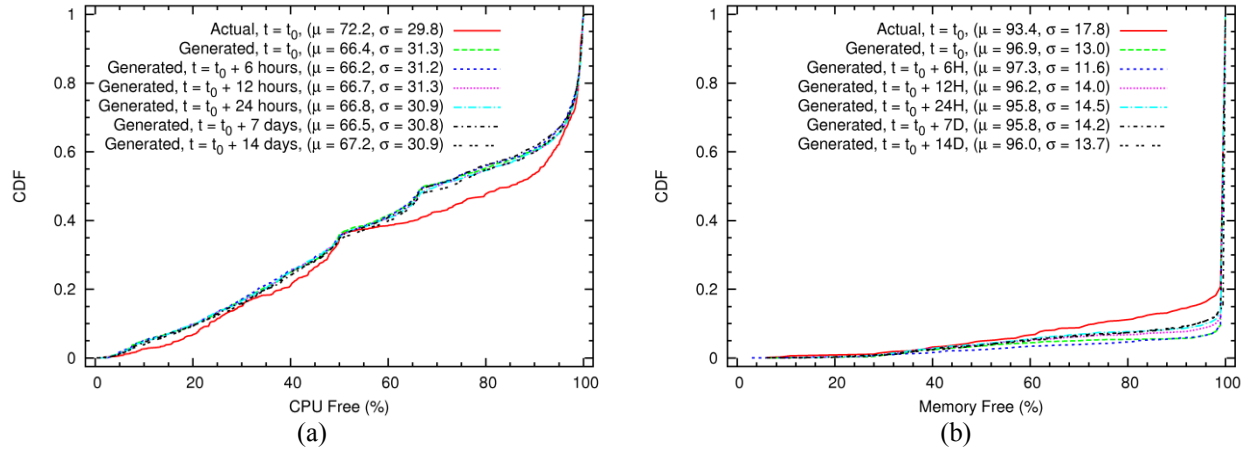


Figure 5.21 – Comparison of dynamic attributes of PlanetLab nodes and nodes generated using the derivative filter-based method: (a) Free CPU; (b) Free memory. Starting time t_0 = 2011/02/01 5:00 UTC.

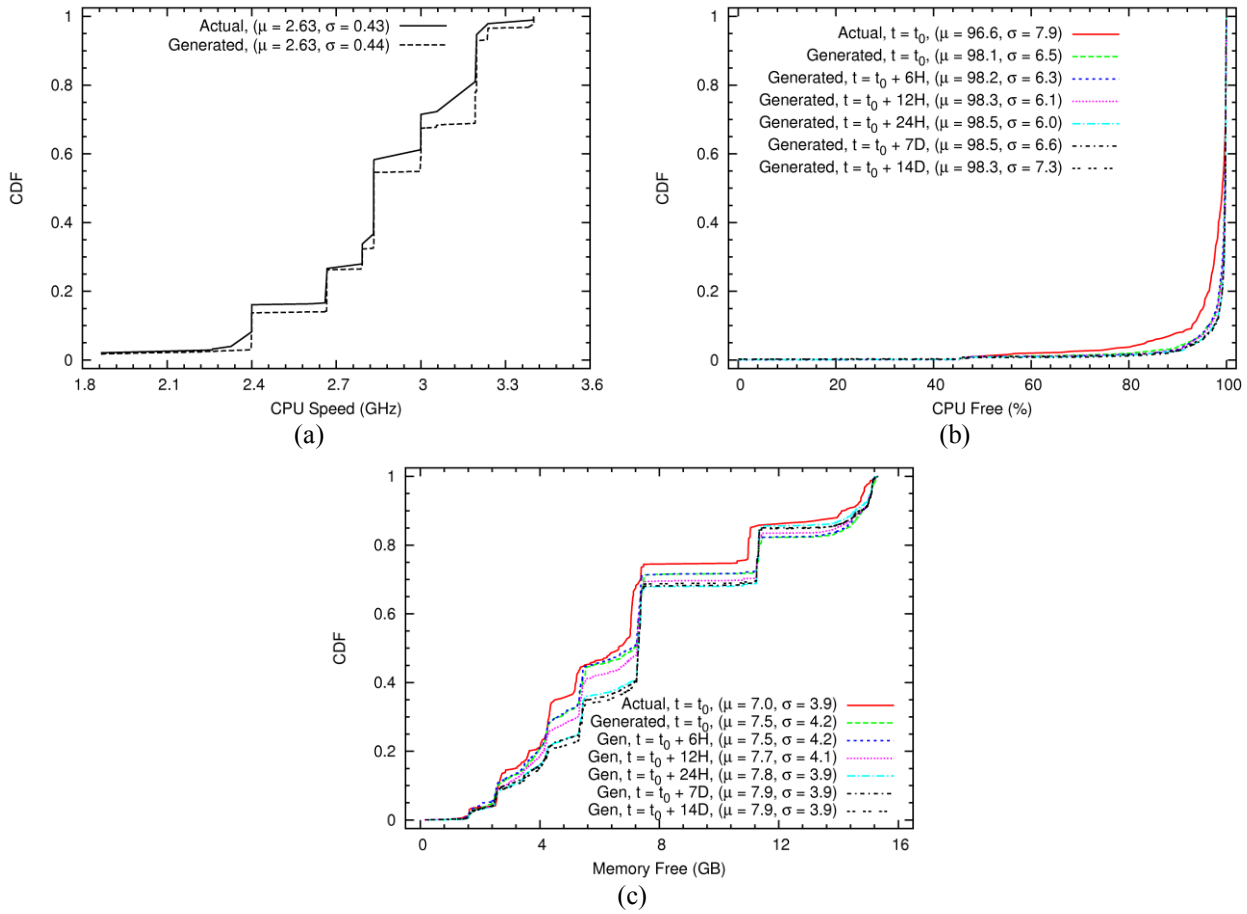


Figure 5.22 – Comparison of attributes of CSU nodes and nodes generated using ResQue: (a) CPU speed; (b) Free CPU; (C) Free memory. Time series split using the regression-coefficients-based method. Starting time t_0 = 2011/12/01 7:00 UTC.

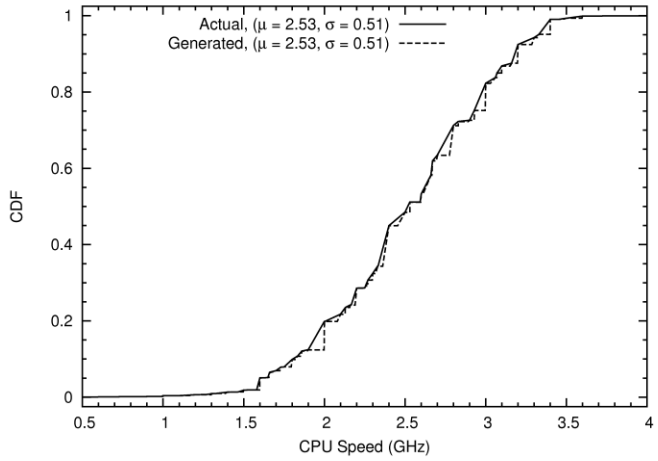


Figure 5.23 – Comparison of CPUSpeed of SETI@home nodes and nodes generated using ResQue.

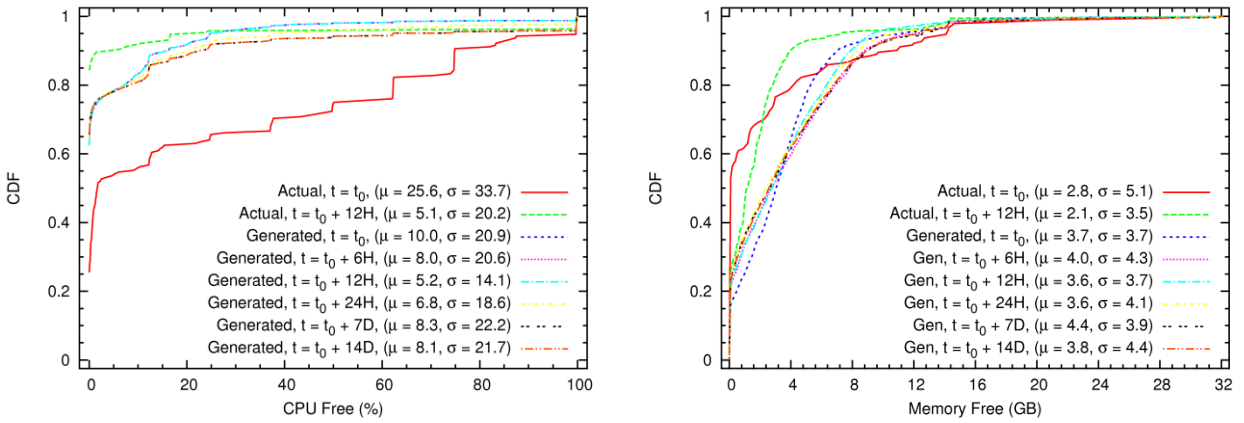


Figure 5.24 – Comparison of dynamic attributes of GCO nodes and nodes generated using ResQue: (a) Free CPU; (b) Free memory. Time series split using the regression-coefficients-based method. Starting time t_0 = 2012/04/23 00:00 UTC.

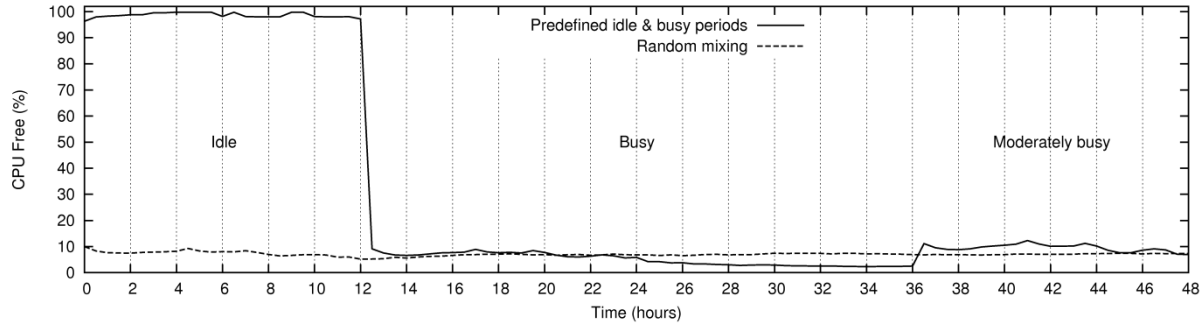


Figure 5.25 – Generation of resource traces with predefined idle and busy periods. 0-12 hours – idle, 12-36 hours – busy, and 36-48 hours – moderately busy.

generated by randomly mixing time-series segments (see Fig. 5.24). As the nodes were mostly busy, CDFs of generated dynamic attributes (sampled at different time instances) were somewhat similar to the actual attribute distributions sampled at busy times, e.g., after 12 hours. This problem can be overcome by explicitly defining busy, moderately busy, and idle periods. For example, Fig. 5.25 shows the average *CPUFree* of a system that is defined to be idle within the first 12 hours, busy during next 24 hours, and moderately busy during the remaining 12 hours. For comparison, results from random mixing of time-series segments are also presented. Therefore, ResQue can also generate node traces with different load distributions.

Using PlanetLab query data as the basis 100,000 multi-attribute range queries were also generated. In addition to coding attributes based on the order they appear in queries, attributes in a query were

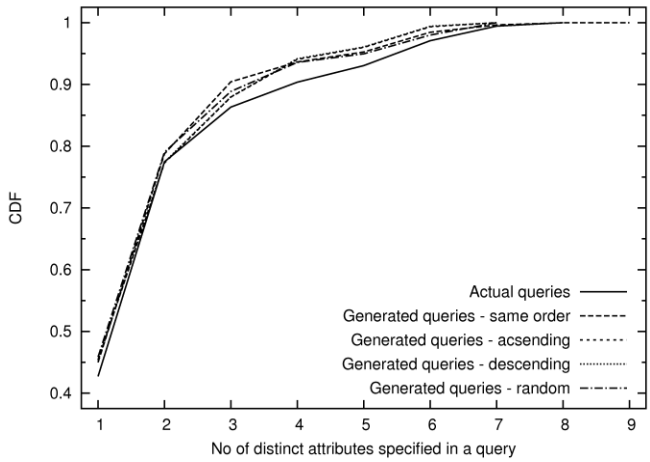


Figure 5.26 – Comparison of number attributes in a query under different coding conventions.

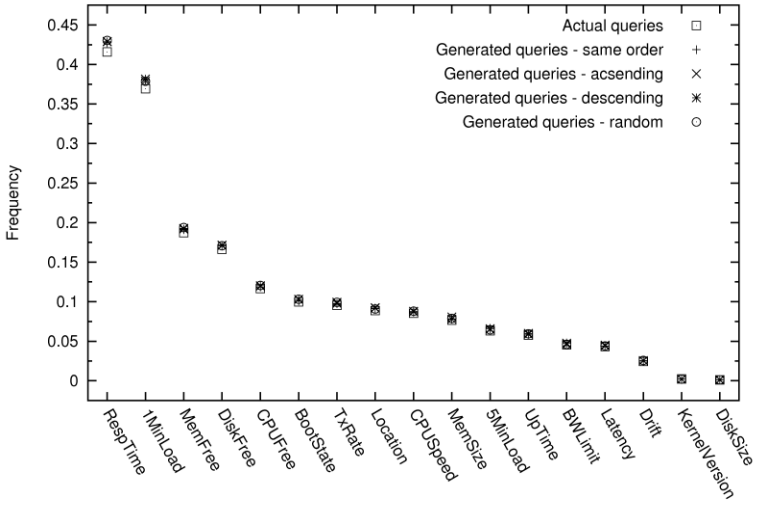


Figure 5.27 – Popularity of attributes generated using ResQue.

also shuffled randomly or sorted in the ascending or descending order before building the PFSM. Figures 5.26 and 5.27 show that the distribution of number of attributes in a query and popularity of attributes also are in good agreement with the original queries. Distributions of number of attributes in a query satisfied the KS-test with a significance level of 0.05. Table 5.3 shows that the frequency of occurrences of attribute pairs is similar to the original queries in Table 5.1 (differences between normalized frequencies of attribute pairs range from 0.0 to 0.023). Differences between normalized frequencies among different coding conventions range from 0.0 to 0.019. Therefore, different coding conventions generate somewhat different query combinations. These findings indicate ResQue can generate static and dynamic attributes

Table 5.3 – Normalized frequency of occurrence of attribute pairs in queries generated using ResQue.

(a) – Order in attributes is same as they appear in queries.

	CPUSpeed	CPUFree	MemSize	MemFree	DiskFree	1MinLoad
CPUFree	0.062					
MemSize	0.052	0.054				
MemFree	0.044	0.064	0.038			
DiskFree	0.061	0.098	0.058	0.084		
1MinLoad	0.008	0.008	0.009	0.004	0.036	
TxRate	0.057	0.087	0.054	0.056	0.085	0.007

(b) – Attributes in queries are sorted ascendingly.

	CPUSpeed	CPUFree	MemSize	MemFree	DiskFree	1MinLoad
CPUFree	0.063					
MemSize	0.043	0.047				
MemFree	0.055	0.083	0.053			
DiskFree	0.059	0.097	0.060	0.102		
1MinLoad	0	0	0	0.003	0.027	
TxRate	0.038	0.064	0.041	0.058	0.071	0.010

(c) – Attributes in queries are sorted descendingly.

	CPUSpeed	CPUFree	MemSize	MemFree	DiskFree	1MinLoad
CPUFree	0.063					
MemSize	0.043	0.046				
MemFree	0.055	0.082	0.051			
DiskFree	0.059	0.097	0.059	0.101		
1MinLoad	0	0	0	0.004	0.027	
TxRate	0.036	0.063	0.039	0.056	0.070	0.010

(d) – Attributes in queries are shuffled randomly.

	CPUSpeed	CPUFree	MemSize	MemFree	DiskFree	1MinLoad
CPUFree	0.063					
MemSize	0.052	0.053				
MemFree	0.061	0.082	0.055			
DiskFree	0.061	0.096	0.060	0.102		
1MinLoad	0	0	0	0.017	0.021	
TxRate	0.058	0.089	0.049	0.072	0.084	0

of resources and multi-attribute range queries while preserving the statistical properties of real-world systems. In [Ba12a, Ba12f] we use the data from ResQue to compare the performance of existing P2P-based resource discovery solutions. Data are also used in [Ba12c] to evaluate the performance of a novel resource and query aware P2P-based resource discovery system and its adaptability.

5.7 Summary

A set of techniques is presented to generate random vectors of static attributes, multivariate time series of dynamic attributes, and multi-attribute range queries while preserving the statistical properties observed in operational systems. A tool is developed to automate the synthetic resource and query generation process and its output is validated using statistical tests. As the proposed mechanism is independent of the dataset, data from any other platform may be used as the basis for trace statistics. Resources and queries generated using the tool are useful in P2P, grid, and cloud computing for evaluating the scalability of applications, resource discovery solutions, and job schedulers, far beyond that is possible with existing test beds.

Chapter 6

RESOURCE AND QUERY AWARE, PEER-TO-PEER-BASED MULTI-ATTRIBUTE RESOURCE DISCOVERY

Distributed, multi-attribute Resource Discovery (RD) is a fundamental requirement in collaborative P2P systems, grid computing, and cloud computing. We present an efficient and load balanced, P2P-based multi-attribute RD solution that consists of five heuristics, which can be executed independently and distributedly. The first heuristic tries to reduce the cost of RD by maintaining a minimum number of nodes in the overlay while pruning nodes that do not significantly contribute to the range query resolution. Nodes deploying the second and third heuristics dynamically balance the key and query load distributions by transferring some of the keys to their neighbors or by adding new neighbors to handle the transferred keys when existing neighbors are insufficient. The last two heuristics, namely fragmentation and replication, form cliques of nodes to dynamically balance the skewed key and query loads associated with highly popular keys/resources. By applying these heuristics in the presented order, a RD solution that better responds to real-world resource and query characteristics is developed. Our solution overcomes several limitations in existing RD solutions, and its efficacy is demonstrated under a variety of real world, single and multi attribute resource and query distributions.

Section 6.1 presents the introduction and motivation. The problem statement is presented in Section 6.2. Five heuristics and their application to single-attribute resources and queries are presented in Section 6.3. How the heuristics are extended to support multi-attribute resources is discussed in Section 6.4. Simulation setup and performance analysis are presented in Sections 6.5 and 6.6, respectively. Section 6.7 presents the concluding remarks.

6.1 Introduction

Collaborative P2P systems require the ability to discover and aggregate group(s) of heterogeneous, distributed, and dynamic resources as and when needed. P2P-based distributed RD is a natural fit for collaborative applications and further enhances their scalability and robustness. P2P-based RD has also been proposed for conventional applications such as grid, desktop grid, and cloud computing, as timely aggregation of complex resources is becoming increasingly necessary due to the proliferation of parallel applications that utilize multiple and distributed resources.

Many P2P-based solutions have been proposed to discover multi-attribute, dynamic, and distributed resources [Al08, Bh04, Ca04, Co09b, Sh07, Sh09]. However, compared to single-attribute P2P systems such as file sharing, formal characterization of real world, multi-attribute resources and queries received attention only recently [Ba11e, Ba12f, He12]. In the absence of data and understanding of the characteristics, designs and evaluations of existing RD solutions have relied on many simplifying assumptions such as independent and identically distributed (i.i.d.) attributes, large domains for attribute values (i.e., number of distinct attribute values $D \gg$ number of nodes N), uniform or Zipf's distributions of all the resources/queries, and queries with a large number of attributes and a small range of attribute values. However, as observed in Chapter 4, the characteristics of real-world systems diverge drastically with attributes of resources being correlated and characterized by different marginal distributions, resources and queries being highly skewed, domains of most attributes being much smaller ($D \ll N$), and queries tending to request a small number of attributes and large ranges of attribute values. Analysis in Chapter 4 also shows that existing solutions have a high resource advertise and query cost (approximating $O(N)$) as attribute values change frequently and queries are less specific. Moreover, they are prone to significant load balancing issues because $D \ll N$, as well as resources and queries are highly skewed. While many solutions are proposed to balance the key and query load in P2P systems [Al08, Bh04, Ga04b, Go04, St03], they also rely on the aforementioned assumptions. Such assumptions affect both the designs

and performance analysis, and consequently the applicability of solutions under real workloads. Therefore, more efficient and load balanced, RD solutions are needed to support real workloads.

We present an efficient and load balanced, resource and query aware multi-attribute RD solution. The solution is based on five heuristics that can be executed independently and distributedly on a ring-like overlay. Ring-like overlay is selected as it turns out to be a relatively more efficient and scalable design choice compared to other solutions (see Sections 4.5.3 and 4.7). The first heuristic tries to maintain only a small subset of the nodes in the overlay as $D \ll N$. It prunes nodes that do not significantly contribute to the range query resolution while reducing the cost (e.g., hops and latency) of resolving queries. The second and third heuristics dynamically balance the key and query load distributions of nodes by transferring part of the keys to their neighbors and by adding new neighbors to handle the transferred keys when existing neighbors are insufficient, respectively. The last two heuristic, namely fragmentation and replication, form cliques of nodes to dynamically balance the skewed key and query loads associated with highly popular resources. In contrast to the common practice of replicating along the overlay ring, cliques of fragments and replicas are placed orthogonal to the overlay ring thereby maintaining lower query cost and better load distribution. By applying these heuristics in the presented order, a RD solution that better responds to the complex characteristics of real-world resources and queries is developed. Our key contributions are the development of a novel heuristic to prune nodes that do not significantly contribute to the range query resolution, placing replicas and fragments orthogonal to the overlay ring, and extending other heuristics to support real workloads while overcoming their deficiencies. While heuristic two is presented in [Ko11] and three is presented in [Ko11, Vu09], we utilize them more efficiently in our solution while being aware of the capacities of nodes and eliminating the need to collect distributed statistics. The fourth and fifth heuristics are introduced in [Ga04b]. However, the solution presented in [Ga04b] is applicable only for relatively stable networks with immutable resources, as the proposed hash function dynamically changes with the load and it needs to be explicitly informed to all the nodes in the system. Our implementation in contrast supports dynamic networks with mutable resources while using a fixed hash function.

Simulation-based analysis is used to evaluate the efficacy of the proposed solution under a variety of single and multi-attribute resource and query distributions derived from real workloads.

6.2 Problem Formulation

We focus on Distributed Hash Table (DHT) based RD solutions due to their scalability and some guarantees on performance [Gu03]. Analysis in Chapter 4 showed that a DHT built on top of a ring-like overlay is relatively efficient and scalable than other available design choices for multi-attribute RD. Let **R** be the set of resources in the system and **A** be the set of attributes used to characterize those resources. We use bold face symbols to refer to a set and the corresponding italic symbol to refer to its cardinality, e.g., $R = |\mathbf{R}|$. List of symbols is given in Table 6.1. A resource $r \in \mathbf{R}$ is defined as follows (see Section 4.2 for further details):

$$r = \left(a_1 = v_1, a_2 = v_2, \ldots, a_i = v_i\right) \tag{6.1}$$

Each attribute $a_i \in \mathbf{A}$ has a corresponding value $v_i \in \mathbf{D_i}$ that belongs to a given domain $\mathbf{D_i}$. $\mathbf{D_i}$'s are typically bounded, may be continuous or discrete, or correspond to a set of categories or names. A multi-

Table 6.1 – List of symbols.

Symbol	Description
$\mathbf{a_q}$	Set of attributes specified in query q
$\mathbf{A}$	Set of attributes used to characterize resources
$\mathbf{D_i}$	Domain of attribute i
h^i_{Query}	No of hops to required by a query to reach the node indexing lower bound l_i
$\mathbf{I}^i$, I^i_{Cap}	Resource index and index capacity of node i
k_i, k_l, k_u	Key of node i, lower bound l_i, and upper bound u_i
$\mathbf{K^i_{In}}$, $\mathbf{K^i_{Out}}$	Set of keys corresponding to *IN* and *OUT* queries
l_i	Lower bound of i-th attribute specified in a range query
m	Required no of resources specified in a query
N	No of nodes in the overlay
$\mathbf{Q}$	Set of queries issues to the RD system
Q^i_{Cap}	Query capacity of node/resource i
$\mathbf{R}$	Set of resources in the system
u_i	Upper bound of i-th attribute specified in a range query
v_i	Value of i-th attribute

attribute, range query q is defined as follows (see Section 4.2):

$$q = \left(m, a_1 \in [l_1, u_1], a_2 \in [l_2, u_2], \ldots, a_i \in [l_i, u_i]\right) \tag{6.2}$$

where, $m \in \mathbf{Z^+}$ specifies the required number of resources and $a_i \in [l_i, u_i]$ specifies the desired range of attribute values (l_i and u_i are lower and upper bounds, respectively). In practice, attributes in a query may specify a mixture of point ($l_i = u_i$) and range ($l_i < u_i$) of values. Let the set of attributes specified in a query be $\mathbf{a_q}$ ($\mathbf{a_q} \subseteq \mathbf{A}$) and $\mathbf{Q}$ be the set of queries issued within the system.

6.2.1 Load Balancing in Peer-to-Peer Systems

Load on a DHT node can be defined in terms of index size, advertise and query messages received, and overlay messages forwarded. Index size is measured using the number of resources/keys or memory/storage required to store those resources or their contact information. Advertise, query, and forward loads are measured using the number of messages or bandwidth consumed. Existing solutions assume all the nodes should be added to the overlay as it helps to balance the load by each node indexing approximately R/N resources or answering Q/N queries. For example, Chord proposed to balance the index size distribution by having $N \log N$ virtual nodes in the overlay [St03] under the assumption that resources are uniformly distributed and $D_i \gg N$. Godfrey et al. [Go04] extended the concept of virtual nodes to balance the query load by moving virtual nodes from highly loaded physical nodes to lightly loaded ones. These solutions are not suitable for real-world RD as the query cost is $O(N \log N)$, resources and queries are highly skewed, and most of the nodes will not index resources or answer queries as $D_i \ll N$ (Chapter 4). SWORD [Al08] proposed to expand the domain size D_i by appending few random bits to the hash values of v_i such that identical resources will be mapped to different DHT nodes. While this helps to distribute the index size, it does not balance the query load, as queries have to start always from the node corresponding to l_i or its hash value. One may argue that attributes such as bandwidth and disk space have a much larger domain; hence, this problem is unrealistic. However, it is not useful to advertise resources at a very high resolution as it significantly increases the advertising cost and users are not interested in

fine-grained queries (Section 4.4.2). Moreover, as real-world systems tend to oscillate between idle and busy periods [Ba12f, Io10] (Section 4.4.1), and their attribute values are not uniformly distributed through D_i. Therefore, it is not useful to explicitly expand D_i. Key transfer is another approach where an overloaded node transfers part of its index (i.e., (*key*, *value*) pairs) to its neighbor(s) [Ko11, Vu09]. This approach is somewhat effective and has a lower overhead. However, when range queries are less specific series of nodes tends to be overloaded, e.g., 89% of the queries specifying free CPU requested a range of [40%, 100%]. Therefore, a node may not be able to transfer its load without making its neighbors even more overloaded. While wave-like load transfer proposed in [Ko11] is useful in such cases, length of the wave needs to be large as query ranges tend to be large in practice. Longer waves are less desirable due to the increase in overhead, as all the nodes along the wave need to coordinate on transferring keys. Alternatively, when a range of nodes is overloaded, it is proposed to migrate nodes in unloaded regions of the overlay to overloaded ones and then transfer the keys [Ko11, Vu09]. This is possible only if the key space is further divisible and distributed statistics are collected to keep track of lists of loaded and unloaded nodes. In practice, it is possible to have a very large number of identical resources. For example, 99% of the nodes in SETI@home were x86 (Section 4.4.1). Moreover, large datacenters tend to simultaneously deploy or upgrade to identical set of nodes. Similarly, a node may receive a very large number of queries due to skewed distributions and large range of attribute values. For example, 89% of the queries that specified the free CPU range of [40%, 100%] have to start the query resolution at the node responsible for indexing the lower bound. Such large indexes and query loads need to be split across multiple nodes using replication and/or fragmentation. In [Ga04b], it proposed to arrange resource attributes on a logical Range Search Tree (RST) that is mapped to a DHT. Each node in the RST is represented as a load-balancing matrix that is expanded and contracted as the load changes. When the index is too large, the number of columns in the matrix (i.e., fragments) is increased. The number of rows (i.e., replicas) is increased when the query load is too high. Each fragment and replica is mapped to the DHT based on its position in the load-balancing matrix. A dynamic hash function is proposed to determine which fragment or replica to query. However, as the size of the load-balancing matrix changes hash function also changes.

An explicit mechanism is needed to inform theses changes all the nodes. Hence, this solution is more suitable for relatively stable networks with immutable resources. Moreover, locality of attribute values is lost when the RST is mapped to the DHT consequently increasing the query cost to $O(N \log N)$. Therefore, existing load balancing solutions do not work efficiently under real workloads.

6.2.2 Problem Statement

We believe that future RD solutions are likely to apply a fixed or dynamic threshold while advertising resources as it reduces the advertising cost and users do not define very specific queries. Applying such a threshold will lead an unbalanced distribution of index size as $D_i \ll N$. Conversely, this can be used to reduce the query cost, as the number of nodes along the ring does not need to exceed the largest D_i (i.e., $N = \max(D_i)$). Then by adding fragments and replicas orthogonal to the ring (contrary to the common practice of adding along the ring), we can balance the index size and query load without increasing the query cost.

Consider a ring-like overlay with a set of **N** nodes indexing **R** resources. Each resource $r \in \mathbf{R}$ is willing to contribute some index capacity I^r_{Cap} and query capacity Q^r_{Cap}. These capacities are typically determined using several factors such as the computing power, memory, bandwidth, or energy of a resource/node, or amount of resources that a user is willing to contribute to the P2P system. We assume I^r_{Cap} and Q^r_{Cap} are measured in terms of the number of resources and messages, respectively. Our goal is to find a solution that minimizes the query resolution latency on a ring-like overlay while satisfying the node capacity constraints. If we assume that the time required to resolve a query within a DHT node is small compared to the network latency and each overlay link has approximately the same latency, then the problem can be restated as minimizing the number of hops required to resolve queries which is given by (see Section 4.2 for the derivation):

$$C^q_{Query} = \sum_{i \in \mathbf{a_q}} \left(h^i_{Query} + \left\lceil \frac{(u_i - l_i)}{D_i} N \right\rceil - 1 \right) \tag{6.3}$$

Users determine ($u_i - l_i$) and $\mathbf{a_q}$, whereas $\mathbf{D_i}$ is fixed. h^i_{Query} depends on N, e.g., for Chord overlay $h^i_{Query} = O(\log N)$. Therefore, the problem further reduces to minimizing N, as it is the only system-level parameter. Our objective is to minimize N subject to the index and query capacity constraints of nodes/resources. More formally:

$$\begin{aligned} &\text{minimize} \quad N \\ &\text{subject to} \quad I^r_{Cap}, \quad Q^r_{Cap} \end{aligned} \tag{6.4}$$

6.3 Handling Single-Attribute Resources

Figure 6.1 illustrates three consecutive nodes ($i - 1$, i, $i + 1$) on the overlay ring illustrated in Fig. 4.1. Let k_i be the key of i-th node. Histograms indicate the keys that are indexed at the node and their heights represent the number of identical resources mapped to that key. Let $\mathbf{I}^i$ be the set of resources indexed at i. Five range queries q_1 to q_5 are indicated as scatted lines. For example, q_1 starts at node i and ends at $i + 1$. While q_4 starts at $i - 1$ and suppose to end at $i + 1$, it terminates at i as the required number of resources are found. As a query q moves from one node to another it appends matching resources to the query. Let k_l represents the key generated by applying a Locality Preserving Hash (LPH) [Ca04] function to the lower bound l_i of a range query. Similarly, let k_u be the hash value of the upper bound u_i. Query resolution starts at the successor node of k_l (see Fig. 4.1). For example, q_1 and q_2 can be considered as coming directly into node i from the overlay network. Set of such queries $\mathbf{Q^i_{In}}$ is defined as *IN queries* where $q \in \mathbf{Q^i_{In}}$ when $k_l \in (k_{i-1}, k_i]$. A query that is answered or reache k_u goes out of the node and the answer is sent to the query originator. For example, q_2, q_3, and q_4 go out from i. Set of such queries $\mathbf{Q^i_{Out}}$ is defined as *OUT queries*, i.e., $q \in \mathbf{Q^i_{Out}}$ when $k_u \in (k_{i-1}, k_i]$ or q is resolved by i. Some queries are forwarded by the predecessor, e.g., q_3 to q_5 are forwarded from node $i - 1$ to i. The set of queries forward from $i - 1$ to i is defined as *forward (FWD) queries* $\mathbf{Q^{i-1}_{Fwd}}$, i.e., $q \in \mathbf{Q^i_{Fwd}}$ when $[k_l, k_u] \cap (k_{i-1}, k_i] \neq \phi$ and $k_l \leq k_{i-1}$. Therefore, the query load on a node is the sum of *IN* and *FWD* queries received within a given

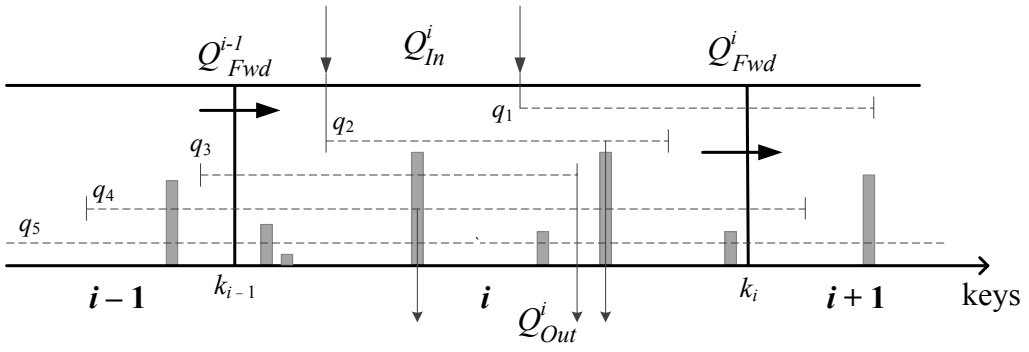


Figure 6.1 – Series of nodes on a ring-like overlay.

time interval *t*. Our objective is to maintain the index size and query load on a node *i* within its bounds, i.e., $I^i \le I^i_{Cap}$ and $Q^i_{In} + Q^{i-1}_{Fwd} \le Q^i_{Cap}$. We first discuss the heuristics in the context of single-attribute range query resolution. Then in the following selection how the heuristics can be extended to support multi-attribute range queries is discussed.

6.3.1 Heuristic 1 – Prune

Consider the query distribution illustrated in Fig. 6.2(a), which is derived from range queries for free CPU and disk space in PlanetLab (see Fig. 4.15). Such a distribution arises when users specify a large $(u_i - l_i)/D_i$ and *m*. Suppose this range is covered by four nodes *a*, *b*, *c*, and *d* and queries start either at *a* or *b* (mostly at *b*) and terminate at *d*. *c* is not answering any queries ($Q^i_{Out} = 0$) and merely forwards them to its successor *d* ($Q^b_{Fwd} = Q^c_{Fwd}$). This occurs when a node does not index any resources or indexed resources are insufficient to answer a given query (either attributes do not match or *m* is too large). It is desirable to remove *c* from the ring as it helps to reduce the number of hops a range query has to travel. If *c* indexes any resources, they have to be moved to *b* or *d* before leaving. A node *i* may pick its successor ($i + 1$) to move keys when $I^i + I^{i+1} \le I^{i+1}_{Cap}$ or may pick the predecessor ($i - 1$) when $I^i + I^{i-1} \le I^{i-1}_{Cap}$. However, removing *c* does not increase the query load on either *b* or *d*. Moving index to the successor is preferred as it reduces the changes to the overlay. For example, *d*'s key will not change when *c* moves its

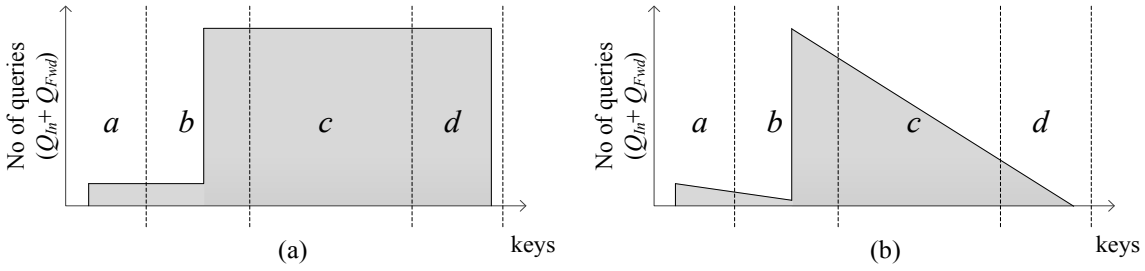


Figure 6.2 – Two example range-query distributions. Scattered lines indicate the range of keys handled by nodes.

index while *b*'s key need to be changed when *c*'s index is moved to *b*. Moreover, following lemma also shows that the query bandwidth requirement of the successor reduces when the predecessor is removed.

Lemma 6.1. *Successor's bandwidth requirement reduces when the predecessor is removed by moving its index to the successor.*

Proof. Suppose node *c* receives a set of queries $\mathbf{Q}^{\mathbf{b}}_{\mathbf{Fwd}}$ from *b* (Fig. 6.2(a)). Let the size of those queries be s_b (in bytes). Suppose *c* appends several matching resources to the queries though it does not completely answer them ($\mathbf{Q}^{\mathbf{c}}_{\mathbf{Out}} = \phi$). Let the number of bytes required to append those resources or their contact information to $\mathbf{Q}^{\mathbf{b}}_{\mathbf{Fwd}}$ be s_c. Then the total number of bytes transferred to *d* is $s_b + s_c$. Similarly, *d* appends a set of resources to the queries with size s_d. Then the total bandwidth requirement of *d* due to the queries that arrives from *c* and queries that leave *d* is $(s_b + s_c) + (s_b + s_c + s_d) = 2s_b + 2s_c + s_d$. When the predecessor *c* leaves the network, *d* will directly receive the set of queries from *b* with size s_b. As *d* now has *c*'s index, it will append the same set of resources. Hence, the size of queries that leave *d* is $s_b + s_c + s_d$. Therefore, the total bandwidth requirement of *d* is $s_b + (s_b + s_c + s_d) = 2s_b + s_c + s_d$. Thus, the bandwidth saving compared to having *c* is s_c. Using a similar argument, it can be also shown that the bandwidth requirement of the predecessor increases when the successor's index is transferred to it. Hence, it is more efficient to move the index to the successor. □

Let us now consider the query distribution in Fig 6.2(b). Such a distribution arises when intermediate nodes are able to answer some of the queries completely (e.g., q_2 and q_4 in Fig. 6.1) or k_u is reached

(e.g., q_3). It is still useful to remove nodes that do not answer many queries to reduce the query cost further. For example, nodes *a*, *b*, and *d* are good candidates as they do not answer many queries. Hence, we remove a node from the ring when the number of *OUT* queries is below a given threshold Q^i_{Thr} (i.e., when $Q^i_{Out} < Q^i_{Thr}$). However, we now need to be aware of both the index size and query load transferred to a node's successor/predecessor to prevent it from being overloaded. For example, if node *i* is removed, its successor $i + 1$ will receive three additional queries q_2, q_3, and q_4 which belongs to $\mathbf{Q^i_{Out}}$ .Thus, $i + 1$ can handle *i*-th node's load only if $I^i + I^{i+1} \le I^{i+1}_{Cap}$ and $Q^{i+1}_{In} + Q^i_{Fwd} + Q^i_{Out} \le Q^{i+1}_{Cap}$. If the index is moved to the predecessor, $i - 1$ will receive two additional queries q_1 and q_2 as its key will change to k_i. Therefore, keys can be moved to $i - 1$ only when $I^{i-1} + I^i \le I^{i-1}_{Cap}$ and $Q^{i-2}_{Fwd} + Q^{i-1}_{In} + Q^i_{In} \le Q^{i-1}_{Cap}$. By keeping track of Q^i_{Out}, a node can decide by itself whether it is not contributing to the system by answering a sufficient number of queries. However, if it indexes any resources or answer any queries, it needs to check with the predecessor/successor before leaving. A node has to continue to remain in the ring, if both the successor and predecessor are not willing to accept its index and/or query load. More nodes can be removed by tightening the threshold Q^i_{Thr}. However, if it is too tight, nodes may need to be frequently removed and then added later using heuristics three to five when the system load fluctuates. When a node is removed from the ring, it will connect to one of the nodes in the ring and use it as a proxy to issue queries and advertise resources (similar to that in a superpeer-based P2P system).

6.3.2 Heuristic 2 – Key Transfer

Suppose *c* in Fig. 6.2(b) is overloaded, i.e., $I^c \ge I^c_{Cap}$ and/or $Q^c_{In} + Q^b_{Fwd} > Q^c_{Cap}$. *c* can reduce the load by moving some of its keys to *b* or *d* (*d* is preferred as it requires minimum changes to the overlay and reduces the bandwidth requirement of *d*). For example, one of the queries at *i* can be reduced if key k_i (Fig. 6.1) is moved before the start of query q_1. Two queries can be reduced, if it is moved even further towards k_{i-1}. Similarly, by reducing k_i index size can be also reduced. Let excess query load at node *i* be

$Q^i_{Excess} = Q^{i-1}_{Fwd} + Q^i_{In} - Q^i_{Cap}$ and index size be $I^i_{Excess} = I^i - I^i_{Cap}$. Only Q^i_{In} can be reduced as Q^{i-1}_{Fwd} is fixed and dependent on the key range of the predecessor. Therefore, it will be useful for a node to transfer its query load to its success only when $Q^i_{In} \geq Q^i_{Excess}$. To decide which keys to transfer to $i + 1$, we need to keep track of k_l (corresponds to the lower bound l) specified in each query q. Let $\mathbf{K^i_{In}}$ be the set of k_ls collected from each $q \in \mathbf{Q^i_{In}}$ (same key may appear multiple times). Then we can find the largest key $k \in \mathbf{K^i_{In}}$ such that the reduced query load $Q^i_{Reduce} = countIF(\mathbf{K^i_{In}}, "\geq k") \geq Q^i_{Excess}$. ***countIF*** is a function that counts the number of keys in $\mathbf{K^i_{In}}$ that satisfies the given condition, e.g., "$\geq k$". Similarly, index size reduced at i should satisfy $I^i_{Reduce} = countIF(\mathbf{I^i}, "\geq k") \geq I^i_{Excess}$. When k_i is moved towards k_{i-1} some of the queries that are being currently answered by i will be forwarded to $i + 1$. For example, when k_i is moved up to k_l of q_1, q_2 and q_3 will be forwarded to $i + 1$. Thus, the successor's query load will be increased by $Q^{i+1}_{Transfer} = countIF(\mathbf{K^i_{Out}}, "\geq k")$. Where $\mathbf{K^i_{Out}}$ is the set of keys that queries terminate at (these include k_us collected from each $q \in \mathbf{K^i_{Out}}$ or the keys that queries like q_2 terminated). The index size of $i + 1$ will also increase by I^i_{Reduce}. Therefore, a subset of the keys can be transferred to the successor $i + 1$ only when $Q^i_{Fwd} + Q^{i+1}_{In} + Q^{i+1}_{Transfer} \leq Q^{i+1}_{Cap}$ and $I^{i+1} + I^{i+1}_{Transfer} \leq I^{i+1}_{Cap}$. If successful, after transferring the keys i-th node's key is set to $k_i = k - 1$.

If the successor is unable to accept the load, the predecessor can be tried. However, the process is reversed where load reduced on i is determined by $\mathbf{Q^i_{Out}}$ and load transferred to $i - 1$ is determined by $\mathbf{Q^i_{In}}$. To decide which keys to transfer, we need to find the smallest key $k \in \mathbf{K^i_{Out}}$ such that $Q^n_{Reduce} = countIF(\mathbf{K^i_{Out}}, "\leq k") \geq Q^i_{Excess}$. Therefore, the transfer is useful only if $Q^i_{Out} \geq Q^i_{Excess}$. Index size reduction at i should also satisfy $I^i_{Reduce} = countIF(\mathbf{I^i}, "\leq k") \geq I^i_{Excess}$. Transferring keys will increase the query load on $i - 1$ by $Q^{i-1}_{Transfer} = countIF(\mathbf{K^i_{In}}, "\leq k")$ and the index size will be increased by I^i_{Reduce}. Before

transferring the keys, node i should also check whether $i - 1$ can handle the transferred loads. If the transfer is successful, predecessor will be pulled towards k_i and its new key is set to $k_{i-1} = k$.

6.3.3 Heuristic 3 – Add New Node and Key Transfer

In range query systems, it is possible for a range of nodes to be overloaded. Therefore, transferring keys to both the predecessor and successor may not be possible. Given that a node is aware of *IN*, *OUT*, and *FWD* loads, it can determine whether it would be useful to add a new node between its current successor or predecessor. Adding a successor is possible and useful when $Q_{In}^{i} \geq Q_{Excess}^{i}$ and there is enough key space between $i - 1$ and i (i.e., $k - k_{i-1} > 1$, where k is determined from heuristic two). Query load transferred to the new successor is $Q_{Transfer}^{new} = countIF(\mathbf{K_{Out}^{i}}, '' \geq k'') + Q_{Fwd}^{i}$ and its index size is $I_{Transfer}^{new} = countIF(\mathbf{I^{i}}, '' \geq k'')$. New successor's key $k_{new} = k_i$ and current node's new key need to be changed to $k - 1$. Similarly, a predecessor can be added when $Q_{Out}^{i} \geq Q_{Excess}^{i}$ and $k < k_i$. Query load transferred to the new predecessor is $Q_{Transfer}^{new} = countIF(\mathbf{K_{In}^{i}}, '' \leq k'') + Q_{Fwd}^{i-1}$ and the index size is $I_{Transfer}^{new} = countIF(\mathbf{I^{i}}, '' \leq k'')$. Key of the new predecessor will be $k_{new} = k$. If the transferred load is too much to be handled by a single node (i.e., $Q_{Transfer}^{new} > Q_{Cap}^{new}$ or $I_{Transfer}^{new} > I_{Cap}^{new}$), a series of successors/predecessors may be added given that there is sufficient key space.

As the first heuristic removes unnecessary nodes, many nodes are not part of the ring. One or more of these nodes can be added as the successor(s) or predecessor(s) when necessary. Nodes that are not in the ring can be found by randomly selecting from the nodes that are connected to a node in the ring, issuing a multi-attribute query to the RD system, or querying a special node that may keep track of those nodes. Therefore, in contrast to [Go04, Ko11, Vu09] our approach does not require an explicit mechanism to track and locate loaded and unloaded nodes in the overlay.

6.3.4 Heuristic 4 – Add New Node and Replicate Index

While the second and third heuristics are effective in distributing some of the load with minor overhead and modifications to the ring, they rely on the assumption that key space is divisible. However, due to skewed resource and query distributions key space is not perfectly divisible, and number of identical copies of a resource or queries for a given range can easily surpass the capacity of even the most resourceful node. Such cases can be detected using $\mathbf{I}^{\mathbf{i}}$, $\mathbf{K}^{\mathbf{i}}_{\mathbf{In}}$, and $\mathbf{K}^{\mathbf{i}}_{\mathbf{Out}}$. Query load can be split across multiple nodes by replicating resources as shown in Fig. 6.3. Such a collection of nodes is called a *clique*. Then a range query needs to visit only one of the replicas along the path consequently splitting the load. To split the load across multiple nodes, predecessor(s) needs to be informed about the existence of multiple successors, which is allowed in many structured P2P solutions such as Chord, Kademlia, and Pastry. While forwarding queries, predecessor(s) may pick one of the successors using round robin or random load balancing policies. When a resource is advertised, it needs to be informed to all the replicas. In practice, a node can handle relatively large number of queries, as most query messages will fit into a single packet and require a sequential search on the resource index. Therefore, a few replicas will be sufficient to handle the most popular queries. A clique may be fully connected to reduce the cost of replication (Fig. 6.3). In contrast to [Ga04b], placing replicas orthogonal to the ring does not require changing the hash function and informing it to all the nodes in the system.

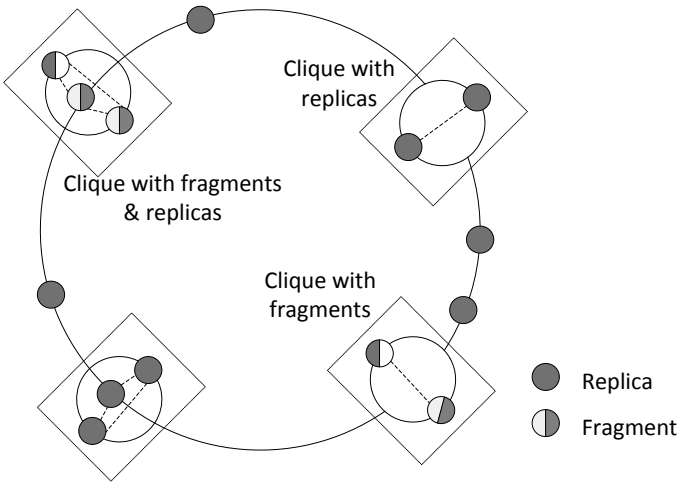


Figure 6.3 – Fragments and replicas placed orthogonal to the overlay ring.

6.3.5 Heuristic 5 – Add New Node and Fragment Index

Heuristic four can be applied only if the number of identical resources is within a node's index capacity. If the number of identical resources is very large, resource index needs to be fragmented across multiple nodes, where each node keeps track of only a subset of the resources. Similar to replicas, fragments are also placed orthogonal to the ring (Fig. 6.3). However, if resources in one fragment are insufficient to resolve a query, other fragments need to be searched before forwarding the query to the successor. Most queries are unlikely to be forwarded to the other fragments, as resource indexes tend to be relatively large in practice. For example, over 10,000 resources each with 20 attributes can be indexed using 1 MB of memory (assuming four bytes per each attribute value v_i).

Heuristics are triggered based on the local statistics collected by a node hence have a lower overhead, and can be executed independently and distributedly. However, it is desirable to deploy all the heuristics within a node as each heuristic addresses a specific concern. Moreover, by applying them in the presented order, an efficient and scalable RD solution can be developed. For example, heuristic one tries to maintain a minimum number of nodes in the overlay while reducing the cost of resolving range queries. A node may have a large index and/or query load regardless of whether it is answering more than Q^i_{Thr} queries. Hence, it is desirable to evaluate rest of the heuristics when a node has to remain in the ring, as its neighbors are not willing to accept the load. The second heuristic tries to balance the load by moving the keys while introducing minimum disruption to the ring. The third heuristic is useful when nodes on the ring are not sufficient to handle the load. However, there is some cost in adding a new node to the overlay as the topology needs to be updated. Fragmentation and replication handle cases of extreme loads but introduce even more changes to the overlay. Thus, by applying the heuristics in the presented order query performance can be improved while reducing the cost of overlay maintenance and key movement. Figure 6.4 illustrates the flow diagram of a node that combines all the heuristics. Histograms can be used to keep track of $\mathbf{I^i}$, $\mathbf{K^i_{In}}$, and $\mathbf{K^i_{Out}}$. Histograms consume only a small amount of memory, as the expected number of distinct attribute values D_i is relatively small. Q^i_{In} and Q^i_{Out} may be calculated from the histograms

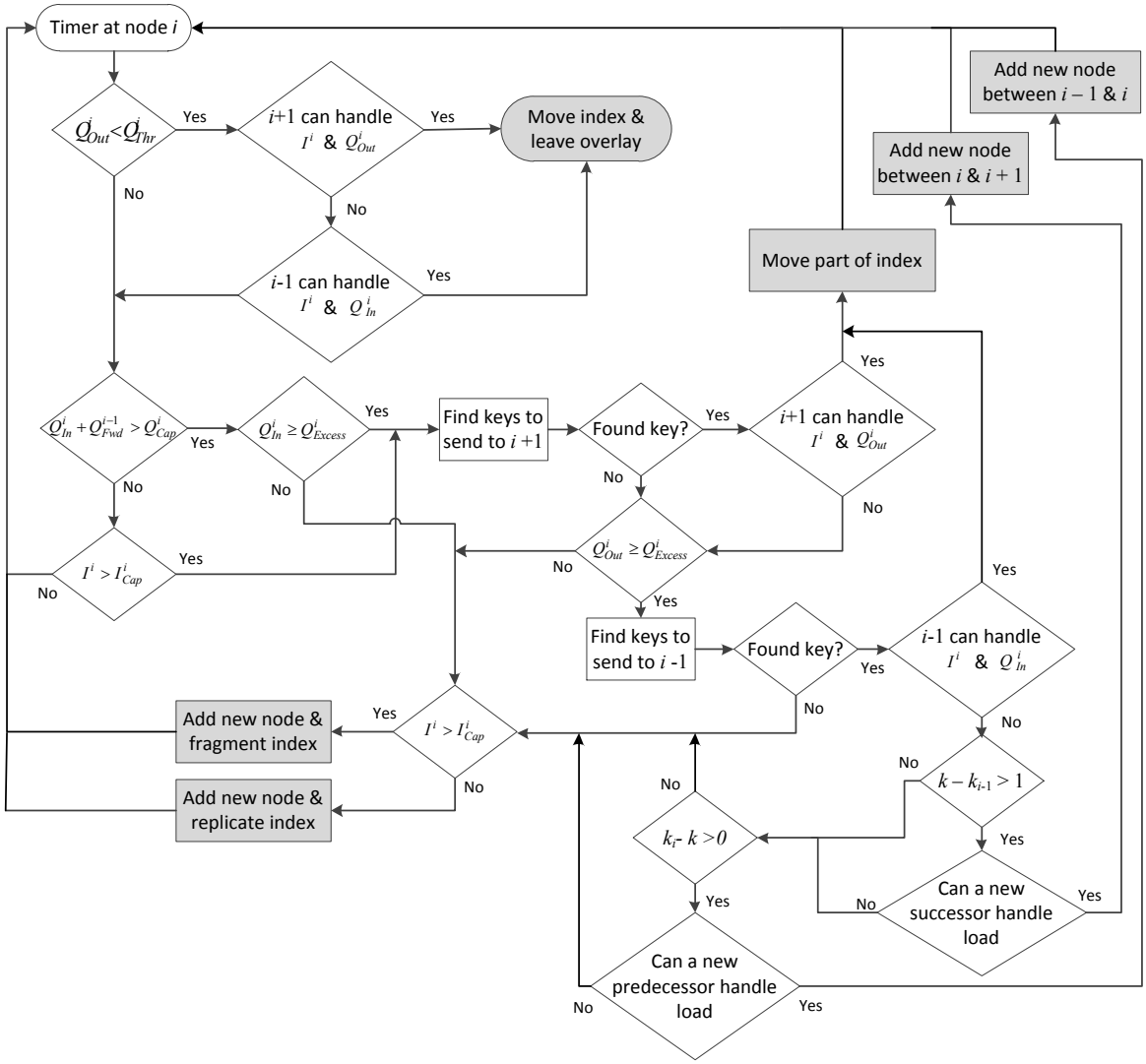


Figure 6.4 – Flowdiagram of a node that implements all five heuristics.

or separate counters may be used. Another two counters are required to keep track of Q^{i-1}_{Fwd} and Q^i_{Fwd}. Therefore, heuristics are triggered based on the local statistics and only the overloaded nodes communicate with their neighbors. Heuristics may be executed periodically or when a counter reaches the capacity of a node. A clique may include both fragments and replicas (Fig. 6.3). If the existence of the fragments and replicas are informed to predecessors, Q^i_{In} can be equally distributed across nodes in a clique. Therefore, notification messages can be sent to all the potential predecessors similar to that in Chord. However, in practice, only the close by predecessors need to be notified as they forward most of the overlay messages [Ba12e]. We do not anticipate a large increase in overlay routing entries, as cliques are small.

6.4 Handling Multi-Attribute Resources

Five heuristics are directly applicable when multiple rings (e.g., Mercury [Bh04]) or a partitioned ring (e.g., LORM [Sh07] and SWORD [Al08]) are used to index different attributes. Multiple-ring-based solutions maintain a separate overlay ring for each attribute type (see Section 2.3.2) where as partitioned-ring-based solutions assign different segments of the address space to different attributes. These solutions maintain a separate resource index for each attribute type similar to single-attribute solutions. Therefore, proposed heuristics are directly applicable. When multiple virtual rings corresponding to different attributes are mapped to the same overlay ring as in MAAN [Ca04] (see Section 2.3.2), a node may have to index the same resource multiple times under different attribute types. In such cases, a resource index may be compressed by removing duplicates entries of the same resource, as it reduces the memory/storage consumption and speed up the query resolution. Therefore, moving a key may not really move an indexed resource as others keys used to index the resource may be still within the range of the node. This problem can be overcome by modifying the ***countIF*** function to take into account the multiple keys used to index the same resource.

6.5 Simulation Setup

A discrete-event simulator is developed to demonstrate the effectiveness of the proposed heuristics. Chord [St03] is used as the underlying overlay, as it supports maintaining multiple fingers to successors. For multi-attribute resources, similar to MAAN [Ca04], we assume multiple virtual rings are mapped to the same address space and queries are issued only to the most selective attribute (i.e., attribute with the smallest $(u_i - l_i)/D_i$). Four single and multi-attribute workloads are derived using real data from P2P file sharing, PlanetLab, and SETI@home and described in Table 6.2. It is known that both the number of queries for a file and copies of a file follow a Zipf's-like distribution [Ha06]. Hence, with the first workload we attempt to demonstrate the applicability of heuristic under skewed resources and point queries. For these workloads, capacities are set as follows: I^i_{Cap} = 500 entries, Q^i_{Cap} = 10 queries/second, and

Table 6.2 – Workloads used in simulations.

Workload	Resources	Queries
File sharing	100,000 copies of 10,000 distinct files generated using ~Zipf's(0.7) [Ha06]	Popularity – Case 1 – ~Zipf's(0.5), Case 2 – ~Zipf's(1.0). Query arrival – 100,000 nodes issue queries with inter-arrival times based on a ~exponential(2 min).
CPU speed	CPU speed of 100,000 nodes randomly sampled from SETI@home. Can be approximated by ~N(2.36, 0.28).	Pulse-like queries derived from PlanetLab query traces. Used empirical CDF to generate ranges of attribute values. Query arrival – 100,000 nodes issue queries with inter-arrival times based on a ~exponential(2 min).
CPU free	Case 1 – Synthetic dataset of 100,000 CPU free values derived using linearly-interpolated empirical CDF of PlanetLab nodes. Case 2 – Case 1 dataset inverted as $x(t) = 100\% - x(t_0)$ at 600 s	Pulse-like queries derived from PlanetLab. Used empirical CDF to generate ranges of attribute values. Query arrival – 100,000 nodes issue queries with inter-arrival times based on a ~exponential(2 min).
PlanetLab	527-node PlanetLab trace with 12 static & 12 dynamic attributes. Also consider 250, 750, 1000 node traces generated using ResQue (Chapter 5).	PlanetLab – Synthetic trace generated using empirical CDFs derived from a_q, popularity of attributes, [l_i, u_i], and m (see Section 4.6). Query arrival – all the nodes issue queries with inter-arrival times based on ~exponential(10 sec).

$Q^i_{\text{Thr}} = 0.1 Q^i_{Cap}$. CPU speed and CPU free workloads assume Q^i_{Cap} = 50 queries/second, as range queries tend to visit many nodes consequently increasing the query load on a node. Such conservative capacities were selected to demonstrate a large enough network under a reasonable simulation time. CPU speed dataset can be approximated by a Gaussian distribution (see Section 4.1). CPU free dataset of PlanetLab nodes is skewed and most nodes were idle (see Fig. 4.6(a)). A node trace from PlanetLab is used as the multi-attribute dataset. As the nodes are described by 24 attributes, in the worst case they may map to 24R nodes. However, this is still smaller than the other three workloads hence for this workload I^i_{Cap} = 100. As the number of nodes is small, we also set Q^i_{Cap} = 25 queries/second. Each simulation is started with an overlay ring having R/I^i_{Cap} nodes, as the network needs to have at least this many nodes to balanced the index size. Predecessors select fragments or replicas using round robin scheduling. Heuristics are evaluated every 30 seconds. To prevent the heuristics from responding to minor variations in index size and query load, Exponentially Weight Moving Average (EWMA) values of counters are used to trigger a heuristic. Results are based on ten samples with different random seeds. Additional details on the simulator are given in Appendix II.3.

6.6 Performance Analysis

We first analyze the single-attribute workloads in detail and then present the results for multi-attribute workload. As our solution with all five heuristics (*Heu 1-5*) will be better than any solution that adds all the nodes to the overlay ring (as N is reduced), we compare our results with a Chord overlay with the same number of nodes. We also compare the results with the second heuristic (*Heu 2*) as it can be readily implemented on top of Chord. Heuristics three to five in their original forms (as in [Ko11, Vu09, Ga04b]) are not directly comparable, as they need specific mechanisms such as special nodes to track loads and dynamic hash functions.

The average query cost of all three solutions under the file sharing workloads was approximately 5.8-hops. When the Zipf's parameter $\alpha = 0.5$ (moderately skewed queries), *Heu 1-5* added 267 nodes to the overlay ring. 304 nodes were added to the ring when $\alpha = 1.0$, as more nodes are needed to handle the increased load due to highly skewed queries and moderately skewed replicas of files. Among the 304 nodes, 257 of them were placed along the overlay ring and the rest were placed orthogonal to the ring. However, no noticeable reduction in query cost was observed for *Hue 1-5*, compared to Chord and *Heu 2*, as the cost of point queries is proportional to log N. Figure 6.5(a) shows the distribution of query load. It can be seen that when all five heuristics are combined, almost all the nodes in the ring were able to stay within the allocated query capacity of 10 queries/second (indicated by the vertical scattered line). While distribution of query load under *Heu 2* is marginally better than having only the Chord ring, one of the nodes still had to handle the query load for the most popular file. When $\alpha = 1.0$, peak load on Chord and *Heu 2* was 77.6 and it was reduced to 28.1 by *Heu 1-5* (2.7 times lower). Similarly, Fig. 6.5(b) shows that *Heu 1-5* were able to maintain the index size of all the node within their capacity where as one of the nodes in Chord and *Heu 2* indexed 3,278 files. Hence, our solution is able to achieve comparable performance for point queries while balancing both the index size and query load.

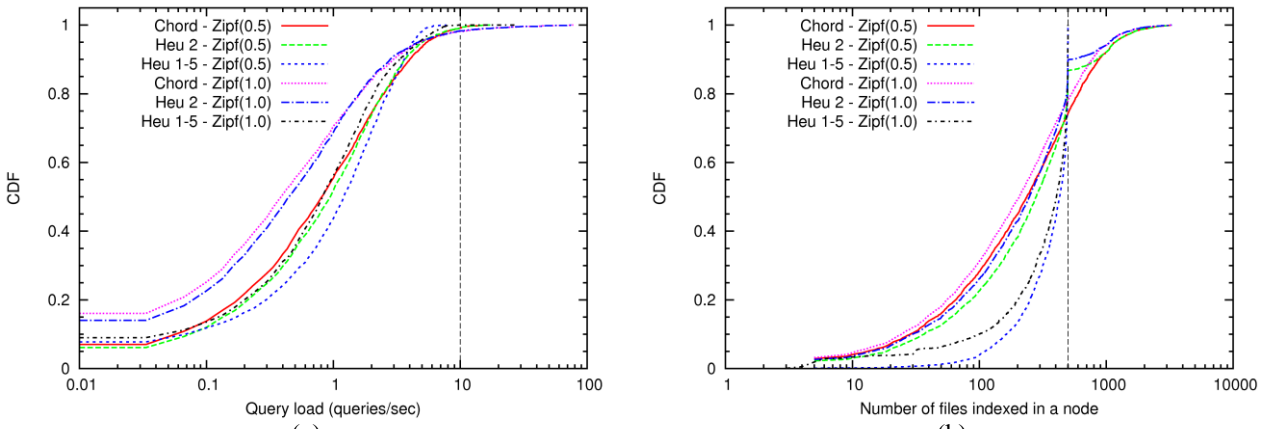


Figure 6.5 – Load distribution of file sharing workloads at steady state: (a) Query load; (b) Index size. Vertical scattered line indicates the node capacity.

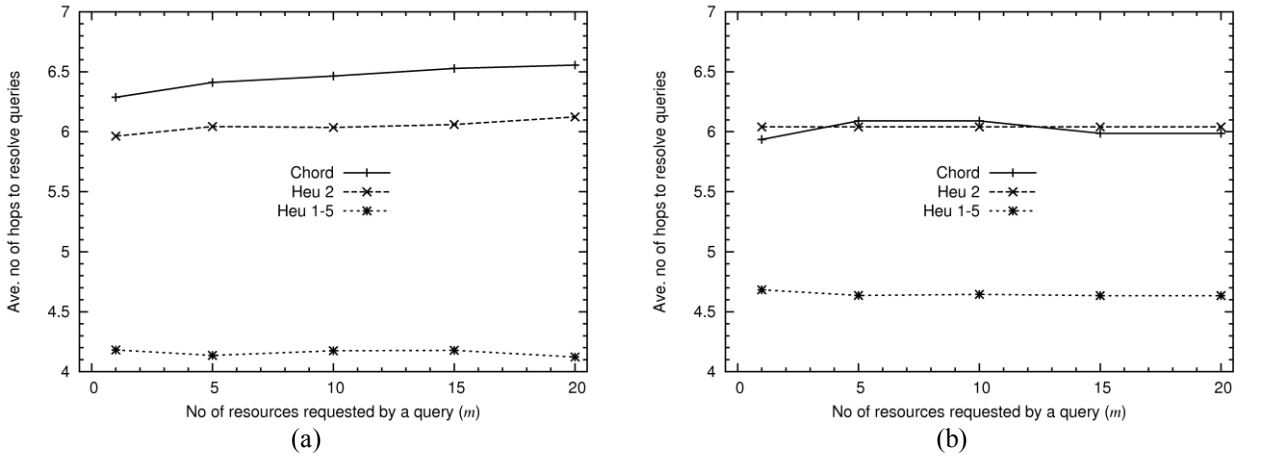
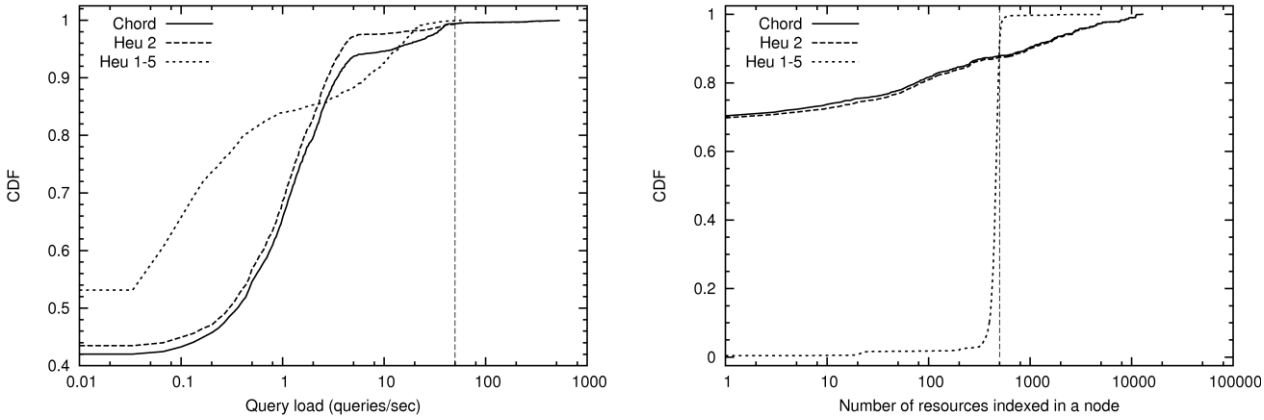


Figure 6.6 – Cost of resolving queries at steady state: (a) CPU speed workload; (b) CPU free workload.

Figure 6.6 compares the query cost under the CPU speed (*CPUSpeed*) and CPU free (*CPUFree*) workloads with increasing m (m is the number of resources requested by a query). When $m = 20$, *Heu 1-5* reduced the query cost of *CPUSpeed* dataset by 37% and *CPUFree* dataset by 23% compared to a Chord overlay with the same number of nodes. Query cost of the *CPUSpeed* dataset increases linearly under Chord and *Heu2*, as the attribute values are spread around following a Gaussian distribution. However, no such increase is observed for *Heu 1-5*, as the placement of nodes are automatically rearranged based on the query loads and size of indexes. Because the *CPUFree* dataset is highly skewed and most nodes were idle, large number of free resources can be found by visiting few nodes. Hence, query cost does not change noticeably with increasing m.

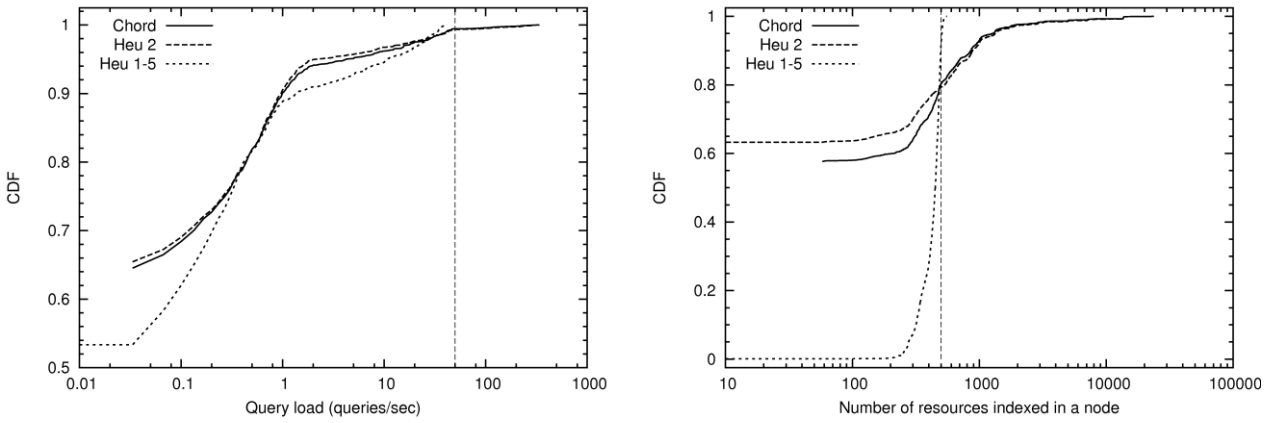
Figure 6.7 shows the query load and index size distribution for the *CPUSpeed* workload. It can be seen that 99% and 91% of the nodes were able to stay within the allocated query and index capacity using *Heu 1-5*, respectively. Largest index under *Heu 1-5* had 556 entries while the other two solutions had 23,733 entries each (42.7 times higher than *Heu 1-5*). Similarly, 92% and 100% of the nodes in the *CPUFree* dataset were able to stay within the allocated index and query capacity (see Fig. 6.8).

Heuristics are triggered when EWMA values of counters exceed the given thresholds. However, the weighting factor β used to calculate the EWMA determines how fast the system gets stabilized and its overhead. We measure the inequality of load distribution among nodes using the Gini coefficient, which has been proposed as a suitable metric to quantify load distribution in P2P systems [Pi06] and many



(a) (b)

Figure 6.7 – Load distribution of CPU speed workload at steady state: (a) Query load; (b) Index size. m = 20.



(a) (b)

Figure 6.8 – Load distribution of CPU free workload at steady state: (a) Query load; (b) Index size. m = 20.

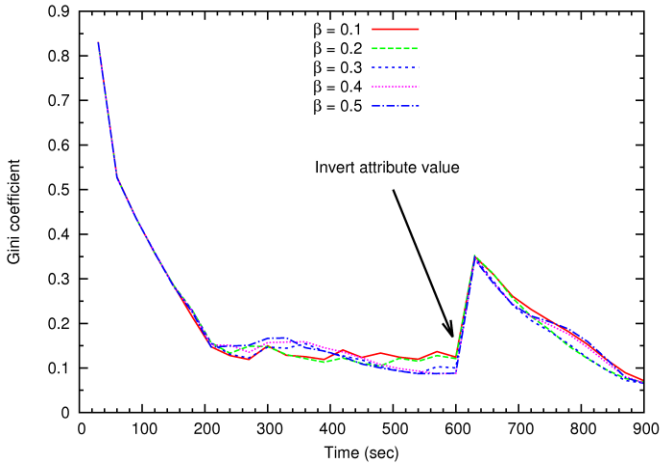


Figure 6.9 – Variation in Gini coefficient of index size distribution of CPU free workload with time. CPU free values were inverted at 600 seconds as explained in Table 6.2. $m = 20$.

networking solutions. Gini coefficient $G \in [0, 1]$, where zero corresponds to perfect equality and one corresponds to the theoretic case of an infinite population with only one individual having a nonzero value. Figure 6.9 illustrates the inequality among index sizes of nodes measured using the Gini coefficient (calculated using the algorithm in [Ro11]). When the weighting factor $\beta = 0.1$ system is biased towards long-term trends hence retain the system in a stabilized state. Large β values quickly respond to short-term trends while constantly moving keys around and modifying the overlay ring. Such frequent changes are not desirable as they increase the cost of load balancing. For example, some oscillations can be seen when $\beta = 0.4$ and $\beta = 0.5$. Moreover, by 570 seconds, 1.6% more messages related to load balancing were generated when $\beta = 0.5$ compared when $\beta = 0.1$. However, while $\beta = 0.1$ quickly reduces the Gini coefficient, with the time Gini coefficient for other β values tend to be even lower. It is known that production systems experience sudden changes in availability of resources [Ba12f, Io10] (also see Section 4.4.1). Therefore, we invert the *CPUFree* value of resources at 600 seconds (as explained in Table 6.2) to measure the responsiveness of heuristics to such rapid changes. Query distribution was not changed, as it is not known whether user queries change in response to such rapid changes in resources. Figure 6.9 shows that system goes back to the original state within ~240 seconds when $\beta = 0.3$. Thus, the five heuristics are also adaptable to rapid changes in attribute values. $\beta = 0.3$ generated 7% less messages related to load balancing

compared to when $\beta = 0.1$ (by 900 seconds). Therefore, we use $\beta = 0.3$ for rest of the performance analysis, as it has a balanced load distribution, lower response time, and lower cost.

We now analyze the multi-attribute workload from PlanetLab, which exhibits the characteristics discussed in Chapter 4. Figure 6.10(a) shows that the average query cost linearly increases with the increasing number of nodes. As the number of nodes in the system increases, both the number of resources to index and queries to answer increase (because the query rate of each node is fixed). Moreover, as the number of attributes increases, the same resource is mapped to many overlay keys hence resources are spread over a large address space. Furthermore, queries for different attributes are issued to different ranges in the address space. For example, queries for *CPUFree* are biased towards the higher value while queries for CPU load are biased towards the lower value. Therefore, more nodes are added to different address ranges in the ring to balance the load. Consequently, query cost increases, as is it proportional to the number of nodes along the ring. Alternatively, though Chord and *Heu 2* have the same number of nodes in the ring, they are uniformly spread around the ring consequently reducing the number of nodes that an average query needs to go through. This is the reason that the cost of resolving multi-attribute resources using *Heu 1-5* is higher than Chord and *Heu 2* with a similar number of nodes. However, worst-case path length of Chord and *Heu 2* is higher than *Heu 1-5* (see Fig. 6.10(b) and Fig. 6.11), as all the nodes are placed along the ring compared to the fourth and fifth heuristics which place some of the nodes

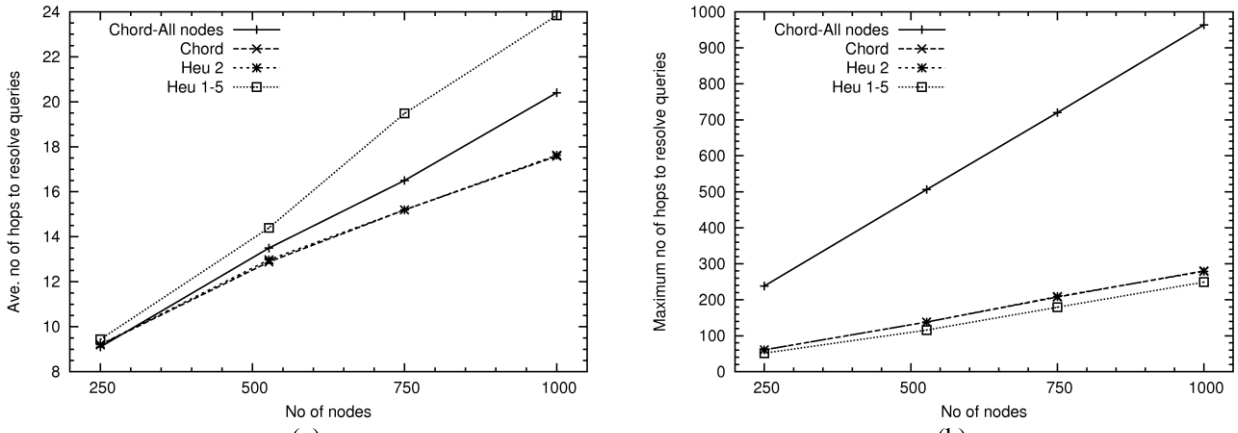


Figure 6.10 – Number of hop required to resolve queries in PlanetLab workload at steady state: (a) Average query cost; (b) Maximum query cost.

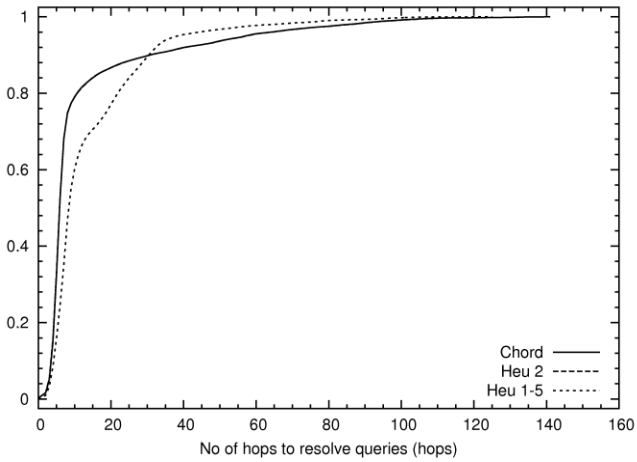


Figure 6.11 – Distribution of query cost in PlanetLab workload at steady state. R = 527.

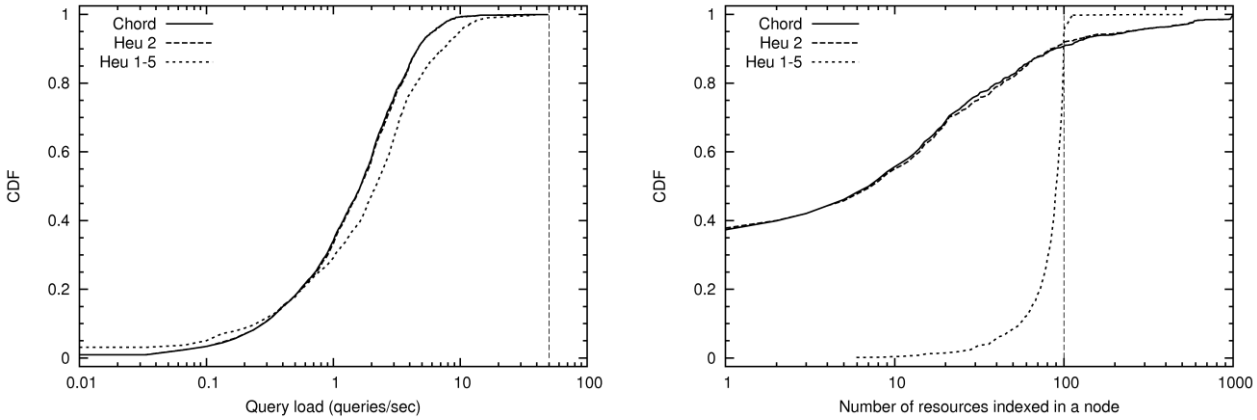


Figure 6.12 – Load distribution of PlanetLab workload at steady state. R = 1,000.

orthogonal to the ring. Distribution of query cost in Fig. 6.11 confirms this behavior. Moreover, both the average and worst-case query cost is still significantly lower than adding all the nodes to the ring (Fig. 6.10) . For e.g., average and worst-case query cost of *Heu 1-5* is 59.5% and 77.5% lower than adding all the 527 nodes to the Chord ring.

Figure 6.12 illustrates that *Heu 1-5* can effectively balance the index size and query load under multiple attributes as well. It can be seen that 100% and 95% of the nodes were able to stay within the allocated query and index capacity using *Heu 1-5*, respectively. One of the nodes indexed 507 resources under *Heu 1-5* while other two solutions indexed all the 1,000 resources. Therefore, proposed heuristics enable discovery of real-world resources with lower overhead while balancing the index size and query

load. Moreover, they rely on local statistics, local communication among members of a clique, predecessors, and successors, as well as do not require dynamic hash functions.

6.7 Summary

Five heuristics for efficient P2P-based multi-attribute RD that alleviates the load-balancing problem were presented. Heuristics rely on local statistics to capture the complex characteristics of real-world resources and queries and try to retain only the nodes that answer a sufficient number of queries in the overlay. Resource index is transferred among existing and new nodes are added to maintain the index size and query load of a node within its capacity. By applying these heuristics in the presented order, a RD solution that better responds to real-workloads was developed. Simulation-based analysis demonstrated their ability to reduce the query cost, balance the load, and adapt to rapid changes in attribute values.

Chapter 7

COMMUNITY-BASED CACHING FOR ENHANCED LOOKUP PERFORMANCE IN P2P SYSTEMS

Large Peer-to-Peer (P2P) systems for file transfer exhibit the presence of virtual communities based on semantic, geographic, or organizational interests of users. Resources commonly shared within individual communities are in general relatively less popular and inconspicuous in the system-wide behavior. Hence, most communities are unable to benefit significantly from performance enhancement schemes such as caching and replication that focus only on the most dominant queries. We propose a distributed Community-Based Caching (CBC) solution that enhances both the communitywide and system-wide lookup performance. CBC consists of a sub-overlay formation scheme and a Local-Knowledge-based Distributed Caching (LKDC) algorithm. Sub-overlays enable communities to forward queries through their members. While queries are forwarded, the LKDC algorithm causes members to identify and cache resources of interests to them, resulting in faster resolution of queries for popular resources within each community. Distributed Local Caching (DLC) requires global information (e.g., hop count and content popularity) that is difficult and costly to obtain. However, by means of an analysis of globally optimal behavior and structural properties of the overlay, we develop the heuristic-based LKDC algorithm that not only relies on purely local information but also provides close-to-optimal caching performance. Simulation-based analysis is used to demonstrate the utility of the analytical model and CBC.

Section 7.1 presents the introduction and contributions. Problem formulation is presented in Section 7.2. Sub-overlay formation and distributed caching requirements are presented in Section 7.3. In Section 7.4, DLC problem, relaxed-DLC problem, and the proposed LKDC algorithm are presented. Simulation setup and performance analysis are presented in Sections 7.5 and 7.6, respectively. Section 7.7 presents the concluding remarks.

7.1 Introduction

P2P systems are continuing to grow, attracting millions of users and expanding into many application domains beyond conventional file sharing. Modern P2P systems share a variety of *resources* such as files, processor cycles, storage capacity, and sensors. Current systems are designed based on either the system-wide behavior, attempting to provide everyone an equal level of service (e.g., average search/download time), or optimized for more dominant users' requirements. In either case, the performance of *lookup* (i.e., the process of searching for resources) degrades as the system sizes continue to grow.

Recent studies [Zh10] show that P2P systems in fact consist of many smaller virtual communities. A *community* is a subset of peers that share some similarity in terms of resource semantics, geography, or organizational boundaries (see Section 2.4). Peers have semantic relationships based on the type of resources they frequently access [Zh10, Ha06]. For example, BitTorrent has many communities dedicated to music, movies, Linux distributions, and games (Section 2.1.1). Users from the same country tend to access similar resources as well. For example, for 60% of the files shared by eDonkey peers, more than 80% of their replicas were located in a single country [Ha06]. Moreover, semantic and geographic similarities are more prominent for moderately popular files. Communities may also arise based on organizational boundaries, e.g., members of a professional organization or a group of universities often forms their own community to share resources and limit unrelated external traffic. CASA (Section 2.2.1) is one such application where diverse communities of end users (e.g., emergency managers, National Weather Service, scientists, media, and transportation agencies) access/share different subsets of data generated by a distributed set of radars. We can further envision distributed collections of large scientific databases such as Genome sequences, Geographic Information Systems (GIS), weather, census, and economic data that are accessed by various communities of users from academic, research, and commercial institutions.

Emerging technological trends such as social networking indicate that we will continue to see the emergence of a large number of small and diverse communities within large P2P systems. Future P2P architectures therefore should support such communities by providing customized services based on their

distinct characteristics. Such architectures should allow the emergence, growth, existence, and disappearance of communities on a continual basis, while enabling them to be a part of a global community or a system. Conversely, the P2P system can significantly benefit by taking into account the characteristics and requirements of these communities.

Content popularity profiles in P2P systems follow a Zipf's-like distribution [Ra04, Ra07, Sr01]. However, resources popularly shared within an individual community typically do not rank high in popularity in the context of the overall P2P system [Ba12e, Ha06] and often are inconspicuous in the system-wide behavior. Therefore, such communities are unable to benefit from performance enhancements such as caching and replication that focus only on the most popular resources. For example, Beehive [Ra04] and PoPCache [Ra07, Ra10] (see Section 2.5.2) force a large fraction of peers (in structured P2P systems) to cache the most popular resources regardless of their interests. In spite of requiring large caches and many probing messages to estimate the global popularity, such solutions are inconsiderate of moderately popular resources. Several caching solutions are also proposed for unstructured P2P systems [Co02, Th04] (see Section 2.5.1). However, due to the random overlay topologies in unstructured P2P systems, even the most popular queries are unable to benefit significantly from caching. Instead, several solutions propose to restructure the overlay topology based on users' interests [Be10, Li09a, Xu10, Ze11]. These solutions provide better performance when a user's interests match those of the overall community. However, community membership is not rigid. A user, for example, may belong to multiple communities or switch from a geography-based community to a semantic-based one. Moreover, a community of researchers analyzing the spread of epidemics may access multiple scientific databases such as Genome sequences, GIS, census, and weather data. Our analysis of search clouds from several BitTorrent communities (discussed in Section 7.2) confirms that user interests in different communities overlap to some degree. As the communities do not exist in isolation, it is desirable to form one large overlay by combining peers from all the communities such that resources can be efficiently accessed across all the communities. However, existing solutions cannot provide optimum performance under shared communities, and they are not designed to build communities based on incomparable similarity measures such as semantics and

geography. Alternatively, mixing of resources from multiple communities is not desirable, as the popularity of individual resources typically subside due to the mixing of many unshared/unrelated resources (Section 7.2). Therefore, it is important to not only maintain all the communities within a single overlay but also cater to their popularities. Moreover, it is necessary to develop proactive caching solutions that are aware of communities' interests, adaptive, as well as message and storage efficient.

We propose a proactive Community-Based Caching (CBC) solution for structured P2P systems where individual communities form seamlessly and cache resources of interest to them while being in a larger overlay. CBC consists of a sub-overlay formation scheme and a Local-Knowledge-based Distributed Caching (LKDC) algorithm. We first propose a method whereby sub-overlays are formed within the overlay network, enabling communities to forward queries through their members. While the queries are forwarded, LKDC algorithm causes the peers running it to identify and cache resources that are popular within their communities. Therefore, lookup queries for popular resources within a community are resolved faster. Consequently, both the community-level and the system-level lookup performance improve. Distributed Local Caching (DLC) requires global information such as hop count and content popularity that are difficult and costly to obtain. However, by analyzing the globally optimal behavior and taking into account the structural properties of the overlay, we show that it is still possible to develop a close-to-optimal caching solution (namely LKDC) that relies purely on local statistics. CBC is independent of how the communities are formed, adaptive to changing popularity and user interests, and works with any skewed distribution of queries. It is more suitable when users primarily access resources from few communities and when the size of a community is moderate to large with respect to the size of the overall P2P system. Furthermore, it introduces minimal modifications and overhead to the overlay network. Compared to Beehive and PoPCache, which utilize large caches and distributed statistics, CBC caches more distinct resources using smaller caches and utilizes only the local statistics. Simulations based on Chord [St03] overlay, for example, show a 40% reduction in overall average path length with per-node cache sizes as low as 20. Less popular communities are able to reduce the path length by three times compared to system-wide caching.

7.2 Problem Formulation

Communities tend to emerge naturally in P2P systems with a large pool of resources and users. For example, as BitTorrent grew, many web-based torrent search engines with specific interests on movies, songs, games, and software emerged in a top-down manner. Most search engines deployed their own trackers leading to islands of BitTorrent deployments (see Fig. 2.3). These isolated search engines are referred to as *BitTorrent communities*. Isolation became a problem, as users with diverse interests had to search in many communities to find peers with better upload capacities. Consequently, BitTorrent protocol version 4.2 enabled content look up across multiple communities using a Distributed Hash Table (DHT). Thus, the current BitTorrent system is a top-down aggregation of diverse communities. However, resource popularity subsides when multiple communities are aggregated. Section 7.2.1 first provides evidence for these characteristics and discusses their implications on lookup performance. Next, the research problem is formulated in Section 7.2.2.

7.2.1 Motivation

As community membership is not rigid, P2P communities do not typically exist in isolation. For example, a user may belong to multiple communities, switch from a geography-based community to a semantic-based one, or belong to a geography-based community in a different country. We analyzed the search clouds from several BitTorrent communities to determine to what extent the communities tend to access the same content. Table 7.1 summarizes the similarities among communities, which were obtained by calculating the Cosine similarity [Sa68] among the lists of file names that appeared in search clouds of eight BitTorrent communities. User interests in most communities overlap to varying degrees. Few communities are independent, e.g., seedpeer.com. Figure 7.1 illustrates the distribution of query popularity for three of the communities. Summary of those three datasets is given in Table 7.2. Ranked popularity distribution can be approximated by a linear function and its gradient determines the Pareto index k. Zipf's parameter α can be estimated from this distribution as $\alpha = 1/k$ [Ad00]. Therefore, Zipf's parameters of the three communities (including two datasets for kat.ph) are 0.53, 0.66, 0.79, and 0.98. This further confirms

that communities have different popularity distributions. Rapid decline in cumulative distribution in Fig. 7.1 (c) and (d) indicates that there are several highly popular queries. The same behavior is also observed when the popularity distribution is analyzed under different time scales (see Fig. 7.2).

Table 7.1 – Cosine similarity among different BitTorrent communities based on their search clouds. Date – 24/07/2010 ~04:55 GMT.

Community*	EX	FE	SP	TB	TS	TE	TR
FE	0.38						
SP	0.00	0.00					
TB	0.40	0.29	0.00				
TS	0.48	0.33	0.00	0.48			
TE	0.53	0.23	0.00	0.31	0.25		
TR	0.10	0.08	0.00	0.06	0.09	0.06	
YB	0.36	0.35	0.00	0.29	0.42	0.20	0.04

* EX – extratorrent.com, FE – fenopy.com, SP – seedpeer.com, TB – torrentbit.net, TS – torrentscan.com, TE – torrentsection.com, TR – torrentreactor.net, YB – youbittorrent.com.

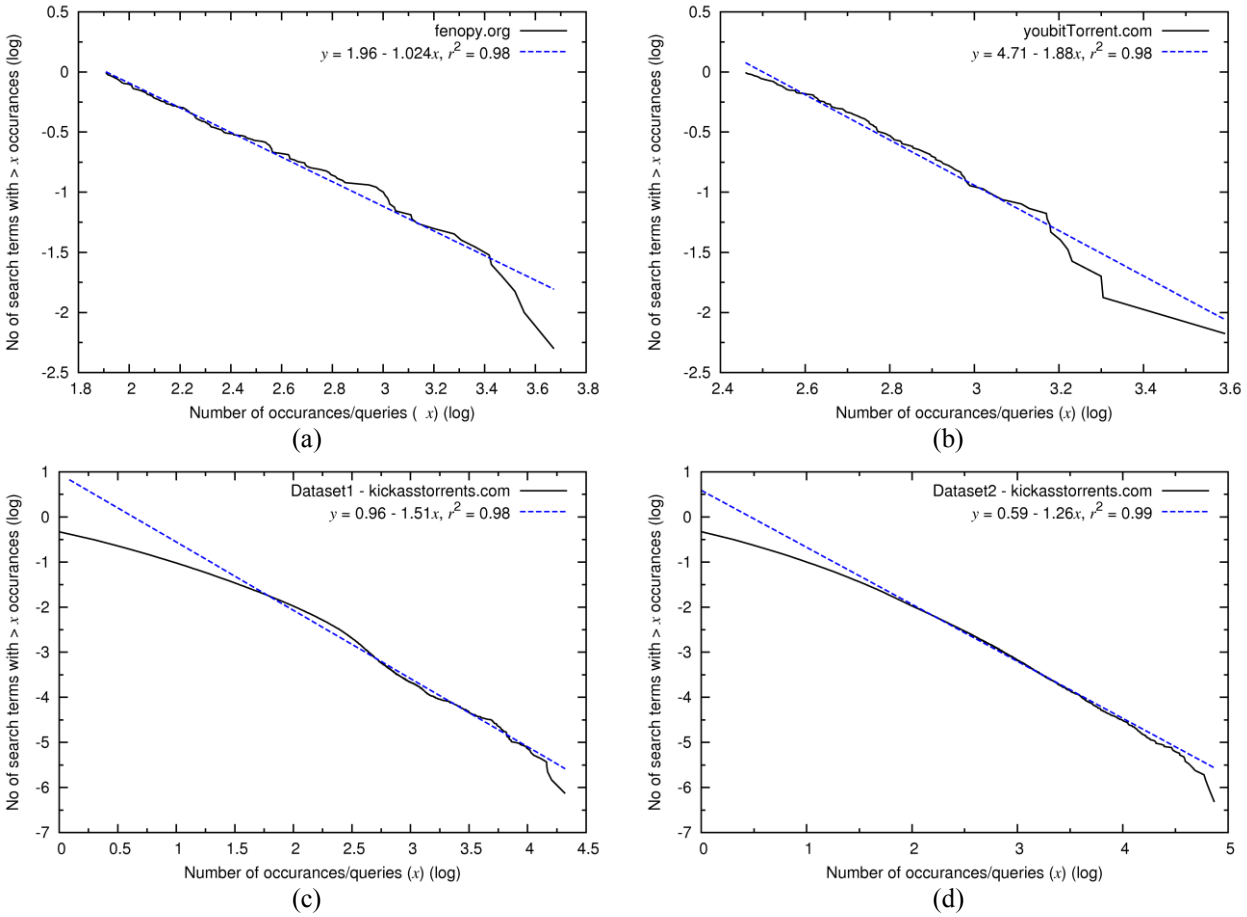


Figure 7.1 – Popularity distribution of BitTorrent communities: (a) fenopy.com. α = 0.976; (b) youbittorrent.com. α = 0.53; (c) Dataset1 – kat.ph. α = 0.66; (d) Dataset2 – kat.ph, α = 0.79.

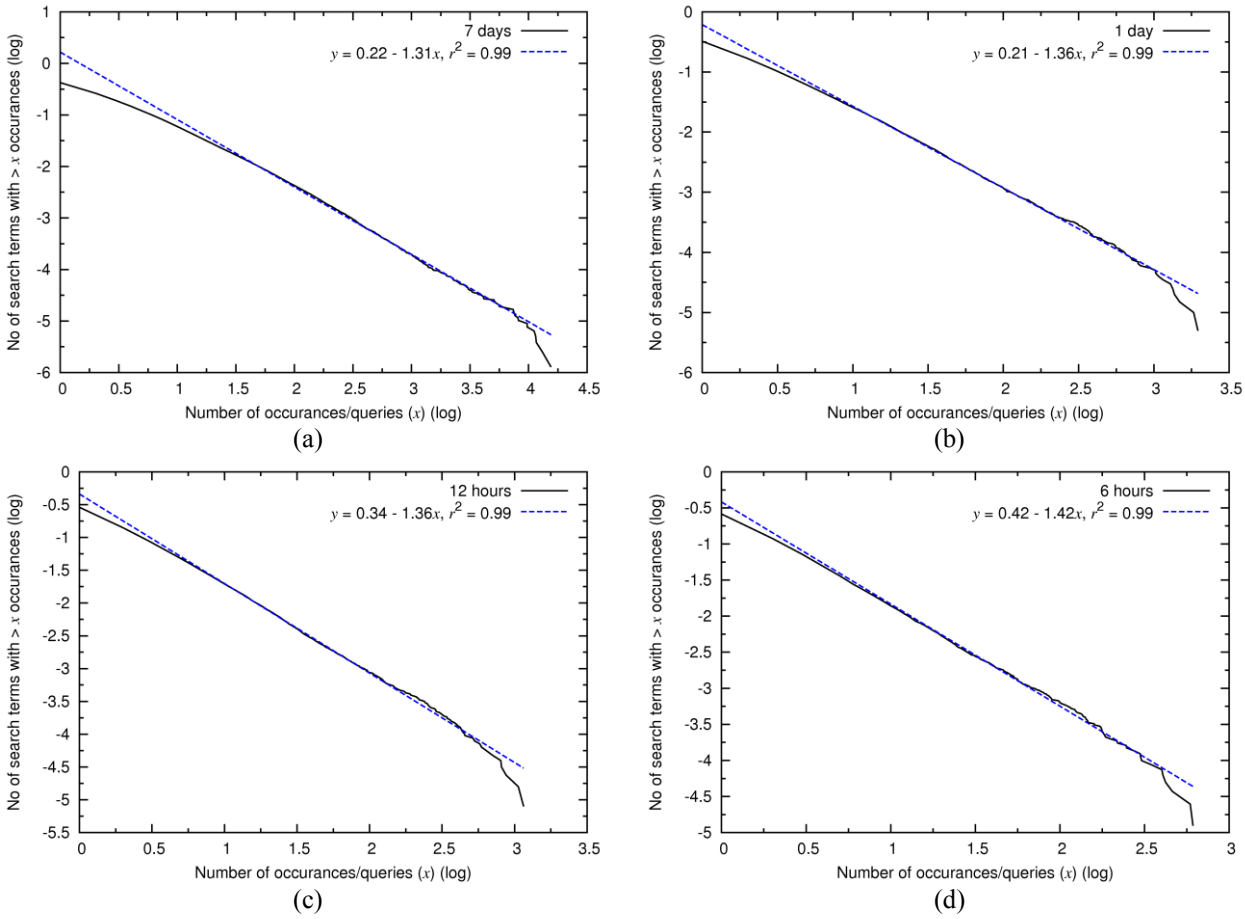


Figure 7.2 – Popularity distribution of Dataset2 (kat.ph) over different time scales starting at 2010/07/06 15:00 UTC: (a) 7 days. $\alpha = 0.76$; (b) 1 day. $\alpha = 0.74$; (c) 12 hours. $\alpha = 0.74$; (d) 6 hours. $\alpha = 0.7$.

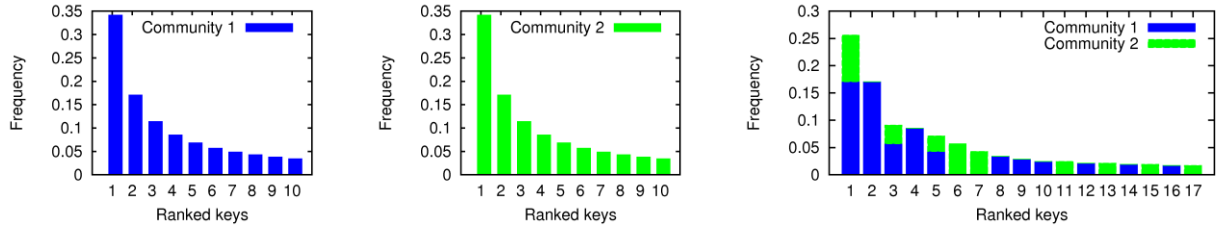
As the communities are not completely isolated, it is necessary and desirable to form one large overlay (by combining peers from all the communities) to efficiently access resources across all the communities. Several P2P solutions (e.g., SWOP [Hu04], SWAN [Li07c], ESLP [Li09b], Gossple [Be10], and Tribler [Ze11]) accommodate communities by restructuring the overlay topology to form clusters of community members. Such clusters provide better lookup performance when users' interests match those of the overall cluster. However, lookup performance degrades when communities are partially isolated (due to inter-cluster lookup queries) where substantial fraction of queries is for resources outside of a particular cluster. For example, Tribler cannot search for contents beyond a node's cluster (i.e., set of directly connected nodes) [Ze11]. A Gossple node cannot search for contents beyond what it has already received/heard from similar neighbors as it utilizes a gossip scheme [Be10]. Therefore, contents

Table 7.2 – Description of BitTorrent search terms datasets.

Dataset	fenopy.com	youbittorrent.com	Dataset1 – kat.ph	Dataset2 – kat.ph
Date	2010/07/24 04:37 UTC.	2010/07/24 04:43 UTC.	2010/06/09 00:00 to 2010/06/27 15:41 UTC	2010/07/06 15:00 to 2010/08/10 09:44 UTC
No of queries	200	150	9,669,034	18,891,335
No of distinct queries	195	150	1,349,976	2,091,699
Zipf's parameter (α)	0.98	0.53	0.66	0.79

are not guaranteed to be found. Though SWOP [Hu04] provides guaranteed content discovery, it utilizes a two-step lookup process within the DHT. Contents are first searched in the local index of cluster members regardless of whether the contents are indexed in them or not. When a look up within the cluster fails, it searches the global index. This two-step process increases the worst-case lookup cost compared to searching only within the cluster. Moreover, these solutions are not designed to build communities based on incomparable similarity measures such as semantics and geography. Whereas, a good lookup solution should provide efficient inter and intra-community lookup and support communities based on different similarity measures.

Resource popularity subsides when multiple communities are aggregated together unless all the communities access the same set of resources. For example, aggregation of two communities with Zipf's-like distributions does not necessarily result in a Zipf's-like distribution unless they have an identical set of resources and popularity distributions. As an example, consider two communities with the same Zipf's parameter $\alpha = 1.0$ (see Fig. 7.3). Suppose resources 1, 3, and 4 in community one are same as the resources 2, 5, and 6 in community two, respectively. Figure 7.3(c) depicts the corresponding aggregated frequency distribution. The frequency of all the resources has reduced and popularity distribution is no longer Zipf's-like as multiple resources have the same frequency. This behavior is more prominent when multiple partially overlapped communities are aggregated. Therefore, even the most popular resources within a community can be inconspicuous in the system-wide behavior. Consequently, communities are unable to benefit significantly from caching solutions such as Beehive [Ra04] and PoPCache [Ra07] (renamed as PCache in [Ra10]), that focus only on the most dominant resources within the entire P2P system. Moreover, both Beehive and PoPCache collect distributed statistics to estimate the global popularity



(a) (b) (c)

Figure 7.3 – Aggregation of popularity distributions: (a) Community 1; (b) Community 2; (c) Aggregated popularity distribution. 10 keys, $\alpha = 1.0$.

of resources. To provide a guaranteed mean path length, an optimization problem is then solved to determine how many cache entries to create and where to place them. For example, Beehive (implemented on top of Chord) places the most popular set of resources on every node in the system, second most popular set of resources on ½ of the nodes, third most popular set on ¼ of the nodes, and so on. Alternatively, PoPCache utilizes the structure of the Overlay Routing Tree (ORT) to place cache entries more efficiently. The number of cache entries allocated to a particular resource is proportional to its global popularity and cache entries are placed along the ORT starting from the *root node* (i.e., node responsible for indexing a given resource). These solutions force a large fraction of nodes to cache the globally popular resources regardless of their individual or community interests. Moreover, global popularity estimation is costly and error prone. Furthermore, the solution obtained by PoPCache is suboptimal as the actual ORT is asymmetric and nodes have limited cache capacity (see Section 7.4.2). In some cases, a resource that is not so popular within individual communities may still become popular in the system-wide behavior if many communities access it. Therefore, it is necessary to develop a proactive caching solution that is aware of communities and their interests, preserves the popularity distribution of individual communities and the overall system, adapts to varying user interests, and message and storage efficient.

7.2.2 Problem Statement

Future P2P architectures need to support a large number of communities while providing services based on their distinct characteristics. However, sharing among P2P communities suggests that communities should not be isolated, and conversely combining multiple communities together subside relative

popularities of contents. Existing solutions are inadequate as they are limited to either isolating communities or combining all the communities together. Alternatively, better lookup performance can be gained by catering to the popularity of individual communities while being members of a larger P2P system.

Consider a P2P system with a set of **M** communities, with community $m \in \mathbf{M}$ consisting of a set of $\mathbf{N}_m$ nodes interested in a set of $\mathbf{K}_m$ resources with normalized popularity f_k, where $k \in \mathbf{K}_m$. Node $n \in \mathbf{N}_m$ has a cache capacity C_n. List of symbols is given in Table 7.3. Our goal is to find a feasible assignment of cache entries to peers that minimizes the average hop count of each community $m \in \mathbf{M}$. The desired distributed solution should support communities based on multiple similarity measures, be adaptive to varying user interests, work with any skewed distribution of queries, introduce minimum modifications to the overlay topology, and be efficient with respect to cache storage and overlay messages exchanged.

Table 7.3 – List of symbols.

Symbol	Description
b	Key length in bits
B	Total cache budget
c_k	Cache capacity allocated to key k
C_n	Cache capacity of node n
C_{ave}	Average cache capacity of a node
f_k	Normalized frequency/popularity of key k
$g(c_k)$	Number of hops reduced by caching c_k entries
H_{ave}/h_{ave}	Average hops in a network with/without caching
h_k^n	Number of hops required to resolve query for key k starting at node n
hop_{max}	Number of hops to forward a community-member-discovery message
k_i	i-th key
K, **K**	Number/set of keys
m_i	i-th community
n_i	i-th node
N, N_m, **N**	Number/set of overlay nodes, N_m – in community m
S_k	Size of key k
T_{cache}	Caching threshold
T_{remove}	Remove threshold for entries in lookup table
v_k^n	Value of caching key k at node n
x_k^n	Whether key k is cached at node n. 1 if cached, otherwise 0.
α	Zipf's parameter
β	Parameter use to approximate $g(c_k)$
θ	Weighting factor for query demand
λ, μ_k	KKT multipliers
λ_k^n	Demand for key k at node n

7.3 Caching Solution for Communities

We focus on structured P2P systems, as they are appropriate for large-scale implementations due to high scalability and some guarantees on performance [Gu03]. Let us discuss a specific example using Chord (Section 2.1.2), which is considered the most flexible and robust structured P2P system [Gu03]. The discussion is applicable in general to other structured P2P systems as well. Chord maps both nodes and resources into a circular key space (see Fig. 7.4(a)) using consistent hashing. However, Chord assumes all nodes to be equal partners and does not support any community formation. A node is assigned to a random location within the ring. Based on the *key*, a resource is indexed at its *successor*, i.e., the closest node in the clockwise direction. Each node n maintains a set of pointers, called *fingers*, to nodes that are at $(n + 2^{i-1})$ *mod* 2^b, where $1 \leq i \leq b$ and b is the key length in bits. For example, node n_E in Fig. 7.4(a) keeps fingers to nodes n_F, n_G, n_H, and n_J. Routing table at a node consists of these fingers and it is called the *finger table*. The fingers are used to recursively forward a message to a given *key* within $O(\log N)$ hops. For example, n_E can reach n_L through the route $n_E \rightarrow n_J \rightarrow n_L$. A node may also identify redundant fingers (to additional nodes) to reduce the latency and enhance robustness, e.g., if n_E knows about n_K, a message may also take the path $n_E \rightarrow n_K \rightarrow n_L$. Nodes can get to know about the demand for different keys by observing the *get(key)* messages that are forwarded through them. Accordingly, they can either cache the resources corresponding to those keys or their location. For example, if some of n_L's contents is cached at n_J, it can respond to a query from n_E to n_L within one hop.

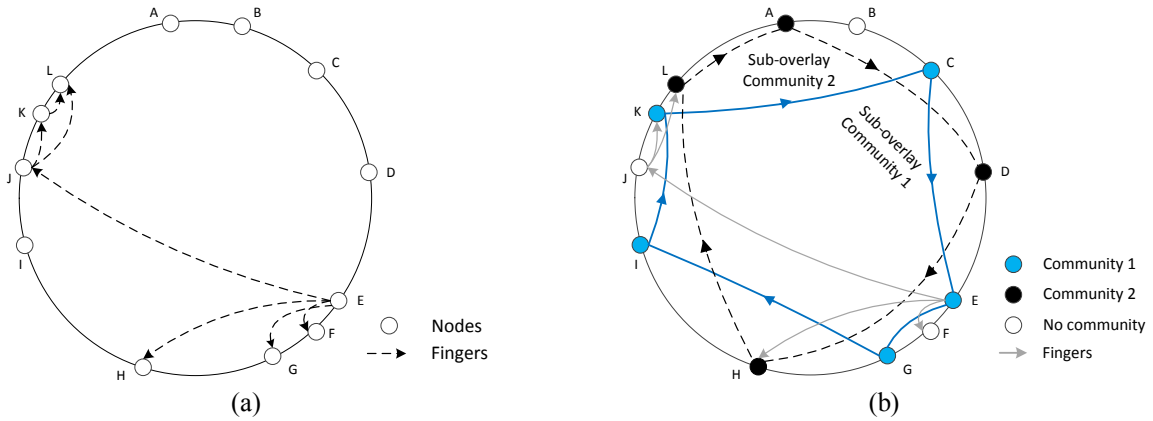


Figure 7.4 – Chord overlay network: (a) Node connectivity in Chord; (b) Two communities formed on top of the Chord overlay.

Section 7.3.1 presents how community members can be used to cache resources popular within a community. A mechanism to identify community members is presented in Section 7.3.2. Then requirements of community-influenced caching are discussed in Section 7.3.3.

7.3.1 Exploiting Community Members to Cache

A *community* is a subset of peers with common interests. However, members of a community may or may not be aware of each other. Figure 7.4(b) illustrates an overlay network having two communities. One of the communities, for example, may be based on semantics while the other may be based on geography. When communities are based on geography or organizational boundaries, nodes can be configured with their unique *Community Identifiers* (CIDs). However, some of the peers may not know their CIDs, may not be aware of the existence of a community with similar interests, or may not even belong to any of the communities. For such cases, solutions such as [Li09a, Gi10] may be extended to assign CIDs to nodes based on their similarity. Dissimilar metrics may be used to group the peers into communities. The only constraint is that each community needs to be identified using a unique CID. For rest of the discussion, we assume such decisions are taken at the application layer [Da03], outside of the overlay or caching solution. Assuming that each peer knows its CID, our goal is to facilitate routing of overlay messages related to a community via its members (by forming a sub-overlay) thus eliminating the inefficiency due to being in a common overlay. During the first couple of hops, the overlay messages tend to hop long distances in the key space and take alternative routes within overlay [Gu03, Ra10]. Messages converge in the last couple of hops as they approach the destination. Such behavior provides an opportunity for a node to reach its own community members in the first few hops, and then resolve queries using their caches. For example, suppose key k_L indexed at node n_L is popular within Community 1 (Fig. 7.4(b)). n_k is likely to cache k_L as it forwards many queries from its community members to n_L. Consequently, future queries for k_L can be answered at n_k reducing one-hop. This enables the communities to identify and cache resources of interest to them while enhancing the overall lookup performance. The destination node of a query may or may not belong to the same community as the query-originator, e.g., n_E and n_L belong to

two different communities. A querying node *n* forwards a query through members of its community *m* (regardless of destination node's community) under the assumption that "*a resource important to n is also important to other members of m and they may have queried it before n did. Therefore, the resource is likely to have been cached in one of the community members along the path*". This assumption and the flexibility of using alternative routes are exploited next to design the CBC solution for large-scale P2P systems with multiple communities. Our goal is to combine desirable features of structured P2P systems and caching in such a manner that multiple communities can coexist and benefit while being in a large P2P system. In doing so, we first propose a mechanism to form sub-overlays by identifying community members and then propose an algorithm to decide what resources to cache.

7.3.2 Sub-Overlay Formation

Suppose each community has a unique *CID*. Each node indicates its communities using one or more *CID*s or uses a predefined identifier to indicate that it is not in a community. Therefore, our solution supports communities based on different similarity measures or allows exceptions based on users' interest, e.g., a node in the U.S. may connect to a community in India just to access Hindi movies. Based on CIDs, nodes try to establish stronger connections among community members allowing them to forward queries through sub-overlays.

To build a sub-overlay, each node needs to identify other community members that are at approximately exponential distances in the key space. For example, it is useful for n_E (Fig. 7.4) to keep pointers to n_G, n_I, and n_K instead of n_F, n_H, and n_J. To take advantage of alternative routes, we need to identify members only for the higher-order pointers/fingers, i.e., ones that point to faraway nodes. To ease the identification process, each node advertises its *CID*s to its successor, predecessor, and other nodes that keep pointers to it. Such advertisements can be piggybacked onto overlay maintenance messages. However, given a large number of communities, it is unlikely that a node will identify members using only the advertisements that are sent to specific nodes. Nevertheless, if nodes receiving such advertisements are willing to track those *CID*s within their routing tables, other nodes may query them to find community

members. For example, if n_E queries n_J's routing table, it may get to know about n_K. The majority of the structured P2P systems such as Chord, Pastry, and Kademlia (Section 2.1.2) maintain many pointers. Therefore, nodes are likely to figure out at least one member for most of the higher-order pointers by sampling a few nodes.

Following mechanism is proposed to discover community members in Chord. Each node in Chord maintains at least $2 \log_2 N$ fingers. i-th finger ($b - 2 \log_2 N \leq i \leq b$) points to a key space of size 2^{i-1}, i.e., i-th finger can be used to reach any key within a distance of $[2^{i-1}, 2^i)$. Following lemma shows that both the node pointed to by the i-th finger and its successor each has a maximum of $i + 2 \log_2 N - b$ fingers, within this range.

Lemma 7.1. *By probing the finger tables of nodes pointed by the i-th finger and its successor* $2(i + 2 \log_2 N - b) - 1$ *distinct nodes can be identified.*

Proof. Consider the i-th finger of node A in Fig. 7.5. A maintains this finger to the successor B of key 2^{i-1}. Then, j-th finger of B points to the address $2^{i-1} + 2^{j-1}$ (let C be the successor of this address). ($j + 1$)-th finger of B points to the address $2^{i-1} + 2^j$ (let D be the successor of this address). Similarly, ($j + x$)-th finger of B points to the address $2^{i-1} + 2^{j+x-1}$. As the ($i + 1$)-th finger of A has to be greater than the last finger x of B, $2^i > 2^{i-1} + 2^{j+x-1}$. After some simplifications we get $x < i - j$. Though fingers in Chord span ($1 \leq i \leq b$), in practice minimum finger of a node starts at $b - 2 \log_2 N$ [St03]. Then $j = b - 2 \log_2 N$. Therefore,

$$0 \leq x < i - b + 2 \log_2 N \tag{7.1}$$

Similarly, k-th finger of C points to D which is the successor of address $2^{i-1} + 2^{j-1} + 2^{k-1}$. ($k + 1$)-th finger of C points to address $2^{i-1} + 2^{j-1} + 2^k$. Similarly, ($k + y$)-th finger of C points to $2^{i-1} + 2^{j-1} + 2^{k+y-1}$. As the ($i + 1$)-th finger of A has to be still greater than the last finger y, $2^i > 2^{i-1} + 2^{j-1} + 2^{k+y-1}$. Using the same argument as (7.1), we get

$$0 \leq y < i - b + 2\log_2 N \tag{7.2}$$

Therefore, B pointed to by the i-th finger and its successor C each has ($i + 2 \log_2 N - b$) fingers, within the range 2^{i-1}. Furthermore, C points to ($i + 2 \log_2 N - b - 1$) distinct nodes compared to

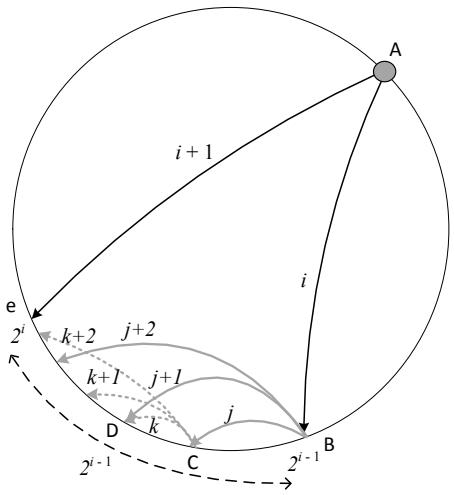


Figure 7.5 – Finger entries in Chord.

B. Thus, by probing the finger tables of those two nodes we can identify $2(i + 2\log_2 N - b) - 1$ distinct nodes. □

Thus, by probing the finger tables of successor of the *i*-th finger and its successor, we can identify $2(i + 2\log_2 N - b) - 1$ distinct nodes. If desired, the finger table of successor's successor may also be probed. Moreover, it can be proven that the probability of finding a community member increases with *i*:

Lemma 7.2. *Community members are more likely to be found for higher-order fingers using the proposed community-member identification scheme.*

Proof. Consider a community *m* with N_m nodes. Chord maps node to uniformly random locations in the ring using consistent hashing. Therefore, probability of sampling a community member anywhere in the ring is N_m/N. According to Lemma 7.1, when the finger number *i* increases (i.e., higher-order fingers), number of distinct nodes that can be found by sampling the finger tables of node pointed by the *i*-th finger and its successor increases. Let ε be the number of distinct nodes found by probing the finger table of the node pointed by the *i*-th finger and its successors. The probability that at least one of those ε nodes belongs to community *m* is:

$$P\{\text{Finding a community member}\} = 1 - \left(1 - \frac{N_m}{N}\right)^{\varepsilon} \tag{7.3}$$

ε increases with i and the number of successors' finger tables sampled. Consequently, the probability of finding a community member for higher-order fingers increases. □

For example, if the nodes are uniformly distributed in the key space, more community members are likely to be available between pointers to n_H and n_J than between n_G and n_H. Chord periodically refreshes finger table entries by sending maintenance messages to identify/validate nodes pointed by fingers. We can get those messages to probe the finger tables for community members. If a member is not found, the message is forwarded to the successor and its finger table is checked. It will be further forwarded to the successor's successor, if a member is still not found. Maintenance message of i-the finger should not be forwarded to the node pointed by the $(i + 1)$-the finger. We limit the number of hops to forward a maintenance message using the parameter hop_{max}. If a member is found, its contact details can be piggybacked onto the response to the maintenance message. If finger tables have limited capacity, nodes may replace the original Chord fingers with the fingers to community members. Otherwise, both fingers may be maintained for resilience.

If a node changes its community, members of the new community can be identified by refreshing the finger table either immediately or during the next cycle of overlay maintenance messages. Hence, nodes can identify relevant community members with minor overhead, and any structured P2P system that provides alternative routers can be used to relay messages through them. Furthermore, worst-case path length bound is still maintained as we preserve the properties of the overlay routing protocol. Our survey of BitTorrent users shows that though users are likely to access contents from multiple communities (e.g., 90% of the users accessed up to six communities), 89% of the time they access content from only one or two communities. Therefore, to improve the lookup performance a node needs to maintain fingers only for its primary set of communities. Details on survey questions and results are given in Appendix I. Overlay messages may be tagged with the *CID* of the source node so that intermediate nodes can use the suitable set of fingers while forwarding a message.

7.3.3 Community-Influenced Caching

As the messages are forwarded through sub-overlays, nodes are able to identify and cache resources that are relevant to their communities. Because we focus on communities' interests and preserve the overlay routing properties, local estimation of relative popularities is adequate to decide what a node should cache. For example, consider a node n with cache capacity $C_n = 1$. If messages mostly come from community members, and key k_a is requested more frequently than k_b, n will cache k_a. Sometimes n may observe even more requests for a k_c that is not accessed by its community. This occurs if n is along the path to an overlay neighbor (in ORT) that indexes a globally popular key (recall that different overlay routes converge as a message reaches its destination). In such a case, it is useful for n to cache k_c to improve overall lookup performance of the entire P2P system. When members of the community interested in k_c (if there is such a community) realize that it is a popular resource, they will add k_c to their own caches. Consequently, n will observe a lower demand for k_c, giving it the opportunity to cache k_a. Therefore, in contrast to previous solutions [Co02, Ra04, Ra07, Ra10], local statistics are adequate to provide a customized service to each of the communities. Next, we discuss how to distributedly allocate the limited cache capacity of each node optimally based on the local statistics and structure of overlay topology.

7.4 Distributed Caching

In Section 7.4.1 we first formulate the Distributed Local Caching (DLC) problem. DLC problem requires global information that is difficult to obtain. Hence, a relaxed version of the problem is formulated in Section 7.4.2 based on the overlay properties to answer the two key questions: *where* to place cache entries? and *how many* cache entries to create?. Based on this formulation, a heuristic-based caching algorithm is proposed in Section 7.4.3.

7.4.1 Distributed Local Caching

In DLC, each overlay node independently decides what keys to cache based on the *get*(*key*) messages that it forwards. For example, suppose n_J in Fig. 7.4 can cache only one key and each node indexes

only one key. If key k_L (indexed at node L) is requested more frequently than k_K, n_J should cache k_L and its corresponding *value*. Therefore, in contrast to previous solutions [Co02, Ra04, Ra07, Ra10], local statistics are adequate to determine what keys to cache at a node. Query arrivals in P2P systems show flash-crowds, as well as diurnal and seasonal effects [Ra04, Zh10]. Therefore, statistics such as periodic, network-wide query counts or arrival rate estimates, used in [Co02, De10, Ra04, Ra07, Ra10], are inadequate to decide effectively when and what to cache. Moreover, such distributed sampling messages introduce a significant overhead. Instead, nodes can still be made adaptive, if local statistics are collected at different granularities such that long-term and/or short-term popularity changes are properly captured. However, to design an effective solution both the local statistics and overlay topology must be taken into account. For example, suppose n_E forwards five messages to n_F and three messages to n_L through n_J. Based on local statistics n_E will cache n_F's resources. Therefore, n_E can answer five queries in the future (assuming same query characteristics) and reduce the total hop count by five-hops. However, if n_E caches n_L's resources, it can answer three queries while reducing the overall hop count by six-hops. Therefore, it is desirable to cache n_L's resources at n_E, instead of n_F's resources, as the objective of DLC is to improve the lookup performance at a node by reducing the path length of all queries that it forwards. However, reduction in path length cannot be accurately estimated unless topology information is available. Moreover, path length varies when nodes join and leave the network frequently. Such tradeoffs also need to be made when cache capacities of nodes are different (e.g., heterogeneous servers, desktops, and mobile devices) and size of resources varies (e.g., file size and a list of IP addresses of a domain name). Hence, overlay topology, cache capacity, size of resources, and their popularity need to be taken into account while determining where to place cache entries and how many cache entries to create.

Consider a P2P system with sets of **N** nodes and **K** keys, and let N and K represent the respective set sizes. Each node is selfish where a node tries to maximize the number of queries it can answer (irrespective of other nodes) by caching a subset of the (*key*, *value*) pairs. Let $S_k \in Z^+$ be the size of key $k \in \mathbf{K}$. Let $C_n \in Z^+$ be the cache capacity of node $n \in \mathbf{N}$. At each node $n \in \mathbf{N}$, there is a demand $\lambda_k^n \in Z^+$ for

$\forall k \in \mathbf{K}$ (e.g., number of queries received over a given period *t*). Assuming demand for *k* does not change in the near future, value of caching *k* depends on its demand (i.e., same number of queries can be answered locally) and the number of hops that will be reduced due to caching. Therefore, value of caching *k* at *n* is $v_k^n = \lambda_k^n h_k^n \in Z^+$ for $\forall k \in \mathbf{K}$, where $h_k^n \in Z^+$ is the number of hops required to resolve a query for *k* starting at *n*. Our goal is to minimize the average hop count at a node, given by:

$$h_{ave}^n = \frac{\text{Totalhops}}{\text{Totalqueries}} = \frac{\sum_{k \in \mathbf{K}} \lambda_k^n h_k^n \left(1 - x_k^n\right)}{\sum_{k \in \mathbf{K}} \lambda_k^n} \tag{7.4}$$

where $x_k^n \in \{0,1\}$ determines whether *k* is cached at node *n* ($x_k^n = 1$) or not ($x_k^n = 0$). This can also be interpreted as maximizing the hop count reduction while satisfying the node's cache-capacity constraint. Then the DLC problem can be formulated as an optimization problem:

$$\begin{aligned} &\text{maximize} R = \frac{\sum_{k \in \mathbf{K}} \lambda_k^n h_k^n}{\sum_{k \in \mathbf{K}} \lambda_k^n} - \frac{\sum_{k \in \mathbf{K}} \lambda_k^n h_k^n \left(1 - x_k^n\right)}{\sum_{k \in \mathbf{K}} \lambda_k^n} = \frac{\sum_{k \in \mathbf{K}} \lambda_k^n h_k^n x_k^n}{\sum_{k \in \mathbf{K}} \lambda_k^n} \\ &\text{subject to} \sum_{k \in \mathbf{K}} S_k x_k^n \le C_n \end{aligned} \tag{7.5}$$

Theorem 7.1. *DLC problem in* (7.5) *is NP-complete.*

Proof. We consider the decision problem of (7.5) as it simplifies the proof. The equivalent decision problem can be stated as follows:

Let x^n be a binary *K*-vector where $x^n = \left(x_1^n, x_2^n, x_3^n, \ldots, x_K^n\right)$. Given a network instance with no cache entries and an integer *T*, is there an assignment of (0, 1) values to x^n such that $R \ge T$ and capacity constraint is satisfied?

To show the NP-completeness, we need to first show that the problem belongs to NP and then prove the problem is NP-hard by reducing a known NP-complete problem to our decision problem [Co09a].

An algorithm which chooses (0, 1) values for elements of x^n, and then check the cache-capacity constraint can decide the problem in polynomial time $O(K)$. Therefore, decision problem belongs

to NP [Co09a]. Next, consider the following instance of the 0/1 Knapsack problem, which is known to be NP-complete [Ga78]:

Instance: A finite set **U**, for $\forall u \in \mathbf{U}$ let $s(u) \in \mathrm{Z}^+$ be the *size* and $v(u) \in \mathrm{Z}^+$ be the *value*, let $S \in \mathrm{Z}^+$ be the *size* constraint, and $V \in \mathrm{Z}^+$ be the *value* goal.

Question: Is there a subset $\mathbf{U}' \subseteq \mathbf{U}$ such that $\sum_{u \in U'} s(u) \leq S$ and $\sum_{u \in U'} v(u) \geq V$?

0/1 Knapsack problem has a one-to-one mapping to our decision problem where cache capacity of a node $C_n = S$, set of keys $\mathbf{K} = \mathbf{U}$, size of a key $S_k = s(u)$, value achieved by caching k at node n $v_k^n = \lambda_k^n h_k^n = v(u)$, value goal $T = V$, and subset $x^n \mathbf{K} = \mathbf{U}'$. This mapping can be done in polynomial time. Therefore, for every Knapsack instance we can construct a DLC problem. Hence, the 0/1 Knapsack problem reduces to the DLC problem. Therefore, the 0/1 Knapsack problem can be solved by solving the DLC problem. However, it is known that the 0/1 Knapsack problem is NP-complete [Ga78]. Therefore, DLC problem is also NP-complete. □

While the optimization problem is NP-complete when the content sizes (S_ks) are different, for the purpose of enhancing the lookup performance it is sufficient to assume S_ks are small and of similar size. For example, when resources are small (e.g., domain names), a cache entry can be a replica of the resource. When resources are large (e.g., files), then a cache entry can point to the location(s) of the resource. Therefore, for practical purposes, we can assume $S_k = 1$ for $\forall k \in \mathbf{K}$. Then the DLC problem for the purpose of content lookup can be formulated as follows:

$$\begin{aligned} &\text{maximize} R = \frac{\sum_{k \in \mathbf{K}} \lambda_k^n h_k^n x_k^n}{\sum_{k \in \mathbf{K}} \lambda_k^n} \\ &\text{subject to} \sum_{k \in \mathbf{K}} x_k^n \leq C_n \end{aligned} \tag{7.6}$$

This problem can be solved using a greedy algorithm [Ga78] that caches the set of resources with the highest value v_k^n. However, to calculate v_k^n we still need h_k^n, which is difficult to obtain in practice.

Next, by analyzing the globally optimal behavior and taking into account the structural properties of the ORT, we show that it is possible to develop a close-to-optimal caching solution without finding h_k^n.

7.4.2 Global-Knowledge-Based Distributed Caching

We formulate a relaxed version of the DLC problem in (7.6) that yields an analytical approximation to determine a suitable cache placement strategy. We consider the structure of ORT, as the topology is important in determining *where* and *how much* to cache. Figure 7.6(a) illustrates a Chord ring with 32 nodes occupying the entire address space (b = 5, address range [0, 31]). Figure 7.6(b) illustrates the asymmetric ORT corresponding to key seven (k_7) for the general case where each node sends one *get*(k_7) message. Branch weights indicate the number of messages forwarded from each node to its parent. A Chord ring with all the nodes is considered to simplify the following discussion. It was also confirmed (through simulations) that such an asymmetric tree exists even when only a small number of nodes are randomly mapped to the Chord ring (i.e., $N \ll 2^b$). Asymmetric ORT explains why the path length in Chord is bounded by $O(log_2 N)$, average path length is $½log_2 N$, and bell shaped distribution of path lengths (e.g., see the number of branches at each level of the tree listed on the right of Fig. 7.6(b)). To our knowledge, the relationship between asymmetric the ORT and above three properties was not observed in prior studies. Similar ORTs can be formulated for other structured P2P systems as well.

Next, we determine the best cache placement strategy given the asymmetric ORT. Node six (n_6) forwards the largest number of messages to n_7. Hence, if there is only one cache entry, it should be placed at n_6 such that all 16 lookup messages can be answered while reducing the number of hops by 16. Suppose there are two cache entries. First entry should be placed at n_6 and the remaining entry can be placed at either n_5 or n_4. If the second cache entry is placed at n_5, it reduces eight-hops and the total reduction is 16 + 8 = 24-hops. Instead, if the second cache entry is placed at n_4, it reduces two-hops for eight messages (between n_4 and n_6) and one-hop for remaining eight messages (between n_6 and n_7). Still the total reduction is 24-hops. If there are three cache entries, they should be placed at nodes n_6, n_5, and n_4, and the total

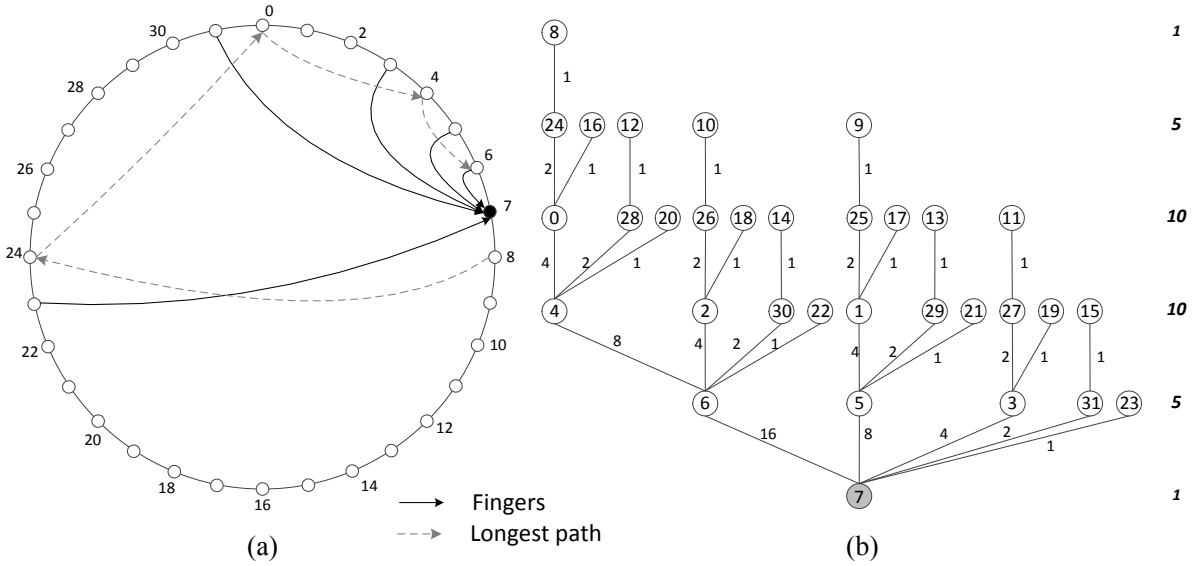


Figure 7.6 – Chord overlay: (a) Ring with 32 nodes. Only the predecessors of node 7 and the longest path is shown; (b) Overlay routing tree of node 7.

reduction is 32-hops. Similarly, if there is a fourth cache entry, it can be placed at n_3, n_2, n_1, or n_0, and 4-hops will be reduced. Following lemma determines the number of hops reduced by allocating c_k cache entries to k.

Lemma 7.3. *If $c_k \geq 1$ cache entries are allocated to key k, the number of hops reduced is*

$$g(c_k) = \frac{N}{2} \sum_{i=1}^{c_k} 1/2^{\lfloor \log_2 i \rfloor}$$

Proof. When cache entries are placed according to the structure of the ORT, the reduction in the number of hops due to addition of each cache entry follows the sequence 16 + 8 + 8 + 4 + 4 + 4 + 4 + 2 + …. When there are N nodes in the network, the height of the ORT is $\log_2 N$. Then the number of messages forward by the predecessor (node that forwards the most number of messages) can be given by (assuming each node sends one message):

$$2^{\log_2 N - 1} = N/2 \tag{7.7}$$

Thus, by placing the first cache entry at the predecessor same number of hops can be reduced. Each node where second and third cache entries are placed help to reduce the number of hops by ½ of (7.7). Similarly, each node where fourth to seventh cache entries are placed help to reduce

the number of hops by ¼ of (7.7). Each of the next eight cache entries will reduce the hop count by $^1/_8$ of (7.7). Therefore, the total number of hops reduced can be given by the sequence:

$$\frac{N}{2}\left(1+\frac{1}{2}+\frac{1}{2}+\frac{1}{4}+\frac{1}{4}+\frac{1}{4}+\frac{1}{4}+\frac{1}{8}+\frac{1}{8}+\frac{1}{8}+\frac{1}{8}+\frac{1}{8}+\cdots\right) \tag{7.8}$$

As c_k is finite, the total number of hops reduced $g(c_k)$ can be rewritten as:

$$g(c_k)=\frac{N}{2}\sum_{i=1}^{c_k}{}^{1}\!\Big/\!_{2^{\lfloor \log_2 i \rfloor}} \tag{7.9}$$

This completes the proof. □

When $c_k = 0$, $g(0) = 0$. The average reduction in hop count is given by $g(c_k)/N$, assuming each node sends a single message. Given that the ORT is asymmetric, this is the best cache placement strategy. PoPCache assumed that the ORT is symmetric, and hence cache entries were placed at level 2 nodes only after placing them at all the level 1 nodes. For example, a cache entry was placed at n_4 only after n_6, n_5, n_3, n_{31}, and n_{23}. Therefore, PoPCache did not effectively utilize the ORT to place cache entries.

Given the ORT-based cache placement strategy, we now determine how many cache entries to create for each key k ($c_k \in Z^+$) based on its popularity. Each key has a corresponding ORT and each overlay node belongs to multiple ORTs. Depending on a node's position in different ORTs and popularity of keys, it may have to cache multiple (*key*, *value*) pairs. However, how much a node can cache depends on its cache capacity C_n. We relax the per node cache-capacity constraint in (7.6), such that caching behavior with respect to each key can be examined separately. However, we still assume a fixed global cache budget B. $B = NC_{ave} \ll NK$, where C_{ave} is the average cache capacity of a node. Furthermore, assume the global popularity of keys is known and the normalized popularity of k is f_k ($0 < f_k \leq 1$). Keys are ordered according to their popularity $f_1 \geq f_2 \geq f_3 \geq \ldots \geq f_K$. Further assume each resource is of unit size (e.g., address of a node that indexes a resource). We name this scheme as Global-Knowledge-based Distributed Caching (GKDC). The corresponding relaxed GKDC-optimization problem is given in Lemma 7.4.

Lemma 7.4. *Relaxed GKDC-optimization problem can be formulated as:*

$$\text{minimize} H_{ave} = h_{ave} - \frac{1}{N}\sum_{k\in\mathbf{K}} f_k g(c_k)$$
$$\text{subject to} \sum_{k\in\mathbf{K}} c_k = B, \quad c_k \le N-1 \quad \forall k \in \mathbf{K}$$

Proof. Average number of hops reduced by placing c_k cache entries of k can be given by $g(c_k)/N$. Average number of hops h_{ave} required to find a key without caching is typically known for a given structured P2P system, e.g., in Chord $h_{ave} = ½ \log_2 N$. Therefore, the average number of hops to find k with caching is given by:

$$h_k^{caching} = h_{ave} - \frac{g(c_k)}{N} \tag{7.10}$$

Then the average hop count H_{ave} of the entire P2P system can be found by weighting the hop count of keys under caching by their normalized popularity. Therefore,

$$H_{ave} = \sum_{k\in\mathbf{K}} f_k h_k^{caching} = \sum_{k\in\mathbf{K}} f_k \left(h_{ave} - \frac{g(c_k)}{N} \right) \tag{7.11}$$

After some simplifications following can be obtained (as $\sum_{k\in\mathbf{K}} f_k = 1$):

$$H_{ave} = h_{ave} - \frac{1}{N}\sum_{k\in\mathbf{K}} f_k g(c_k) \tag{7.12}$$

Our goal is to minimize H_{ave}. As $KN \gg B$ we should utilize the total cache budget B to minimize H_{ave}. However, it is not useful to cache the same key multiple times on a node. Therefore, c_k should not exceed $N - 1$ (ignoring the root node, which already has a copy of the resource). Thus, (7.12) should satisfy the following constraints on c_k:

$$\sum_{k\in\mathbf{K}} c_k = B \quad \text{and} \quad c_k \le N-1 \quad \forall k \in \mathbf{K} \tag{7.13}$$

Together (7.12) and (7.13) complete the proof. □

The summation term in the objective function indicates the number of hops reduced due to caching. First constraint captures the global cache-capacity constraint. The second constraint bounds c_k, as it is not useful to cache the same key multiple times at a node. While formulating the optimization problem, PoPCache used the upper bound $O(\log N)$ instead of h_{ave}, did not bound c_k, and assumed that the ORT is

symmetric. Therefore, it provides a loose upper bound to H_{ave} (demonstrated in Section 7.6.1) and is unable to fully utilize B. Beehive does not consider a bounded B, structure of the ORT, and support only a Zipf's-like popularity distribution. Optimization problem can be restated as maximizing the hop count reduction:

$$\begin{aligned} &\text{maximize} \sum_{k \in \mathbf{K}} f_k g(c_k) \\ &\text{subject to} \sum_{k \in \mathbf{K}} c_k = B, \quad c_k \le N-1 \quad \forall k \in \mathbf{K} \end{aligned} \tag{7.14}$$

Theorem 7.2. *H_{ave} is minimized when*

$$c_k = \begin{cases} N-1 & \text{if } k \le l \\ \dfrac{f_k\left(B - l(N-1)\right)}{P(l, K, \alpha)} & \text{otherwise} \end{cases}$$

where P(l, K, α) is the sum of popularity of last K – l keys and l is the smallest key identifier that satisfies

$$\frac{f_{l+1}\left(B - l(N-1)\right)}{P(l, K, \alpha)} < N-1$$

Proof. This is a NonLinear-Integer-Programming (NLIP) problem hence cannot be solved analytically as it does not have a known structure [Li06]. Numerical solutions also fail even under moderately large N and K, which dramatically increase the search space of the NLIP problem. Therefore, we consider the corresponding relaxed NLIP problem, where $c_k \in R^+$, as it provides a useful lower bound [Li06] to the objective function in Lemma 7.4. $2^{\lfloor \log_2 i \rfloor}$ in $g(c_k)$ can be approximated as $2^{\log_2 i - \beta}$. β is a parameter that is used to indicate the upper bound ($\beta = 0$), lower bound ($\beta = 1$, when $c_k \ge 2$), and average value ($\beta = log_2\ 1.5$) of $2^{\lfloor \log_2 i \rfloor}$. Then $g(c_k)$ can be rewritten as:

$$g(c_k) = N 2^{\beta-1} \sum_{i=1}^{c_k} \frac{1}{i} \approx N 2^{\beta-1} (\gamma + \ln c_k) \tag{7.15}$$

where $\gamma = 0.5772156649$ is the Euler's constant. Then the objective function is:

$$N 2^{\beta-1} \sum_{k \in \mathbf{K}} f_k (\gamma + \ln c_k) = N 2^{\beta-1} \left(\gamma + \sum_{k \in \mathbf{K}} f_k \ln c_k \right) \tag{7.16}$$

We only need to focus on the summation term in (7.16) and the constraints in (7.14), as rest of the parameters are constant for a given network. This is a global optimization problem, as (7.16) is concave [Bo04] and the corresponding minimization problem is convex. Karush–Kuhn–Tucker (KKT) conditions [Bo04] can be used to solve the minimization problem of (7.16). Then (7.16) should satisfy the following (using formula for corresponding minimization problem):

$$-\nabla\sum_{k\in\mathbf{K}} f_k \ln c_k + \lambda\nabla\left(\sum_{k\in\mathbf{K}} c_k - B\right) + \mu\nabla(c_k - N + 1) = 0 \tag{7.17}$$

From stationarity and dual feasibility:

$$-f_k/c_k + \lambda + \mu_k = 0, \quad \mu_k \geq 0 \quad \forall k \in \mathbf{K} \tag{7.18}$$

From the complementary slackness:

$$\mu_k(c_k - N + 1) = 0 \quad \forall k \in \mathbf{K} \tag{7.19}$$

There are $2K + 1$ unknowns and same number of equations. However, complementary slackness provides only a check. Hence, we need to substitute values for λ and μ_k, and then check the feasibility. Suppose $\mu_k = 0 \ \forall k \in \mathbf{K}$. Then by substituting $\lambda = f_k/c_k$ (from (7.18)) to the budget constraint, $1/\lambda = B$. Therefore, $c_k = f_k B$. As $c_k \leq N - 1$, $c_k = f_k B \leq N - 1$. Then $f_k \leq (N - 1)/NC_{ave} \approx 1/C_{ave}$. This implies that f_k cannot be large (e.g., if $C_{ave} = 20$, $f_k \leq 0.05$). However, in practice, the most popular key k_1 may contribute to a large fraction of the queries, if the popularity distribution is skewed. For example, $f_1 = 0.092$ for a Zipf's distribution with $\alpha = 1.0$ and $K = 30{,}000$. Therefore, the solution is not optimum. Suppose $\lambda = 0$. Then $\mu_k = f_k/c_k$ suggests $\mu_k > 0$. From (7.19), $c_k = N - 1 \ \forall k \in \mathbf{K}$. This cannot be true as $B \ll NK$. Suppose $c_k < N - 1$ for some of the keys. For those keys $\mu_k = 0$ (from (7.19)). Suppose the most popular l keys are cached in all $N - 1$ nodes. Then from the budget constraint:

$$\begin{aligned} l(N-1) + \sum_{k=l+1}^{K} c_k &= l(N-1) + \sum_{k=l+1}^{K} \frac{f_k}{\lambda} = B \\ \frac{1}{\lambda}\sum_{k=l+1}^{K} f_k &= \frac{1}{\lambda} P(l, K, \alpha) = B - l(N-1) \end{aligned} \tag{7.20}$$

where $P(l, K, \alpha)$ is the sum of popularity of last $K - l$ keys. α is a parameter that captures the popularity distribution of keys (e.g., Zipf's parameter). By substituting $1/\lambda$ from (7.20) in $\lambda = f_k/c_k$:

$$c_k = \begin{cases} N-1 & \text{if } k \le l \\ \dfrac{f_k\left(B - l(N-1)\right)}{P(l,K,\alpha)} & \text{else} \end{cases} \tag{7.21}$$

and l is the smallest key that satisfies $c_{l+1} < N - 1$. □

Equation (7.21) suggests that the most popular keys should be cached in all the nodes, and the remaining cache capacity $B - l(N - 1)$ should be allocated in proportion to the popularity of rest of the keys. This is the best cache-capacity-allocation strategy given that the ORT is asymmetric. Therefore, in contrast to PoPCache, we are able to fully utilize the available cache capacity B and provide a tight bound to H_{ave}. Moreover, (7.21) is valid for any popularity distribution. H_{ave} can be determined by substituting (7.21) in Lemma 7.4. It is also possible to solve the optimization problem to determine the required cache budget B given a target H_{ave}. The optimum value of the objective function provides a lower bound to the original NLIP problem as the optimization problem is convex [Li06]. It was observed that the numerical solution to the relaxed-NLIP problem (i.e., $c_k \in \mathrm{R}^+$) is identical to our analytical solution. Correctness of the analytical solution and comparison with PoPCache are presented in Section 7.6.1. While these bounds are valid for Chord, we believe a similar approach will yield the bounds for other structured overlays by taking into account the structure of their ORTs.

7.4.3 Local-Knowledge-Based Distributed Caching

Cache placement and capacity allocation strategies obtained using the analysis of GKDC can be used to develop a heuristic-based algorithm for the DLC problem. Asymmetric ORT indicates that a key should be cached first at the node that forwards the largest number of messages. Then at one of the nodes that forwards the second largest number of messages, and so on. This will result in a consistent reduction in path length of messages. Theorem 7.2 says that the cache capacity should be allocated in proportion to

the global popularity of keys. In DLC, nodes are not aware of the global popularity of keys. However, most popular keys are evident throughout corresponding ORTs while moderately popular ones are evident at lower levels of the ORTs. Therefore, a good approximation to the proportional allocation can be obtained using a heuristic that captures the relative popularity of keys at a node. For example, if a node with cache capacity $C_n = 1$ forwards messages of k_a more frequently than messages of k_b, it should cache k_a. Moreover, such a heuristic enables the enforcement of per node capacity constraint where a node will cache locally most popular C_n keys. Furthermore, local statistics can be collected at different granularities such that long-term and/or short-term popularity changes are properly captured. We propose a caching algorithm based on the perfect Least Frequently Used (LFU) algorithm to reduce the caches from being thrashed while ensuring overlay dynamics are sufficiently captured.

Figure 7.7 illustrates the proposed caching algorithm described using the common API in [Da03]. Each node *n* has a *cache*, which can store up to C_n (*key*, *value*) pairs. Each overlay message (*msg*) has a *source* node, message *type* (*put* or *get*), and a *key*. For each *get*(*key*) message that a node receives, we track *key*'s *demand* $\in R^+$ using a (*key*, *demand*) pair which is stored in the lookup table *LT*. *get*() messages also maintain a list of nodes (*cList*) that have decided to cache the resource. ***forward*** is an upcall, to the DHT layer, invoked at each node that forwards a message. It enables intermediate nodes to cache, collect statistics, or drop messages. *put*(*key*, *value*) messages are handled as usual. When a *get*(*key*) message is received, each node keeps track of the demand for the corresponding *key* regardless of whether it is already cached or not (line 4 in Fig. 7.7). The local *cache* is then checked to see whether the *msg* can be answered. If so, replies are directly sent to the *source* node and to the list of nodes in the *cList* that are interested in caching the resource (lines 5-8). *msg* is then dropped (line 9). If the *key* is not in the *cache*, the node tries to determine whether it is useful to get a copy of the resource. If the *cache* is already full, it checks whether the given *key* has a higher demand than the LFU cache entry (lines 12-13). A node may also request a copy of the resource, if the *cache* is not fully occupied and the demand is above the caching threshold T_{cache} (lines 17-18). In either case, the node appends (piggyback) its identifier to the *cList* in the *msg*. The *msg* is then forwarded to the next node. Intermediate nodes may also append their identifiers to

```
void forward(key, msg, nextHop*)
1   If msg.type = PUT                    //put message
2      return
3   If msg.type = GET                    //get message
4      addLookup(key)                    //Track demand
5      If key ∈ cache                    //In cache
6         sendDirect(msg.source, key, cache[key])
7         For each i in msg.cList[ ]     //Send to each cache requester
8            sendDirect(msg.cList[i], key, cache[key])
9         nextHop ← NULL                 //Drop original get message
10     Else                              //Not in cache
11        If cache.size( ) = C_n         //Cache already full
12           key_lowest ← getCachedKeyWithLowestDemand(LT[ ])
13           If LT[key] > LT[key_lowest] //Higher demand
14              msg.cList[ ] ← myNodeID  //Request a copy
15              delete cache[key_lowest] //Remove lowest
16        Else
17           If LT[key] > T_cache        // Higher demand
18              msg.cList[ ] ← myNodeID  //Request a copy

void addLookup(key)
19  For each i in LT[ ]
20     If i = key                        //Increase demand for key
21        LT[i] = (1 + θ) × LT[i]
22     Else                              //Decrease demand for others
23        LT[i] = (1 – θ) × LT[i]
24     If LT[i] < T_remove               //Very low demand
25        delete LT[i]                   //Remove key
26  If key not in LT
27     LT[key] ← θ
```

Figure 7.7 – Local knowledge-based distributed caching algorithm.

the *msg*, if they also decide to cache the resource. The threshold T_{cache} reduces the caching overhead and cache thrashing.

If resources are small and relatively static (e.g., domain names), then a cache entry can be a replica of the resource. If resources are large, mutable, or the set of peers having a resource is dynamic (e.g., files and processor cycles), then the cache entries can point to sources of the resources. Therefore, our caching scheme can locate all copies of small and relatively stable resources, or point to a subset of large, mutable, or dynamic resources. The caching algorithm works with any distribution of queries and gain better performance when queries are highly skewed.

Using the ***addLookup*** function, a node also tracks the demand for keys that are not in the cache but forwards messages through it. Thus, LKDC algorithm is a *perfect LFU* algorithm. It enables the nodes to rapidly adapt to varying popularity and arrival patterns. However, it is important to properly balance

the past and new information to reduce the caching overhead. Therefore, whenever a new message arrives, a node multiplies the current demand of the key by a weighting factor $(1 + \theta)$, $0 \leq \theta \leq 1$. *demand* of all other keys in the *LT* is multiplied by $(1 - \theta)$. If a key appears for the first time its demand is set to θ (lines 26-27). When θ is close to zero, the algorithm is biased towards the past thus effectively responding to long-term trends. When θ is closer to unity, bias is towards the current information, thus a node responds to rapid changes. θ and T_{cache} control the adaptability of the caching solution while minimizing unnecessary cache requests. It is recommended to set $T_{cache} > \theta$ to reduce cache thrashing. When the caching scheme is applied to multiple communities, each community may use a different θ based on its perceived behavior. Though perfect LFU algorithms are known to take better caching decisions, they have a higher overhead as the *LT* can grow arbitrarily large. A threshold (T_{remove}) is used to remove keys without sufficient *demand* thereby limiting the size of *LT*. The computational cost of the algorithm is $O(size(LT))$. As *LT* is not large with respect to *K*, we can afford to execute the algorithm every time a *get(key)* message arrives. Therefore, it can rapidly adapt to changing popularity, message arrival patterns, and piggyback cache requests on *get*(key) messages while incurring minimum overhead.

7.5 Simulation Setup

To validate the analysis in Section 7.4, we first simulated an overlay network with 1,000-5,000 nodes using the OverSim P2P simulator [Ba07b]. Caching algorithms were implemented on top of Chord, and the Zipf's parameter α was varied from 0.5 to 1.5. For comparison, PoPCache was also simulated with accurate global popularity of keys. CBC is simulated using a 15,000-node network with ten communities. Though hardware limitations prevented us from simulating a much larger network, the size of our network is either comparable or larger than prior studies such as [Ra04] and [Ra07]. Nodes were assigned to different communities as shown in Table 7.4. Parameters for each community were selected to observe the behavior under different scenarios. Zipf's and similarity parameters were selected based on our own observations in Table 7.1, Table 7.2, and [Ra04, Sr01]. To simplify the performance analysis, a static

Table 7.4 – Configuration of different communities.

Community	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}
No of nodes (apx.)	600	600	600	1,200	1,200	1,200	1,200	1,200	2,400	4,800
Zipf's parameter	0.85	0.95	1.10	0.5	0.80	0.80	1.0	0.90	0.90	0.75
No of keys	40,000	30,000	30,000	40,000	40,000	40,000	50,000	50,000	50,000	50,000
Similarity with community (x)	0.2 (m_8)	0	0.1 (m_7)	0.2 (m_9)	0.3 (m_8) 0.5 (m_7)	0	0.1 (m_3) 0.5 (m_5)	0.3 (m_5) 0.2 (m_1)	0.4 (m_1) 0.2 (m_4) 0.3 (m_{10})	0.3 (m_9)
Queries for k_1	4,516	8,535	17,100	603	6,454	6,454	21,059	11,956	23,911	17,030

network is assumed and queries were issued only after the network was stabilized (around 2,000 s). Each node issued queries based on a Poisson distribution with a mean inter-arrival time of 15 seconds. Based on the simulations, we observed that it is sufficient to set $T_{remove} = \theta^{10}$ (as query demands are weighted) to gain good performance while limiting the lookup table size to 50-80 entries. To measure the ability of geographic communities to improve the latency, transit-stub networks with 10 ASs and 750 routers were generated using BRITE [Me01] while using GT-ITM [Ze96] as the underlying topology generator. Nodes in the same community are assigned to the same AS. Based on [Ca02], overlay node to router delay is set to 1 ms and the average delay of the core network links is set to 40 ms. Results are based on ten samples, which were sufficient to attain average number of hops within ±5% accuracy and 95% confidence level. Additional details on the simulator are given in Appendix II.4.

7.6 Performance Analysis

We first validate the analytical results obtained in Section 7.4 using a network with a single community. Then performance of CBC is evaluated under different α, c_n, T_{cache}, geographic communities, and sudden popularity inversion.

7.6.1 Local-Knowledge-Based Distributed Caching

Figure 7.8(a) compares the analytical and simulation results while varying N. Our analytical solution using discrete $g(c_k)$ from Lemma 7.3 and its continuous approximation using $\beta = log_2 1.5$ closely

match the simulation results. $g(c_k)$-based model provides an upper bound to all the simulation results. Difference between LKDC algorithm (*LKDC-Sim*) and GKDC (*GKDC-Sim*) is ~0.4-hops. Therefore, LKDC provides a desirable caching solution without the cost of estimating global popularity, structure of the ORT, or relaxing the per node cache-capacity constraint. The PoPCache analytical model is derived using the upper bound $O(\log N)$ for path length. Therefore, the resulting average hop count (H_{ave}) is very high, e.g., H_{ave} = 8.82 for setup in Fig. 7.8(a). Hence, as a simple correction, we replaced the upper bound with h_{ave}. Still, corrected PoPCache analytical model overestimates H_{ave} by 1.1-1.4-hops (see Fig. 7.8(a)). Performance of *PoPCache-Sim* (with accurate global popularity information) and *LKDC-Sim* is similar. However, *PoPCache-Sim* placed 258 keys in one of the nodes whereas *LKDC-Sim* placed only 20 keys in a node (reduce largest index size by 12.9 times). Therefore, our algorithm is more useful as it does not require relaxing the cache-capacity constraint or sampling messages to estimate the popularity. Moreover, PoPCache performance could degrade in real deployments where popularity estimation is susceptible to sampling errors. *GKDC-Sim* has the lowest H_{ave}. It was realized that, though the ORTs in our simulations were asymmetric, branch weights were somewhat off from Fig. 7.6(b). For example, the most popular path was carrying 55-65% of the queries though our model assumed 50%. Therefore, the most popular path answers more quires (this is particularly useful for moderately popular keys) consequently reducing H_{ave} of the overall network. This explains why the analytical model provides a useful upper bound. It was

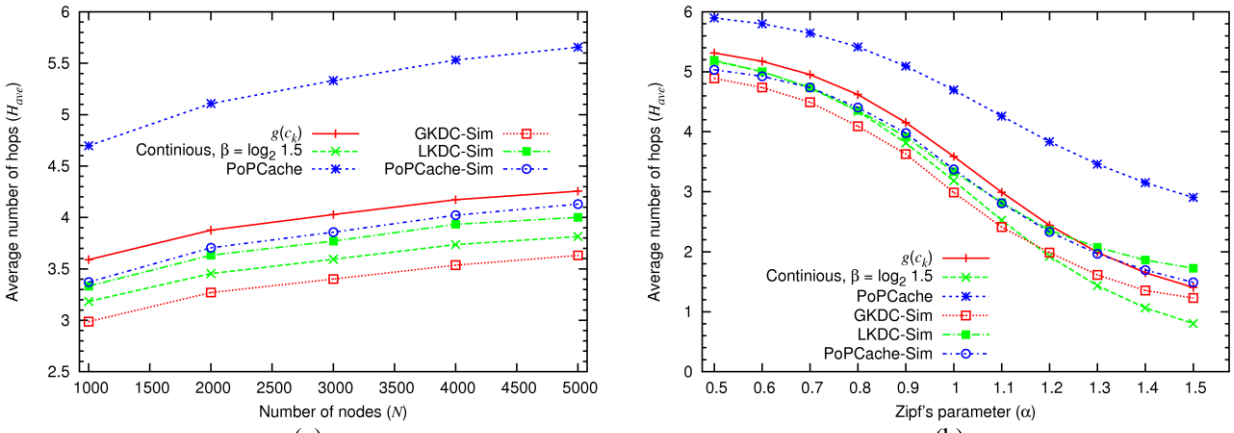


Figure 7.8 – Average hop count: (a) While varying number of nodes. $K = 30N$, $\alpha = 1.0$, $C_n = 20$, $T_{cache} = 0.12$, (b) While varying Zipf's parameter. $N = 1{,}000$, $K = 30{,}000$, $C_n = 20$, $T_{cache} = 0.12$.

observed that *LKDC-Sim* naturally arranges cache entries among nodes reflecting the structure of the ORT, while it has to be explicitly defined in [Ra04, Ra07, Ra10]. When many nodes at higher levels of the ORT start caching popular keys, they do not forward messages to lower-level nodes. Therefore, the relative popularity of keys cached at lower-level nodes reduces. Consequently, cache storage allocated to those keys is reallocated to less popular keys. This is not possible in PoPCache and Beehive, as they force all the nodes along the ORT or within a specific address range to cache keys.

Analytical and simulation results with varying Zipf's parameters (α) are shown in Fig. 7.8(b). Performance trends are same for $\alpha \leq 1.0$. However, when $\alpha > 1.0$, analytical solution tends to underestimate H_{ave} compared to the simulation results. This is also a consequence of slight differences in branch weights of ORTs. When the popularity distribution is highly skewed (e.g., $\alpha > 1.0$), only the most popular keys are cached. These keys will place more and more cache entries at higher levels of the ORT assuming branches other than the most popular one will get ~50% of the queries. However, in practice they get only 35-45% of the queries. Therefore, our analytical model underestimates H_{ave}. Alternatively, many keys are cached when the popularity distribution is less skewed. These moderately popular keys can benefit from the large fraction of queries (> 50%) that goes through the most popular branch in the ORT. Therefore, actual H_{ave} is below the theoretical value. Hence, our analytical model provides an upper bound to H_{ave} when $\alpha \leq 1.0$ and a lower bound when $\alpha > 1.0$. Production P2P systems are known to have $\alpha \leq 1.0$ [Ra04, Ra07, Sr01] hence our analytical solution is also useful for production systems.

Mixed Integer NonLinear Programming (MINLP) solution provides a lower bound for the NLIP problem [Li06]. As we were unable to solve the NLIP problems neither analytically nor numerically (for large N and K), we compare the performance against the following family of allocations:

$$c_k = \frac{(f_k)^r B}{\sum_{k \in \mathbf{K}} (f_k)^r} \tag{7.22}$$

where $r \in R^+$. $r = 1$ for PoPCache and $r = 0.5$ for unstructured P2P [Co02]. Figure 7.9 shows H_{ave} against r. MINLP-based solution provides the lower bound while $g(c_k)$-based discrete solution provides the upper

bound. Minimum value for (7.22) is obtained when $r \approx 0.95$, and it is higher than both *LKDC-Sim* and *GKDC-Sim*. $r < 1.0$ suggests that slightly more cache capacity should be allocated for less popular keys (when the distribution is highly skewed), which justifies the allocation in Theorem 7.2. Optimum r varies with N, K, and α. Therefore, (7.21) provides a consistent cache capacity allocation mechanism under a range of K, N, and α values while providing better lookup performance. Impact of parameters such as C_n and T_{cache} are discussed in the next section.

7.6.2 Community-Based Caching

Figure 7.10 compares H_{ave} of the entire system under *Chord*, *passive caching* (i.e., nodes cache responses to their own queries), *caching* (i.e., *LKDC* algorithm without sub-overlays), and the overall *community-caching* solution (i.e., *LKDC* deployed on top of sub-overlays). Lookup performance converges with all the caching schemes as the query distribution is steady. *Passive caching* reduces the H_{ave} from 7.79-hops to 7.32-hops and it is further reduced to 5.88 and 4.64 hops by *caching* and *community caching*, respectively. Thus, our caching solution reduces H_{ave} by 40.5%.

H_{ave} observed by each community is shown in Fig. 7.11. Communities with highly skewed query distributions (i.e., large α) and/or lower number of distinct keys gained significant performance improvement. For example, H_{ave} of communities m_3, m_7, and m_9 is reduced by 53%, 48%, and 48%, respectively. Moderately popular communities m_1, m_2, m_5, m_6, m_8, and m_{10} gained 31-42% improvement based

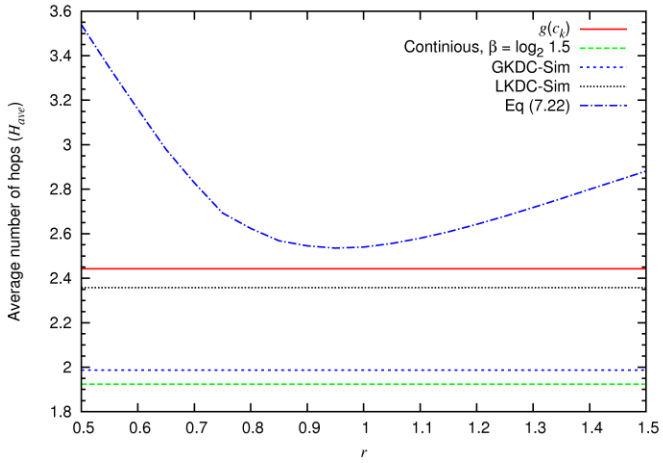


Figure 7.9 – Validation of optimum hop count. $N = 1{,}000$, $K = 30{,}000$, $\alpha = 1.2$, $C_n = 20$, $T_{cache} = 0.12$.

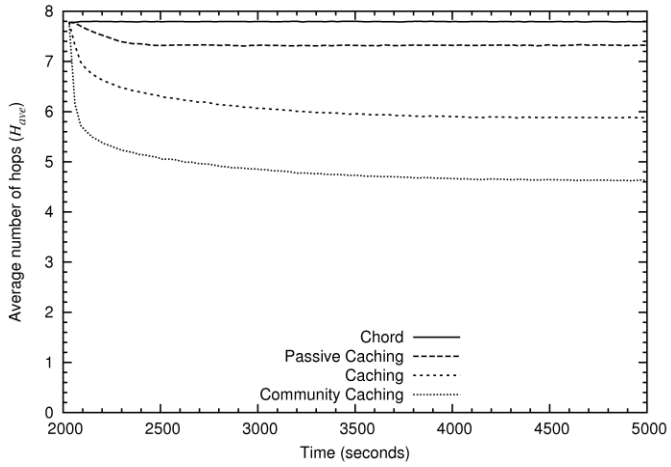


Figure 7.10 – Average hop count vs. time. Bucket size = 30 s, $C_n = 20$, $\theta = 0.1$, $T_{cache} = 0.12$, $hop_{max} = 4$.

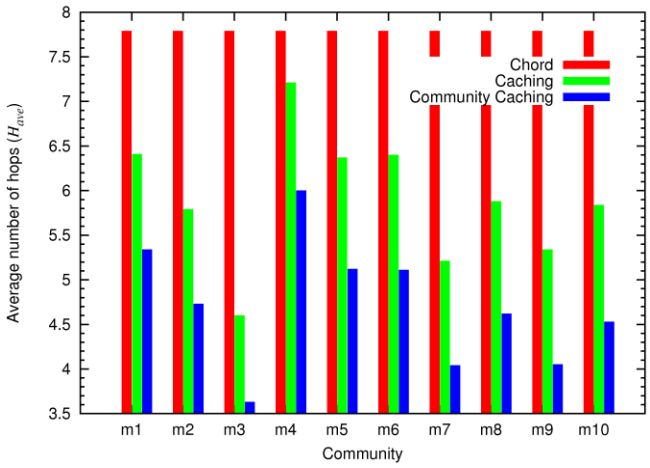


Figure 7.11 – Average hop count observed by each community at the steady state. $C_n = 20$, $\theta = 0.1$, $T_{\text{cache}} = 0.12$, and $hop_{max} = 4$.

on their specific distributions. The most popular query in m_4 had a global rank of 285 (Table 7.4). Therefore, it gained only 7.4% improvement with *caching*. However, *community caching* was able to reduce the hop count by 23% (3.1 times improvement over *caching*). Performance gain by each community was dependent only on its popularity distribution, and both large and small communities benefited equally, e.g., $\{m_1, m_5$ and $m_6\}$, and $\{m_7$ and $m_9\}$. No correlation between performance and similarities among communities was observed.

Next, we measure the impact of number of communities in the system and their relative sizes on lookup performance. When the number of communities $M = 1$, it is not necessary to run the sub-overlay formation scheme as fingers already point to community members. However, LKDC can still be used to

identify and cache popular resources using only the local statistics. If $M = N$, each node is in its own community. Therefore, sub-overlay formation scheme is not useful as there are no other community members. If users are completely independent and access random contents, caching is not useful either. However, it has been shown in [Ba11a] that even for seemingly unrelated events, if there is some preference it could lead to a skewed distribution. For example, independent users downloading a song that they learned from an external source such as media. As far as there is some skewness in resource access, LKDC algorithm can be used to improve the lookup performance and the performance gain will depend on the skewness of the popularity distribution and cache capacity.

To analyze the performance under a different number of communities and their relative sizes, we split community 6 (m_6 in Table 7.4) into a set of small communities. m_6 was selected as it is independent from rest of the communities; therefore, its size can be varied without affecting other communities. m_6 was split into 600×2 (two communities each with 600 nodes), 300×4 (four communities each with 300 nodes), 150×8, and 75×16 communities while keeping the total number of nodes as constant (N = 15,000). This resulted in a set of networks with 10, 11, 13, 17, and 25 communities respectively. Figure 7.12(a) shows the lookup performance while varying the number of nodes in communities. H_{ave} increased by 0.53-hops (10% increase) when the community size is reduced from 1,200 nodes to 75 (16 times smaller than original m_6 and 0.5% of the overall network). However, the formation of sub-overlays still provides much better lookup performance (at least 16% improvement) than applying only the proposed caching algorithm. Thus, the proposed sub-overlay formation scheme is effective in finding some of the community members for higher-order fingers even when the size of a community is relatively small. Figure 7.12(b) illustrates that H_{ave} of rest of the communities were not significantly impacted while m_6 was split into smaller and smaller sized communities. Therefore, relative size among different communities (or their ratios) does not affect the lookup performance gained by a community. Hence, performance mainly depends on the skewness of resource popularity, cache capacity, and relative size of a community with respect to the size of the overall network. However, if the community size is very small compared to N (e.g., in decentralized social networks with few members), it may not be possible to find a community

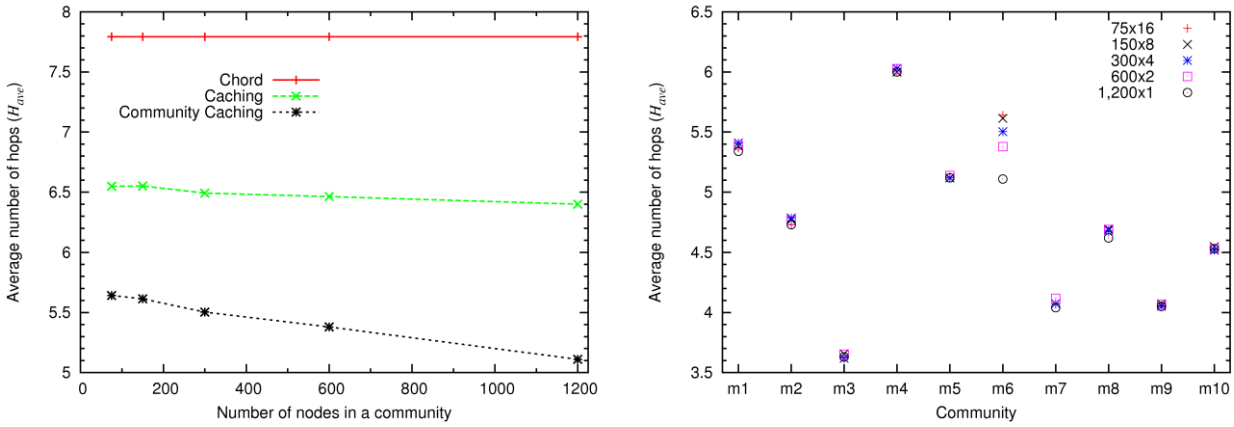


Figure 7.12 – Lookup performance under varying number of communities and community sizes obtained by splitting community six: (a) Average hop count while varying the number of nodes in a community; (b) Average hop count of all the communities. C_n = 20, α = 0.1, T_{cache} = 0.12, and hop_{max} = 4.

member even for higher order fingers. In such cases, explicit community tracking schemes such as [Gi10] may be extended to identify community members.

Figure 7.13 shows 33-50% reduction in latency when communities are based on geography. When communities are large, a node can find community members for most of the overlay pointers/fingers. Such communities appear as a sub-overlay sustained using only its members. Consequently, popularities are preserved, overlay traffic is localized within the community, and the effective size of the sub-overlay is equal to the number of nodes in the community (e.g., m_9 and m_{10} appear as two sub-overlays with 2,400 and 4,800 nodes, respectively). As the traffic is mostly local, m_9 and m_{10} gained 45% and 50% reduction in latency. We use the continuous approximation of Theorem 7.2 (when $\beta = log_2$ 1.5) to estimate the lower bound of H_{ave} for each community and the overall system. Corresponding h_{ave} is used for m_9 and m_{10}. For the overall system, the model predicts H_{ave} = 4.53 and according to the simulations H_{ave} = 4.63. Thus, the model can also be used to obtain a useful lower bound for a large P2P system with multiple communities.

Figure 7.14 plots the cumulative distribution of hops. As expected, *community caching* was able to respond to most of the queries within the first few hops. *Chord* resolved 65% of the queries within 8-hops while *community caching* was able to resolve 96% of the queries by then. P*assive caching* initially

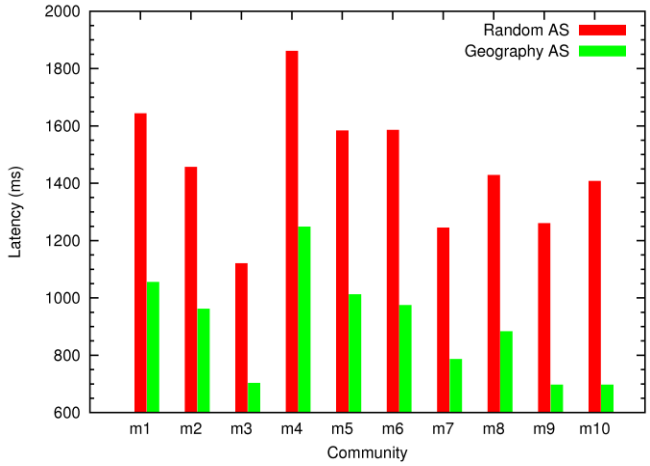


Figure 7.13 – Latency of geographic communities using community caching. C_n= 20, θ = 0.1, T_{cache} = 0.12, and hop_{max} = 4.

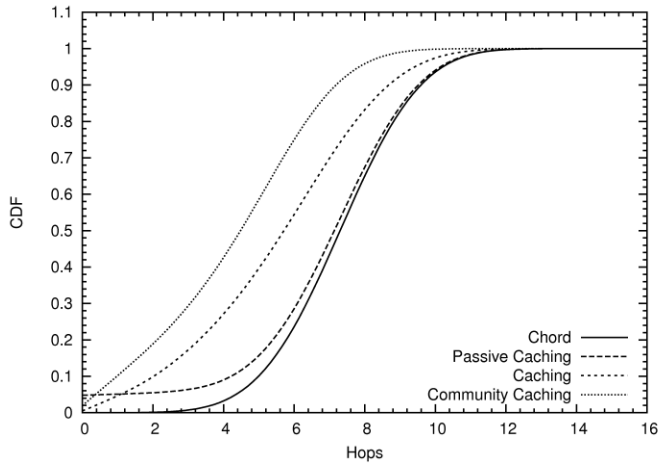


Figure 7.14 – Cumulative distribution of overlay hops required to resolve queries. C_n = 20, θ = 0.1, T_{cache} = 0.12, and hop_{max} = 4.

has a higher hit rate as some nodes respond to their own queries based on the past results. Figures 7.10 to 7.14 confirm that by focusing on individual communities, it is possible to improve both the communitywide and system-wide lookup performance.

Performance gain with increasing C_n is shown in Fig. 7.15. Though H_{ave} rapidly reduces with increasing C_n, it tends to saturate after a while. This is an artifact of the Zipf's-like popularity distribution where significant performance can be gained by caching a few highly popular resources. Yet, diminishing return is gained with very large caches. Therefore, while trying to provide a guaranteed mean, both Beehive and PoPCache had to force the nodes to cache several hundreds of resources on average and several

thousands in the worst case, regardless of nodes' capabilities or interests. In contrast, our caching scheme provides comparable lookup performance using small caches. We also compared the lookup performance against heterogeneous cache capacities where C_n is set to ~U(10, 30) and ~U(0, 40) (still C_{ave} = 20). H_{ave} = 4.63 when C_n = 20 and it increases to 4.77 and 4.88 when C_n is ~ U(10, 30) and ~U(0, 40), respectively. This confirms that our solution can effectively adapt to varying C_n as H_{ave} is increased by only 0.25-hops (5.4% increase).

Network converges faster and attains minimum H_{ave} when the weighting factor θ = 0.1. Such a low value of θ stabilizes the network based on long-term trends in popularity. Impact of caching threshold T_{cache} on the convergence time is shown in Fig. 7.16. Higher T_{cache} values increase the convergence time but reduce the caching overhead (see Table 7.5). Alternatively, lower thresholds (e.g., T_{cache} = 0.1) rapidly respond to popular queries consequently improving the lookup performance. This trend continues until caches get full. When a cache is full and T_{cache} is lower, even a key with a marginally higher demand than θ forces one of the cached keys to be flushed. When a query for the flushed key appears again, its demand increases forcing another key to be flushed. Repetition of this process results in cache thrashing and increased path length due to higher miss rate. For the given setup, θ = 0.1 and T_{cache} = 0.12 balance both the convergence time and caching overhead.

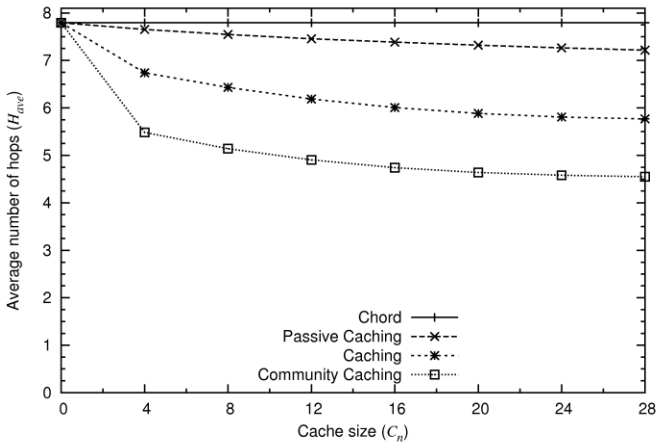


Figure 7.15 – Lookup performance under varying cache size (C_n). Steady state results with θ = 0.1, T_{cache} = 0.12, and hop_{max} = 4.

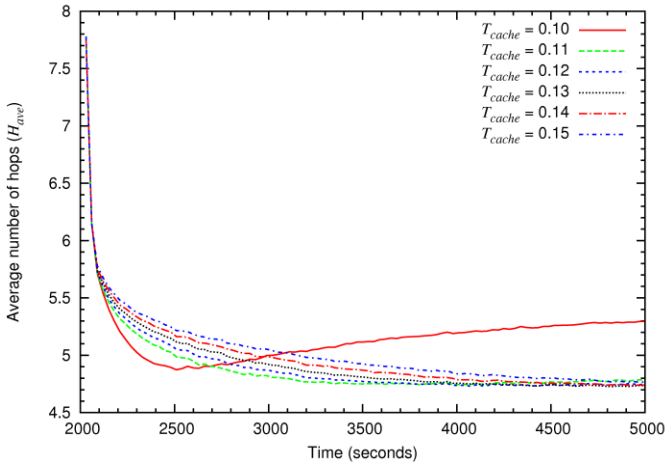


Figure 7.16 – Caching threshold's impact on convergence time in community caching. Bucket size = 30 s, $C_n = 20$, $\theta = 0.1$, and $hop_{max} = 4$.

Table 7.5 – Number of cache requests per node in community-based caching.

T_{cache}	0.1	0.11	0.12	0.13	0.14	0.15
Average	281.4	34.9	25.8	20.3	16.2	13.5
Minimum	0	0	0	0	0	0
Maximum	1,611.7	233.5	159.2	122.1	83.9	65.3

To observe the adaptability of CBC to rapid popularity changes, we invert the popularities of queries, where the least popular query suddenly becomes the most popular and vice versa. This is a worst-case scenario. Figure 7.17 shows the convergence of the network after popularly inversion around 4,000 seconds. H_{ave} increased only by 0.5-hops and the network stabilized following the same convergence pattern. It is sufficient to select $hop_{max} = 4$. We do not expect a significant increase in hop_{max}, even for a very large network, as it is inversely proportional to community size N_m. Our solution introduces minimum overhead as cache and community-member-discovery requests are piggybacked on *get()* and overlay maintenance messages, respectively. Caching also alleviates hot spots within an overlay network because many nodes can answer popular queries. CBC solution was able to reduce the maximum number of queries answered by a Chord node from 25,151 to 1,677 (15 times reduction). Similarly, the peak number of queries forwarded by a node was reduced from 27,574 to 5,191 (5.3 times reduction). Thus, the proposed solution also provides good load balancing properties.

7.7 Summary

A sub-overlay formation and a distributed caching solution that adapts according to interest patterns of explicit P2P communities are proposed. It allows queries to be forwarded to community members while enabling them to cache resources that are of interest to their community. An analytical solution is used to determine the best cache placement and capacity allocation strategies and to provide useful bounds on performance. The proposed caching algorithm that utilizes only the local statistics is independent of how the communities are formed and works with any skewed distribution of queries. Overall solution enhances both the communitywide and systems-wide lookup performance, and introduces minimum storage, network, and computational overhead.

Chapter 8

DISTRIBUTED MULTI-SENSOR DATA FUSION OVER NAMED DATA NETWORKS

Named Data Networking (NDN) routes data based on their application-layer content names enabling location independence, in-network caching, multicasting, and enhanced security. We present a proof of concept solution that demonstrates the applicability of NDN for data fusion in Distributed Collaborative Adaptive Sensing (DCAS) systems with multiple end users, applications, and sensors. In this example, a network of weather radars name data based on their geographic location and weather feature (e.g., reflectivity of clouds or wind velocity) independent of the radar(s) that generated them. This enables end users to specify an area of interest for a particular weather feature while being oblivious to the placement of radars and associated computing facilities. Conversely, the DCAS system can use its knowledge about the underlying system to decide the best radar scanning and data processing strategies. Sensor-independent names also enhance the resilience, enable processing data close to the source, and benefit from NDN features such as in-network caching and duplicate query suppression consequently reducing the bandwidth requirements of the DCAS system. The solution is implemented as an overlaid NDN enabling the benefits of both the NDN and overlay networks. Simulation-based analysis using reflectivity data from an actual weather event showed 87% reduction in average bandwidth consumption of radars and 95% reduction in query resolution latency.

Section 8.1 presents the introduction and contribution. Proposed naming convention, overlay construction, query-subscription scheme, and data-generation-time-aware caching policy are presented in Section 8.2. Extensions of the solution to support sensor and event specific queries are discussed in Section 8.3. Simulation setup and performance analysis are presented in Sections 8.4 and 8.5, respectively. Section 8.6 presents the concluding remarks.

8.1 Introduction

Modern Internet users value the ability to access contents irrespective of their locations, whereas the Internet was designed to facilitate end-to-end resource access. Conflict between the usage and design objectives has led to many issues such as location dependence, traffic aggregation, and security. Consequently, many clean-state designs for the Internet propose to access/route data based on their content names [Ja09a, Ko07, St02]. Named Data Networking (NDN) [Ja09a] (a.k.a. Content Centric Networking (CCN)) is gaining traction as one of the viable clean-state designs particularly in the presence of CCNx open source implementation [Palo]. NDN enables in-network caching, multicasting, duplicate message suppression, enhanced security, and mobility. When data are not already dispersed within the network, NDN delivers user queries to potential data sources enabling on demand data generation. In contrast, the majority of other content-naming solutions, e.g., [Ko07, St02], are based on Distributed Hash Tables (DHTs) that index only the pre-generated data. Moreover, NDN supports different levels of abstractions and incremental deployments ranging from overlay networks, content delivery networks, and small ISPs to eventual Internet-wide deployment.

Emerging DCAS systems [Ku06, Le12, Mc05, Mc09] sense the physical world at a far greater spatial and temporal resolution that has not been hitherto possible. These systems rely on a multitude of heterogeneous and distributed sensors ranging from mote based, resource limited, low power, and task-specific wireless sensor nodes to resource rich, high power, and multipurpose sensors such as radars. Typically, DCAS systems deploy redundant sensors to increase the accuracy and resilience. Data generated by these sensors are distributedly fused/processed using groups of computational, storage, and bandwidth resources [Le12]. A key defining characteristic of DCAS systems is the *data pull* where end-user information needs determine how and what group(s) of system resources are utilized to generate the required data [Ku06, Le12]. Thus, DCAS systems have to operate the sensors and computing resources collaboratively, and adapt them to changing conditions in a manner that meets the competing end-user needs.

Collaborative Adaptive Sensing of the Atmosphere (CASA) [Ku06, Mc05, Mc09] is a DCAS system based on a network of weather radars that collaborates and adapt in real time to detect, track, and

forecast hazardous, localized weather phenomena such as tornados and flash floods (see Section 2.2.1). The CASA network consists of a heterogeneous set of weather radars, processing nodes, and data-fusion algorithms that operates collaboratively to detect hazardous atmospheric conditions while satisfying many diverse and conflicting end-user requirements. Distributed and collaborative data fusion provides an attractive implementation choice for real-time radar data fusion in CASA, wherein multiple data volumes are constantly being generated, processed, pushed and pulled among groups of radars, storage, and processing nodes. CASA supports a diverse set of meteorological algorithms (referred to as *applications*) and end users. Table 2.3 lists a subset of the applications that are currently supported by CASA. Each application pulls one or more types of data from one or more radars. For example, radar images that we see on TV newscasts are drawn using reflectivity data from clouds that are typically generated by a radar. More accurate reflectivity images can be generated using the Network-Based Reflectivity Retrieval (NBRR) algorithm that pulls reflectivity data from three or more radars that sense the same region in atmosphere within an acceptable time window [Li07b]. Both Doppler velocity and reflectivity data from two to three radars are needed to estimate the wind velocity accurately. The same data are used for tornado-tracking applications. Therefore, the same data type may be accessed by multiple applications. Applications require different amounts of computational, storage, and bandwidth resources as they use different data types, amounts of data, and meteorological algorithms. Known weather patterns, geography, cost, and availability of the infrastructure determine where the applications are deployed. For example, tornado-tracking applications are deployed only in areas that are likely to have tornados. These applications are accessed by a diverse set of end users (see Table 2.4) such as the National Weather Service (NWS), Emergency Managers (EMs), scientists, media, and commercial entities. Users may issue queries periodically for surveillance purposes or when an interesting weather event is detected within their Area Of Interest (AOI). For example, a NWS forecast office sends a separate query for each of the applications listed in Table 2.3 for counties under their jurisdiction (except for air surveillance). For surveillance purposes, they may pull data from reflectivity and velocity applications every five minutes regardless of the current weather conditions. However, when an active weather event is detected, reflectivity, velocity,

NBRR, nowcasting, and QPE (Quantitative Precipitation Estimation) applications are queried at a higher sampling rate. These queries are periodically issued for the area of active weather (which may change with time) until the weather event subsides or move out of their jurisdiction. A researcher trying to understand the physical properties of a tornado may use velocity and tornado-tracking applications every 30 seconds to acquire samples more frequently. Alternatively, commercial entities may sample their AOIs at a much lower sampling rate, as they are interested in mid to long-term changes in weather. Each CASA radar generates raw data at rates up to 800 Mbps, which reduces to 3.3 Mbps with preprocessing. In some cases, e.g., to preserve the accuracy or for archiving purposes, it is preferable to transfer raw data. The next generation of solid-state CASA radars is expected to generate raw data at several Gbps. Moreover, due to the spatial and temporal locality in weather events, corresponding end-user queries also exhibit high spatial and temporal locality. Therefore, even more bandwidth is needed to handle concurrent queries without increasing the latency. Furthermore, a nationwide CASA radar network deployment in the U.S., a strong possibility given the many advantages of the CASA paradigm, is estimated to require 10,000 radars [Mc09]. These radar nodes are to be interconnected via a combination of wired and wireless networks. However, though the system is mission critical, it is neither economical nor feasible to maintain a separate network or allocate very large bandwidth to radars and processing nodes. Hence, existing networking infrastructure and resources have to be utilized efficiently to achieve the system objectives of maximizing the detection and warning accuracy while reducing the cost.

DCAS systems, including current CASA deployments [Li07a], typically bind data to the sensor(s) that generated them by assigning data names based on the sensor identifier. Conversely, end users in many cases are interested in data related to a particular weather event in a given AOI, and are not concerned with which sensor(s) generated the data. Therefore, naming data based on the sensor creates a conflict similar to that in the current Internet, and reduces the ability to utilize the spatial and temporal locality in user interests and redundant sensors to enhance the performance of the DCAS system.

By naming data based on their geographic location of the weather event or atmospheric condition, data type (e.g., wind velocity and temperature), and/or event name (e.g., hail and tornado), DCAS systems

can gain the benefits of NDN. For example, a CASA end user may specify the query "*get current wind conditions in southwestern Oklahoma*" while being oblivious to the placement of radars and associated computing resources. Such sensor-independent names enable NDN benefits such as in-network caching, duplicate query suppression, on-demand data generation, security, and mobility. Hence, DCAS systems are even better applications for NDN compared to web access, streaming, VoIP [Ja09b], and text-based chat [Palo] that utilize only a subset of the NDN features. Occasionally users may still want to access the sensors using their unique names (e.g., because a radar has a specific capability or a user wants to calibrate/validate a radar). Therefore, it is important to support both the sensor independent and dependent naming conventions within a DCAS system. Currently, NDN has to be deployed as an overlay network due to the absence of an Internet-wide deployment. However, use of overlay networks provides the added benefits such as the ability to deploy application-specific routing mechanisms [Ba13], fault tolerance, better QoS, and in-network data fusion [Ba07a]. Therefore, DCAS systems can be made more efficient and robust by combining the benefits of NDN and overlay networks.

We present a proof of concept multi-user, multi-application, and multi-sensor DCAS system based on CASA that is implemented on an overlaid NDN. A hierarchical naming convention that names the data based on their geographic location and type independent of the sensor(s) that generated them is proposed. Such sensor-independent names enable end users to specify an AOI for a particular event or data type(s), while being oblivious to the placement of sensors and associated computing facilities. Conversely, the DCAS system can use its knowledge about the underlying system to decide the best radar scanning and data processing strategies. Such a design also enhance the resilience, enable processing data close to the source, and NDN benefits such as in-network caching and duplicate query suppression consequently reducing the bandwidth requirements of the DCAS system. An extension is proposed for NDN to support many-to-one data retrieval, as multi-sensor data fusion applications need the ability to retrieve data from multiple sources that match a given name. The overlay network enables geographic-name-based query routing. 2-dimensional version of Content Addressable Network (2D-CAN) (see Section 2.1.2 and [Ra01]) is used as the underlying overlay network, as it provides a direct mapping between the

geographic space and overlay address space while preserving the locality. A subscription mechanism for periodic queries and a caching policy based on the data generation time that is more suitable for DCAS systems are also presented. How the proposed solution can be extended to support sensor and event specific queries is also discussed. Simulation-based analysis is used to evaluate the efficacy of the proposed solution using design parameters from the CASA IP1 test bed [Br07, Mc09] and reflectivity data from an actual weather event. Simulation results show 87% and 95% reduction in average bandwidth consumption of radars and query resolution latency, respectively.

8.2 Multi-Sensor Data Fusion Over NDN

CASA end users typically specify an AOI for a particular weather feature, and are less concerned about the placement of radars and associated computing resources. Hence, it is beneficial to decouple the data, security, and access from the sensors. Such decoupling enables data to be pulled from any available sensor covering the given AOI, and to be processed using any computing node(s) with the desired application. Moreover, such flexibility enables load balancing and resilience, which is essential in mission critical DCAS systems like CASA. Thus, NDN is a good fit for data delivery in DCAS systems as it decouples the identity, security, and access from the end node/sensor while accessing data using content and context aware names. High spatial and temporal locality exhibited by end-user queries can be used to reduce the bandwidth requirements of the overall DCAS system seamlessly by benefiting from caching and duplicate message suppression in NDN. Moreover, receiver-driven communication and on-demand data generation features of NDN are essential in CASA-like systems that dynamically allocate system resources in response to end-user information needs. We have demonstrated overlay networks and peer-to-peer-based distributed data fusion [Le12] are attractive implementation choices for large-scale CASA deployments due to their scalability, flexibility, and reliability. Therefore, by implementing NDN on top of overlay networks, more efficient and robust DCAS systems can be developed. 2D version of CAN provides a direct mapping between the geographic space and overlay address space, and hence can be used to preserve the locality among radars, computing resources, and end users. Moreover, it can forward packets

even in the presence of voids in the physical space (radars and computing resources are not placed uniformly due to terrain conditions and variability in population distribution) while alleviating local minima problem in other greedy routing schemes. It is not necessary for each dimension of the CAN's *d*-dimensional torus (*d*-torus) to be of equal length. We first present the proposed naming convention. Then discuss how the overlay is constructed using CAN and NDN is used to resolve queries. Finally, a subscription scheme for periodic queries and a data-generation-time-aware caching policy are presented.

8.2.1 Naming Data

A query specifies a desired AOI and application. Queries are typically issued periodically, as end users are interested in the most recent data. Therefore, a name in an interest packet sent from an end user to an application needs to include at least the *AOI*, *application*, and *time*. *time* is used to indicate that the user is looking for the most recent data. Based on the hierarchical naming convention recommended by NDN, one of the following formats can be used to specify a name:

1. `/application/AOI/time`
2. `/AOI/application/time`

The first format gives preference to the application and forwards an interest packet looking for an application that can process data for the given AOI. This enables processing data close to the end user as an interest packet stops as soon as a node with the desired application is found. It can be also used when a user is looking for specific radar (by replacing the application name with the radar/sensor name). The second format gives preference to the AOI and forwards an interest packet looking for an application near the given AOI. This enables processing data close to the source. It is more suitable for CASA-like systems as it is desirable to process large volumes of data close to the source and send the fused data across the network, than carrying the data across the network and fusing it at or closer to the destination. For example, a velocity application pulls Doppler velocity and reflectivity data from at least two radars, and hence pulls at least four (2×2) distinct data items for each point in AOI. However, after processing the data, only a single value is sent to the user (for each point in AOI) resulting in at least 4:1 bandwidth reduction. Even

the second name format can locate a specific radar, by replacing the AOI with the radar location and setting application type as *radar*. Given these benefits, we use the second name format for the proposed solution.

AOI is typically specified as a rectangular area (see Fig. 8.1) covering few tens to thousands of square kilometers, depending on the end users' role. As seen in Fig. 8.1, AOI can be specified using the coordinates of the lower-left and upper-right corners. However, a single data packet may not be able to carry the data for a very large AOI. Therefore, AOI needs to be split into a set of smaller tiles (see Fig. 8.1). The smallest tile currently supported by CASA radars is ~100×100 m^2. However, it is not useful to send a separate interest packet for each tile as users are not interested in weather at a very specific point in space and the overhead is high as weather features are typically represented using a four-byte number per tile (e.g., reflectivity in dBz or wind speed in km/hour). Hence, it is desirable to request multiple tiles within an interest packet. For example, a data sample with one data type from a 3×3 km^2 area fits into a 4 KB packet recommended by CCNx [Palo]. Therefore, we format the second name as follows:

`/x1/y1/x2/y2/application/time`

where (x_1, y_1) and (x_2, y_2) are the lower-left and upper-right corners of the set of tiles. These coordinates are typically specified using latitudes and longitudes. Moreover, breaking a large AOI into multiple smaller AOIs also enables better utilization of cached data. For example, though AOI_1 and AOI_2 in Fig. 8.1 do not overlap completely, the subset of tiles that overlap needs to be pulled from radar *R* only once.

Once the interest packet reaches an application, application then needs to find one or more radars that cover the given AOI and capable of producing the desired data types. While multiple radars cover a

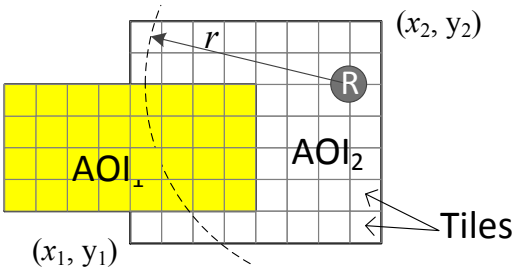


Figure 8.1 – Overlapping areas of interests. *R*– radar, *r* – transmission range of the radar.

given area, some of the radars may be busy scanning different areas requested by other applications. Some radars may not be functional or available at the time for some reason (e.g., radar failure due to severe weather). Therefore, the application needs to identify a subset of the radars that can provide the desired data. CASA radars use a distributed task negotiation mechanism to determine the scan strategy of each radar while increasing the utility of the overall system [An11]. Hence, it is useful to send a subscription message to all the radars covering the given AOI enabling them to negotiate among themselves on which radars will provide the data to the application. Subscription messages can be supported by extending the name format as follows:

$$/x_1/y_1/x_2/y_2/\text{radar}/\text{time}/\text{subscription}/n/\text{dataType}$$

It enables the network to forward an interest packet to all the radars covering the given AOI. Suffix of the name indicates this is a *subscription* request for data from *n* radars for the given *dataType* (e.g., reflectivity or Doppler velocity). Radars that already have the desired data or are willing to generate the data will respond with a data packet indicating their location and a list of tiles that data can be provided for. A radar may not be able to provide data for all the tiles of a given AOI, as its range *r* may not cover the entire AOI (e.g., as in Fig. 8.1) or it may have already committed to generate data for other areas. Instead of passing a list of tile locations to the application, packet sizes can be reduced by sending a *bitmap* indicating for which tiles the data are or will be available. After receiving data packets from radars, application then sends a separate interest packet to pull the data from each selected radar. To reduce the latency, applications may pull data for different tiles from different radars as soon as it receives a data packet from a radar. Following name format is proposed for those interest packets:

$$/x_R/y_R/x_R/y_R/\text{radar}/\text{time}/x_1/y_1/x_2/y_2/\text{bitmap}/\text{dataType}$$

Now the location of radar (x_R, y_R) takes precedence, as we are pulling data from a specific radar. x_R and y_R are indicated in the data packet from the radar. They are repeated to comply with the name format. $/x_1/y_1/x_2/y_2/$ indicates the AOI and *bitmap* indicates the required list of tiles from the given radar.

8.2.2 Overlay Construction and Query Resolution

CASA end users interact with the radar and data fusion network using a set of proxies [Li07d]. Proxies are desirable as not all end users can be part of the overlay due to lack of resources, e.g., mobile devices used by EMs deployed in the field. Proxies can also split a large AOI into multiple smaller AOIs, map a city/county name to its latitudes and longitudes, map a radar name to its location, provide access control, and priority enforcement of end users [Li07a]. While it is not necessary for end users to use NDN to communicate with a proxy, mobile end users such as EMs may use NDN-enabled devices to benefit from mobility features in NDN.

Figure 8.2(a) depicts a 2D CAN overlay formed using a set of radars, applications, and proxies. Scattered lines indicate the zone boundaries and circles with stars indicate the Zone Controllers (ZCs), i.e., nodes responsible for indexing resources that map to their zones. Radars and applications (deployed on computing nodes) are uniquely identified using their geographic location and node type (e.g., *radar* and *reflectivity application*), and indexed at the respective ZC. Therefore, a ZC also maintains a resource index (hash table) in addition to the Content Store (CS,) Pending Interest Table (PIT), and Forwarding Information Base (FIB) maintained by all the nodes. A ZC's FIB is populated with the information about the neighboring zones enabling greedy routing over the 2D-CAN torus. FIBs in other nodes are populated with a default route to their ZC. As the applications and proxies are deployed based on known weather patterns, frequency of use, availability of infrastructure, and cost, they are not uniformly distributed

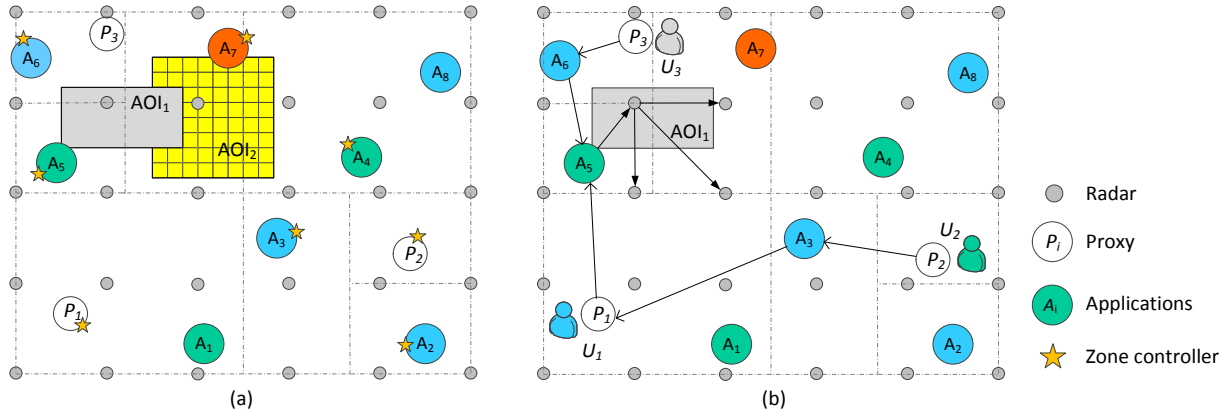


Figure 8.2 – Radar data fusion network: (a) 2D CAN overlay; (b) Interest packet routing. Nodes with the same color belong to the same application type.

throughout the sensor field. Therefore, each application may not be deployed in each zone. However, once an interest packet arrives for a given AOI that overlaps with the zone, the ZC should be able to locate the given application. Hence, neighboring ZCs are configured to share information about the applications that are within their zones. Each ZC needs to know $k \geq 1$ computing nodes (preferably the nearest ones, as it helps to reduce latency) for each application type. Resiliency can be improved when $k > 1$. Sharing information among ZCs does not introduce high overhead as the network is mostly static except for occasional failures in computing nodes. The size of the resource index at a ZC increases with the number of application types, k, and density of radars. However, it will not be very large as there are only a few tens of application types, k does not need to be large, and inter-radar distance is typically 30 km.

Figure 8.2(b) illustrates the forwarding of a set of interest packets from users searching data for application type A_5 and AOI_1. First, user U_1 sends its query to proxy P_1. P_1 then creates an interest packet with a name indicating AOI_1 and A_5 using the second name format. If AOI_1 is very large, it will be split into a smaller set of AOIs and multiple interest packets will be issued. CAN supports routing packets only to a given point within the torus. Therefore, at the overlay level, an interest packet is routed based on the center of the specified AOI. Using CAN's greedy routing scheme the packet is then forward towards the zone covering the center of AOI_1. After reaching the desired zone, ZC's resource index is searched for a computing node(s) capable of running A_5. For example, U_1's interest packet is forwarded from P_1's zone to A_5's zone, which has the application A_5. Similarly, U_2's interest packet follows the path $P_2 \rightarrow A_3 \rightarrow P_1 \rightarrow A_5$. U_3's interest packet is first forwarded from P_3 to its ZC A_6 and then from A_6 to A_5 (path is $P_3 \rightarrow A_6 \rightarrow A_5$). As the paths from U_1 and U_2 overlap at P_1, NDN will suppress one of the interest packets if they are concurrent or will respond to the second interest using P_1's cache (if data are more recent than the given *time*). Therefore, only two interest packets are delivered to the node running A_5.

Once the interest packet reaches A_5, it needs to find a set of radars that cover AOI_1. Only local radars are indexed at the ZC. Therefore, A_5 sends a message to the ZC (in Fig. 8.2 A_5 is a ZC) searching for a radar covering the given AOI. Once a radar is found, a subscription interest packet is sent to that radar. It then broadcasts the interest packet to all the other radars responsible for covering the given AOI. This is

possible because radars already know their neighbors irrespective of the zones they belong to, as it is necessary for the distributed task negotiation mechanism [An11]. We believe this will be the case even in other DCAS systems, as sensors need to know their neighborhood to collaborate. If not, ZCs can be used to keep track of sensors/nodes in the same neighborhood (similar to applications). Routing policies associated with FIB entries are used to indicate that the subscription messages need to be broadcast to all the relevant radars [Ja09a]. Once a subscription interest is received, a radar first checks it CS for the requested data generated after the given *time*. If matching cache entries are not found for one or more tiles in AOI_1, it negotiates with other radars to decide what areas to scan based on the given AOI, data type(s), and required number of radars *n*. If data are already available for a subset of the tiles and/or the radar decides to generate data for the remaining subset of the tiles, it then responds with a data packet indicating the list of tiles (using a *bitmap*) for which the data will be available, data generation time, and its location. Though multiple radars may respond to the subscription interest with a data packet, the current NDN proposal [Ja09a] allows only the first data packet to be delivered to the receiver. However, applications typically need data from $n \geq 1$ distinct radars for each tile in the AOI. Therefore, an array of counters (one for each tile in the AOI) is added to the PIT entries corresponding to subscription interests, enabling a node to accept up to *n* data packets for each tile. This modification does not hinder NDN's ability to overcome denial of service attacks, as *n* is typically small. For example, CASA applications typically do not require data from more than three radars per tile (i.e., $n \leq 3$). Such modifications are easier to integrate into NDN when it is implemented as an overlaid network. Once a data packet with a list of tiles arrives from a radar, application then sends another interest packet to pull data for the tiles that have not already received data from other radars. Another *bitmap* may be used to indicate the required list of tiles. This interest packet is forwarded using the location of the radar, which was indicated in the received data packet. Once data for all the tiles are received from *n* radars, the application processes the data. The processed data are then sent to the user(s) through the reverse path created by PIT entries. The data processing time of an application depends on the complexity of the algorithm and the data types. For example, reflectivity data for different

tiles can be processed independently and concurrently as soon as data arrive whereas NBRR requires data for multiple tiles before processing can begin.

8.2.3 Subscription Scheme for Periodic Queries

End user queries are issued periodically both in the surveillance mode and when an active weather event is detected. Figure 8.3(a) illustrates the query arrival times from a user. As some weather features do not change very rapidly (e.g., reflectivity of clouds and temperature), users are willing to accept data generated few seconds to minutes in the past (denoted by δ) than waiting for the latest data to be generated. δ depends on the weather feature being monitored, end user's role, and time interval γ ($> \delta$) between two successive queries. For example, a scientist querying for wind velocity may set δ = 15 seconds while a user from the media may set δ = 60 seconds. Therefore, a query arriving at time t is willing to accept data generated after $t - \delta$. Hence, *time* in the name of an interest packet is set to $t - \delta$. After the query is issued, the user has to wait some time before the desired data are either pulled from a CS or generated by a radar(s). Let ε denote the *waiting time* which depends on the data generation time from radars, data processing time, network delay, and whether data are already cached. Ideally, $t + \varepsilon < t + \gamma$, as otherwise the next query will arrive before the current query is answered.

However, differences in t, γ, and δ among users connected to the same proxy and access the same data reduce the possibility of benefiting from cached data. Figure 8.3(b) illustrates the query arrival time of two users. Query Q_1 arrives at t_1 and requests data for application type A_1. Suppose application A_1 had to wait until radar R generates the data at t_2. At t_3, A_1 finishes processing the data and sends the results to

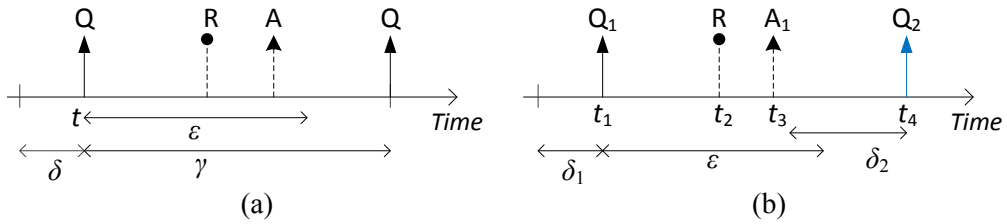


Figure 8.3 – Timing diagram of query arrival (Q), radar data generation (R), and data processing at application (A): (a) Query from a user; (b) Queries from two users.

the user by $t_1 + \varepsilon$. Another query Q_2 arrives at t_4 looking for the same data generated after $t_4 - \delta_2$. However, Q_2 cannot benefit from already cached data, as the data generation time $t_2 < t_4 - \delta_2$. This problem persists even in future rounds of queries. Given both the users' objective is to get data that are more recent for the desired AOI every γ_1 and γ_2 intervals, the problem can be overcome by developing a subscription scheme that can provide data periodically to users without issuing multiple interest packets for the same data at a proxy. A user i subscribes to a proxy indicating the desired AOI_i, A_i, γ_i and δ_i. During the first round, if the cached data are too stale a new interest packet will be issued and new data will be pulled from a radar(s). Then, from the second round onwards a proxy issues only one interest packet for a given application type and AOI. Time to issue the next interest packet t_{next} can be calculated using the current t_{next} (for the first subscription $t_{next} = t_1 + \gamma_1$) and the arrival time t_i of a new subscription from user i with period γ_i. Then $t_{next} = \min(t_{next}, t_i + \gamma_i)$. Once the corresponding data packet arrives, it is forwarded to the relevant end users according to their γ_i and δ_i. This makes sure that no user will receive data any later than they are supposed to receive without the subscription scheme. However, some users may receive data earlier than their next period ($t_i + \alpha\gamma_i$, $\alpha \geq 1$), but they still get a consistent view of the weather event(s) as γ_is are preserved. As the subscription scheme reduces the duplicate interest packets, bandwidth requirement of all the nodes involved in providing data for the given AOI and application is reduced.

8.2.4 Caching Based on Data Generation Time

NDN caching exploits the spatial and temporal locality of user interests to reduce the bandwidth requirements of the network. However, as users are most interested in recent data, data becomes less important with time irrespective of its popularity when it was just generated. Therefore, traditional caching policies such as Least Frequently Used (LFU) and Least Recently Used (LRU) are not effective in deciding what entries to remove from a CS when it is full. Instead, it is beneficial to remove the cache entry corresponding to the oldest data, i.e., one with the earliest data-generation time. We name this caching policy as *Oldest First Caching* (OFC). As the data packets are already tagged with the data-generation time, no additional counters are needed as in LFU and LRU caching. OFC is different from First In First

Out (FIFO) caching as the cached data/entries depend on queries and the oldest data item may not be the first entry to be stored in the CS. For example, because δ_i varies with the application type, user, and γ_i, it is possible for a new query to pull data for a different AOI that is older than the entries already in the CS.

8.3 Supporting Sensor and Event Specific Queries

While queries that specify an AOI are the most common and resource (both bandwidth and computation) consuming types of queries in CASA, end users are also interested in sensor and event specific queries. For example, a user may request data from a specific radar because it may have special sensing capabilities or for calibrating the radar. Event-specific queries look for locations/sensors where a particular weather event is detected, e.g., "*find all locations where hail is detected*" and "*find all locations within my jurisdiction where wind speed is 60 km/h or higher*". Such event-specific queries are typically issued to identify AOIs for developing weather events. For example, locations with rotating wind may be an indication of a developing tornado, and hence could be useful in identifying areas to scan at high frequency and locations to deploy EMs/spotters. While sensor and event specific queries are relatively infrequent in CASA, other DCAS systems may use them in different proportions. Hence, it is important to support multiple query types and naming conventions within the same NDN network. Next, we discuss how the overlaid NDN network in Section 8.2 can be extended to support sensor and event specific queries.

Second name format proposed in Section 8.2.1 can be used to resolve sensor-specific queries by mapping the sensor name to its geographic location and setting the sensor type as the application type. Then the queries can be resolved using CAN's greedy routing, as the radars/sensors are already indexed in the 2D-torus according to their geographic locations. However, a mechanism is needed to map the sensor names to their geographic locations. Such mapping is typically accomplished through a lookup table or a database, as the names typically do not reflect their exact geographic locations, e.g., name of a city or codes assigned to weather stations and radars [Mc09]. Therefore, such mapping can be facilitated by maintaining a copy of the lookup table or database at each proxy node. If it is costly to maintain a separate copy at each proxy, the CAN DHT formed in Section 8.2 can be used to index the sensor names and

their locations. A distributed index in the form of a DHT is more suitable as it enhances the scalability and provides a consistent view of both the static and mobile sensors, and their locations. Mobile sensors may be indexed in the DHT without making them ZCs to prevent their movement from disrupting the overlay topology. Proposed 2D-torus can provide name-to-location mapping within $O(\sqrt{N})$ hops [Ra01], where N is the number of nodes/zones in the overlay.

Event-specific queries are typically expressed as multi-attribute range queries, for example:

$$Q = \begin{Bmatrix} Latitudes \in [30\,\mathrm{N}, 32\,\mathrm{N}],\ Longitudes \in [100\,\mathrm{W}, 103\,\mathrm{W}], \\ WindSpeed \in [60\,\mathrm{kmph}, \mathrm{MAX}],\ Pressure \in [100\,\mathrm{kPa}, \mathrm{MAX}] \end{Bmatrix}$$

where MAX indicates the maximum possible attribute value. Multi-attribute range queries are typically resolved by first indexing the attribute values (i.e., sensor readings) in a centralized database, random set of overlay nodes, or DHT, and then issuing queries to those nodes (Chapter 4). DHT-based solutions are more appropriate for CASA-like systems because of their scalability and some guarantees in performance. DHT-based solutions use either one or more overlay rings or a *d*-torus to index attributes (Sections 2.3.2 and 4.5.3). Adopting a ring-like overlay requires building another overlay within the CASA network while a *d*-torus requires extending the proposed 2D-torus to multiple dimensions. Moreover, ring-like overlays have a much higher advertising cost when attribute values (i.e., sensor readings) change frequently (see Sections 4.5.3 and 4.7). *d*-torus has a much higher query cost when applied to real-world queries, as they tend to specify only a subset of the attributes and large ranges of attribute values (Section 4.7). For example, resolving query Q on a *d*-torus that indexes readings from $d > 4$ sensor types will require searching a very large volume of the torus. Moreover, in practice, multi-attribute range queries are resolved by either mapping the *d*-torus to a ring or by visiting each zone that overlaps with the query volume in parallel and then sending separate answers to the query originator (Section 2.3). Mapping the *d*-torus to an overlay ring not only requires another overlay but it also breaks the locality of sensor reading as *d* increases. Consequently, the query cost increases. Separate answers have to be sent to the query originator, as it is not straightforward to aggregate answers from zones of neighbors and neighbors of neighbors on a *d*-torus. This is not possible in NDN as an interest packet is expected to receive only one data

packet. The mechanism proposed to extend PIT entries to support retrieving data from n radars (in Section 8.2.2) cannot be applied, as the number of query responses tend to be very large (because the search volume is large) and cannot be known a priory (as ZCs know only their neighbors). Next, we propose a mechanism to index sensor readings using the proposed 2D-torus.

Suppose sensor readings are specified using a set of attributes $\mathbf{A} = \{a_1, a_2, a_3, \ldots, a_m\}$. We index sensor readings in the 2D-torus by applying Locality Preserving Hash (LPH) functions [Ca04] to each pair of attributes (a_1, a_2), (a_3, a_4), …, (a_{m-1}, a_m) such that the m-dimensional space is mapped to at most $\lceil m/2 \rceil$ points in the 2D-torus. This is more efficient than mapping attribute values to m points in an overlay ring (see Sections 4.5.3 and 4.7). LPH hash functions are used as they preserve the locality along each pair of dimensions. For example, consider a sensor node s that may have readings for only a subset of the attributes in **A**, e.g., $s = (a_1 = v_1, a_2 = v_2, a_k = v_k)$, where $a_1, a_2, a_k \in \mathbf{A}$. Then its readings are indexed in the 2D-torus based on LPH values of $(a_1, a_2) = p_1$ and $(a_k, a_{rnd}) = p_2$ (see Fig. 8.4(a)). a_{rnd} is a random or a fixed value added to make a pair of attributes. Depending on the application, a random value may be used to spread the points on the 2D-torus providing load balancing while a fixed value may be used to indicate that no such sensor exists. Similar to [Ca04], all the sensor readings of s are advertised to each point such

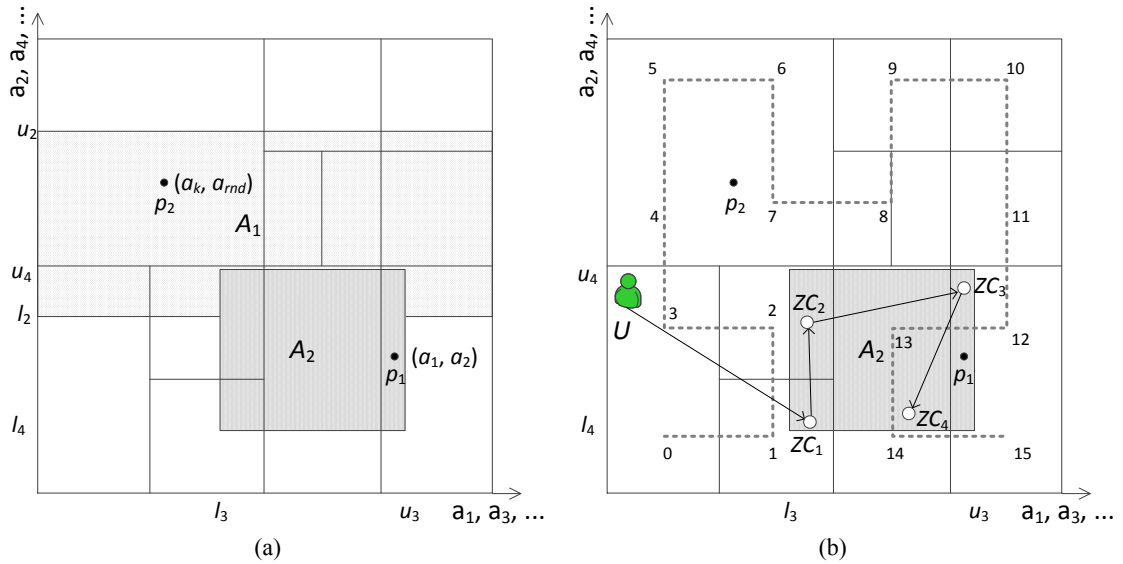


Figure 8.4 – Use of 2D-torus to index sensor readings and resolve range queries: (a) Indexing sensor readings and query areas; (b) Query resolution using a space-filling curve.

that a query can be resolved by searching only one of the points. Sensor readings are re-advertised when their values change (a fixed or dynamic threshold may be applied to reduce the number of advertisements).

Suppose we are given an event query $q = \{a_2 \in [l_2, u_2], a_3 \in [l_3, u_3], a_4 \in [l_4, u_4]\}$, where l_i and u_i are lower and upper bounds of the desired attribute values. Similar to the sensor readings, LPH functions can be applied to each pair of attribute value ranges to determine the search areas (see A_1 or A_2 in Fig. 8.4(a)). The range of attribute values for the unspecified attribute a_1 have to be set as $a_1 \in [\text{MIN}_1, \text{MAX}_1]$, where MIN_i and MAX_i are minimum and maximum values of a_i. We now need to search only one of the two areas (A_1 or A_2) as the sensor readings are mapped to a 2D-torus. We issue query q (as an interest packet) to the smallest area A_2 as it reduces the search space and consequently the cost of resolving the query. Therefore, search space of our solution is much smaller than the search space/volume in a d-torus, which dramatically increases due to unspecified attributes. To overcome the problem of having to forward an interest packet to all the neighboring zones that overlap with the query area and generating separate data packets, we use a Space-Filling Curve (SFC) to determine the order to visit the overlapping zones in the 2D-torus. In [Ba82], it is shown that SFCs can be used to find a computationally efficient, close-to-optimal shortest path route for a given set of points on a 2D area. We use a Hilbert SFC because of its ability to maintain the best locality in a 2D area [Mo01]. Given the number of attributes/dimensions, MIN_i and MAX_i, and resolution along a dimension, any node can calculate the set of points along the SFC that needs to be visited to search the query area. Then the event-specific queries are resolved as follows. The query initiator first identifies the smallest area to query. It then uses the SFC to identify the first point to visit on the 2D-torus that overlaps with the minimum query area. Then an interest packet is generated using the second name format proposed in Section 8.2.2 with the AOI set to the location of the first point. Actual query is appended to the end of the name. Interest packet is then forwarded to the ZC that covers the first point using CAN's greedy routing. For example, in Fig. 8.4(b), the interest packet is forwarded from user U to ZC_1. Once it reaches the first point, ZC calculates the second point to visit and forwards the packet towards that point. A ZC needs to be visited only once even though it may cover multiple

points to be visited (e.g., ZC_3 in Fig. 8.4(b)). Once the interest packet reaches the ZC that covers the last point (e.g., ZC_4 in Fig. 8.4(b)), the query is resolved by searching the ZC's index for matching sensor readings. Then a data packet is generated and the results are forwarded through the reverse path (e.g., $ZC_4 \rightarrow ZC_3 \rightarrow ZC_2 \rightarrow ZC_1 \rightarrow U$). While the data packet is being forwarded towards the query originator, intermediate ZCs are also queried and matching sensor readings are appended to the data packet. Therefore, only one data packet will be received for an interest packet. Therefore, we can also support event-specific queries within the same solution without creating another overlay while overcoming issues in the ring and d-torus based designs.

8.4 Simulation Setup

CASA IP1 test bed [Br07, Mc09] in Oklahoma had only four radars. Radars are currently being relocated to Dallas, TX where it will be expanded into an eight-radar network. Therefore, to demonstrate a much larger radar network, a discrete-event simulator is developed using parameters from the IP1 test bed to reflect real-world deployment scenarios. We consider a sensor field covered by 121 radars placed on an 11×11 grid with an inter-radar distance of 30 km. This enables us to focus on the inner 300×300 km^2 area that is covered by multiple radars while discarding the border effects. Sensing range r of a radar is set to 40 km. The size of a tile is set to 100×100 m^2 and four bytes of data were generated per data type per tile (after preprocessing). Largest AOI specified in an interest packet is set to 6×6 km^2 as end users are not interested in weather related to a very small area and it reduces the overhead per interest packet. Radars are unsynchronized and generate data for a 360^o scan every 30 seconds. Radars know other radars that have overlapping coverage (within 2r of each other). For management and administrative purposes, radars are grouped into a set of Data Fusion Groups (DFG) (see Fig. 8.5). A DFG typically contains nine (3×3) radars hence the given sensor field has 16 DFGs. Reflectivity and velocity applications are deployed randomly within each DFG (one per DFG) as they are frequently used. NBRR, nowcasting, and QPE applications are deployed only in four randomly selected DFGs, as they are used only when an active weather event is detected and deployed based on known weather and usage patterns. Applications

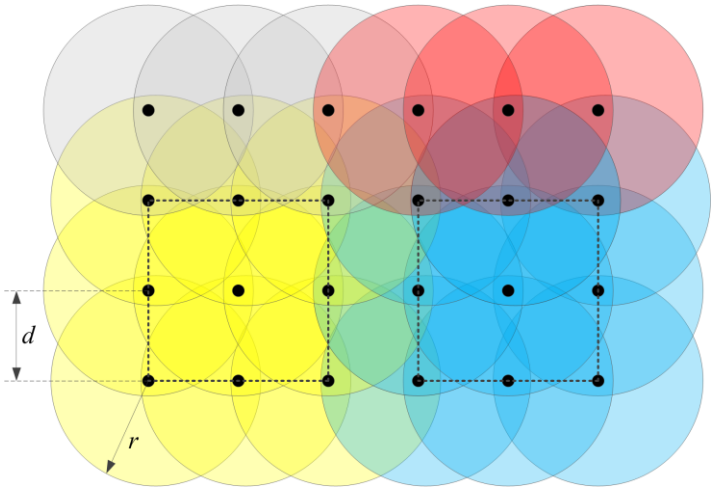


Figure 8.5 – Data fusion groups for network of radars. *d* – inter-radar distance, *r* – transmission range.

are deployed randomly within each selected DFG. Altogether 44 computing nodes (16×2 for reflectivity and velocity, and 4×3 for NBRR, nowcasting, and QPE) are deployed within the sensor field. Five proxies are placed randomly within the sensor field. The overlay network consists of these 49 (44 + 5) nodes. Radars are not used to forward overlay messages as they may have limited bandwidth due to deployments in areas without well-established infrastructure. However, they are indexed at respective ZCs enabling subscription interest packets to find them. ZCs share information about applications and each ZC knows one node for each application type (i.e., $k = 1$). Link bandwidth is set to 1 Gbps. Size of CS is varied from zero to 100 MB in 25 MB increments. PIT entries expire after 120 seconds to prevent the PIT from becoming arbitrarily large.

An area of 300×300 km^2 is typically covered by two NWS forecast offices and 30 EMs, these numbers are derived using the number of NWS forecast offices in the U.S. [NWS] and EMs deployed in a county/state. We assume each user is responsible for a distinct fragment of the sensor field (i.e., AOI), and therefore assigned ½ (2x1) of the sensor field to each NWS office and $^1/_{30}$ (6×5) of the sensor field to each EM. We further assume eight scientists and 20 users from the media are also interested in retrieving data from the entire sensor field in the surveillance mode as described in Table 2.4. Altogether, there are 60 users. Queries for NBRR, nowcasting, and QPE are issued only when an active weather event is detected. Therefore, to determine AOI to specify for those queries, we use reflectivity data from an actual

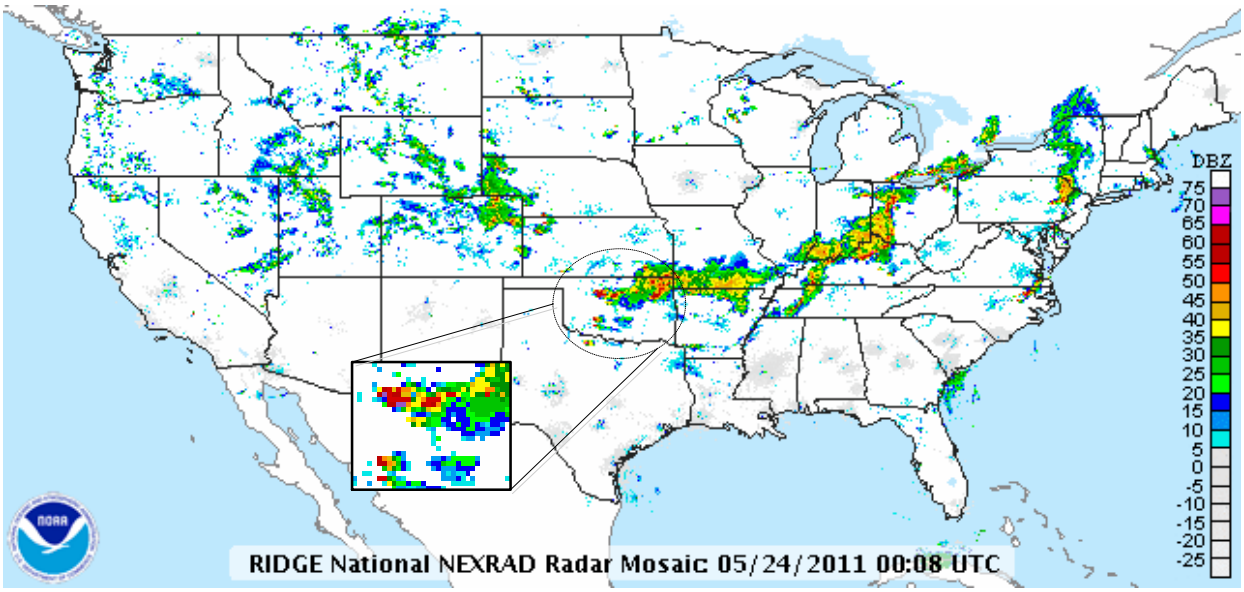


Figure 8.6 – Reflectivity data from a severe weather event over Oklahoma, U.S. on 24/05/2011 at 00:08 UTC [NOAAa].

weather event over Oklahoma between 23/05/2011 20:00 UTC and 24/05/2011 02:00 UTC (see Fig. 8.6, data obtained from [NOAAa]). This weather event led to several violent (EF4) tornadoes within the IP1test bed. Minimum bounding rectangles covering areas with reflectivity over 25 dBz are considered of interest and the AOIs are updated as the weather event migrated. Reflectivity images from [NOAAa] were available only every one hour. Hence, midpoints between two successive samples were used to adjust the AOIs every half an hour as shown in Fig. 8.7.

To evaluate the solution for event-specific queries, we considered a 1,000×1,000 km^2 area covering parts of Arkansas, Kansas, Missouri, Oklahoma, and Texas (between latitudes 32^0 N and 41^0 N and longitudes 92.38^0 W and 103^0 W). We used data from 1,081 weather stations within this area that reported humidity, pressure, temperature, wind direction, and/or wind speed. A three-day trace of sensor readings starting from 2012/06/29 00:00 GMT was collected from [NOAAb]. Sampling interval of weather stations varied between five minutes to one hour. Fixed thresholds were applied to sensor readings to reduce the advertisements due to minor variations in sensor readings (see Appendix II.5). However, at least one advertisement/update was sent every 30 minutes or one hour depending on the sampling interval of the weather station. 10% of the weather stations were randomly added to the CAN overlay as ZCs and other

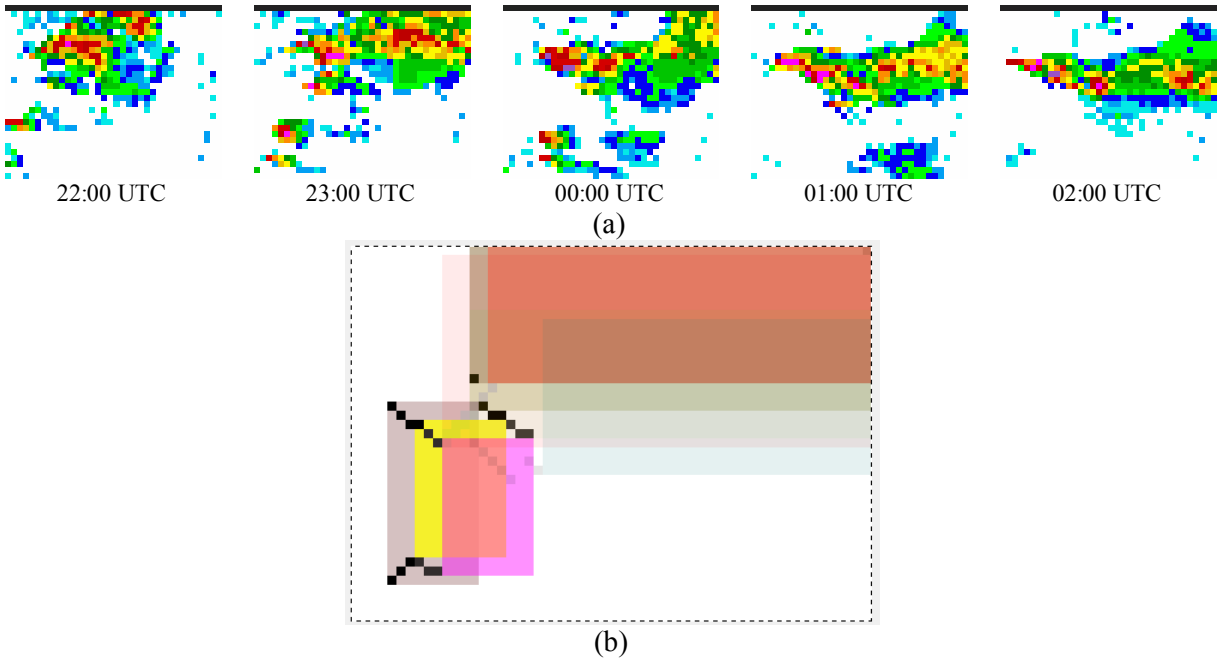


Figure 8.7 – Use of reflectivity data to define AOIs: (a) Change in reflectivity with time; (b) Use of midpoints to define AOIs.

stations were directly connected to them. We simulated 500 users (30 NSW, 320 EMs, 30 scientists, and 120 media) where each issued queries based on an exponential distribution with an inter-arrival time of 300 seconds. As the composition of queries is unknown, we generated random queries by selecting one or more attributes and range of attribute values as specified in [Ca04]. Each query also specified end user's AOI. Hilbert SFC was generated using the algorithm in [Moor]. Results are based on five samples with different random seeds. Additional details on simulators are given in Appendix II.5.

8.5 Performance Analysis

Sensors in a DCAS system are typically the bottleneck in terms of the bandwidth, as they may not be connected by high-bandwidth links due to lack of infrastructure in places where they are deployed. Therefore, in Fig. 8.8, we analyze the amount of data pulled from a radar within a given time period under different cache capacities and policies. Error bars indicate 1th and 99th percentile. Due to the periodic but asynchronous arrival of queries from different end users, it is more appropriate to evaluate the aggregated amount of data pulled within a time window than instantaneous values. Proposed subscription scheme for

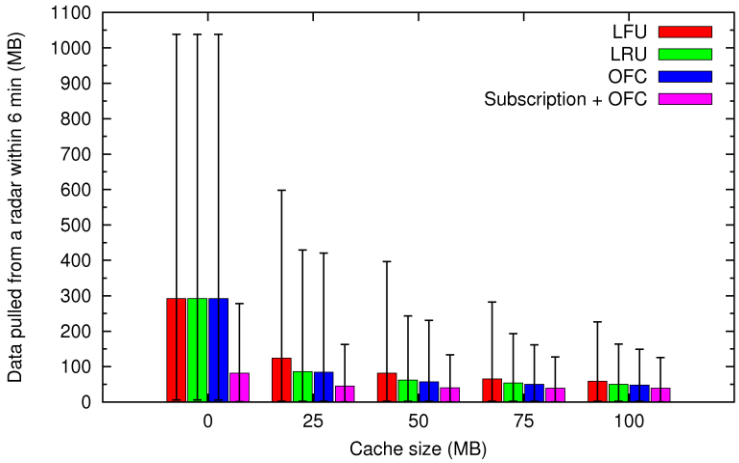


Figure 8.8 – Data pulled from a radar while varying the cache size.

queries reduce the peak and average load on a radar by 61% and 72%, respectively, even without any caching (size of CS CS_{size} = 0). Peak and average loads further reduced to 87% and 86% with CS_{size} = 25 MB and diminishing return is gained with increasing CS_{size} (e.g., load reduced by another 1% when CS_{size} = 75 MB). Among the three caching polices considered, *OFC* outperformed *LFU* and *LRU*. For example, when CS_{size} = 100 MB *OFC* reduced the peak load by 85% (without the subscription scheme) while *LFU* and *LRU* reduce it by 76% and 82%, respectively. Load reduction due to both the *OFC* and *LRU* caching policies indicates that data generation and access times are better predictors of future data-access patterns when users are interested in more recent data. However, compared to *LFU* and *LRU*, *OFC* caching has the added benefit of not requiring any counters as the data packets are already tagged with the data-generation time. Figure 8.9 shows the cumulative distribution of load on radars. Without any caching or subscription scheme, large volumes of data have to be pulled from the radars that are responsible for covering the area of active weather. While caching reduces the load on radars, a more balanced distribution of load can be achieved when the subscription scheme is combined with caching. It was also observed that the total load on radars could be reduced by 28% by applying only the duplicate messages suppression feature in the PIT. Figure 8.10 also shows similar load reductions for applications and proxies.

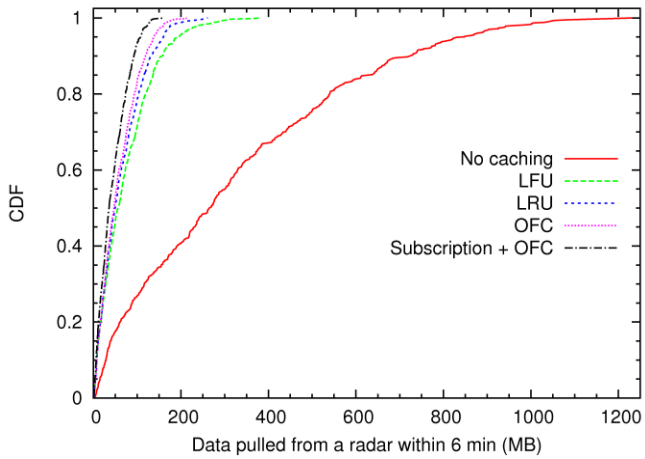


Figure 8.9 – Amount of data pulled from radars. Cache size 75 MB.

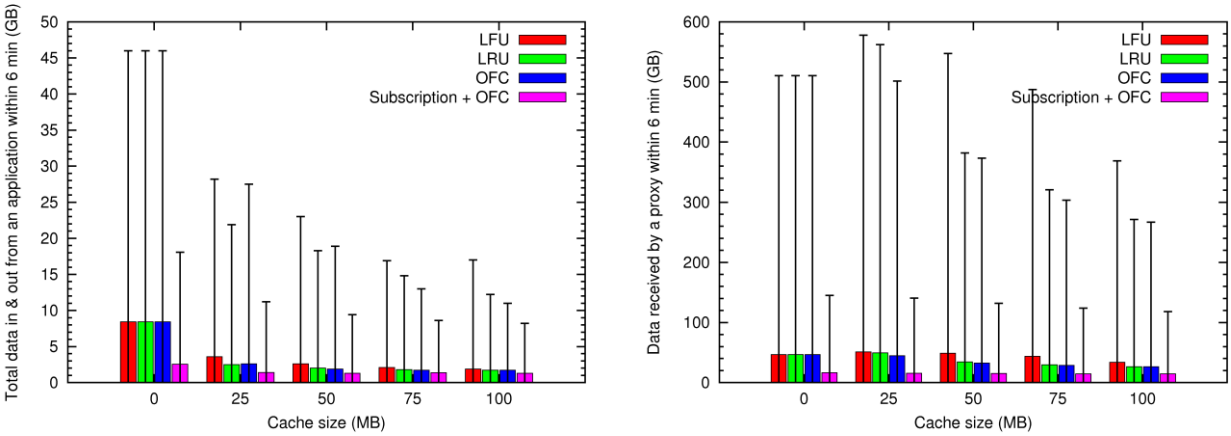


Figure 8.10 – Data pulled from: (a) Applications; (b) Proxies.

Alternatively, end users are concerned about the quality of the received data. Figure 8.11 shows the waiting time ε under different caching policies and CS_{size}. When $CS_{size} = 0$ multiple interest packets go all the way up to the radars while aggregating traffic along the path. This behavior is confirmed by Fig. 8.12 which shows that the number of overlay hops traveled by an interest packet without caching is 73% higher than when $CS_{size} = 100$ MB. Consequently, ε increases, as more time is required to send the corresponding data packets back to the end users over multiple hops under a bounded link bandwidth. It was also observed that 3% of the queries got lost due to PIT timeout as some data packets were significantly delayed. The subscription scheme reduced average ε by 72% without caching. Even though it pulls the latest data from radars, packets are not significantly delayed as the links are less congested. When caching

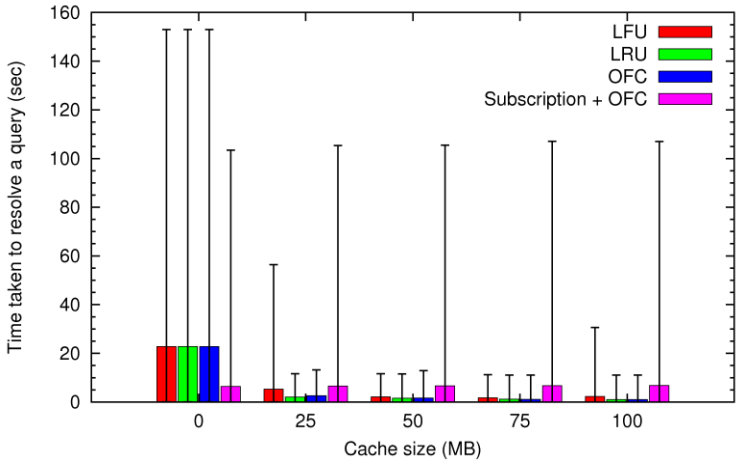


Figure 8.11 – Time taken to resolve a query (waiting time ε).

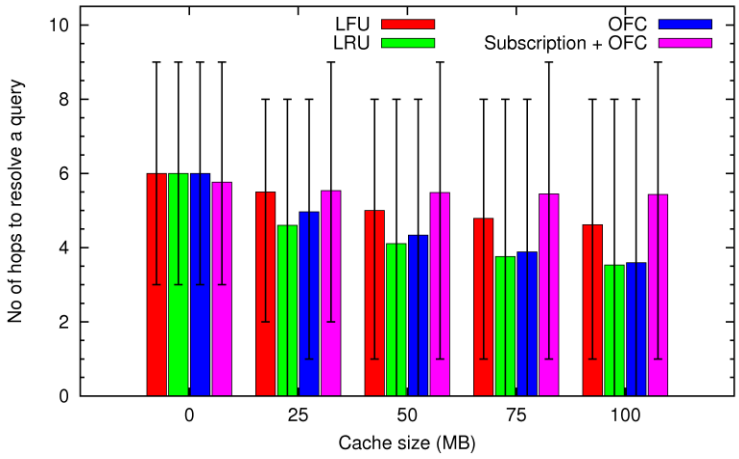


Figure 8.12 – Number of overlay hops travelled by interest packets.

is employed, only the interest packets looking for the latest data are forwarded to the radars while other interest packets are answered from CSs along the path. Therefore, most users receive data within a short time consequently reducing ε. For example, when CS_{size} = 75 MB peak and average ε reduce by 88% and 95% using both *OFC* and *LRU* caching. As the subscription scheme is looking for recent data, it cannot significantly benefit from caching. Hence, ε did not reduce with increasing CS_{size}. Staleness (i.e., the time between the data generation and delivery to users) is another metric of quality as end users are interested in getting more recent data. Figure 8.13 shows the staleness. Subscription scheme and caching reduce the duplicate interest packets in the network while making overlay links less congested. Hence, data packets can be delivered with a lower delay consequently reducing the staleness. Therefore, by combing DCAS

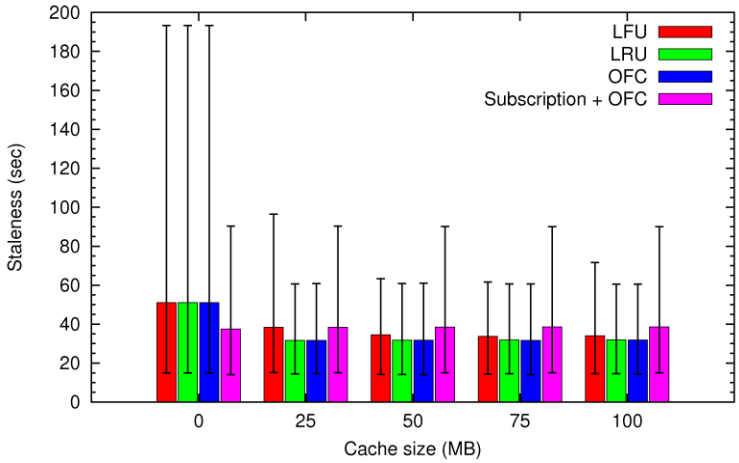
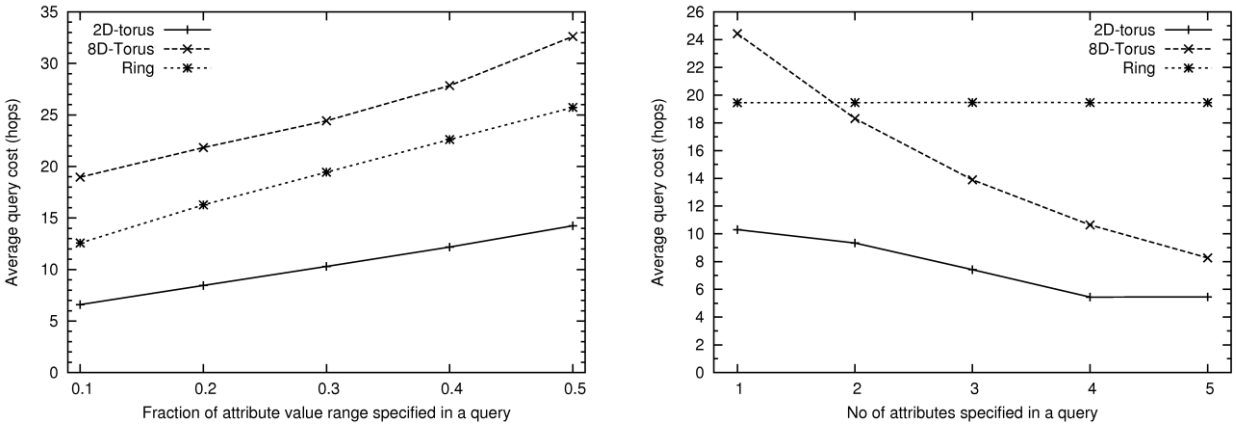


Figure 8.13 – Staleness of received data.

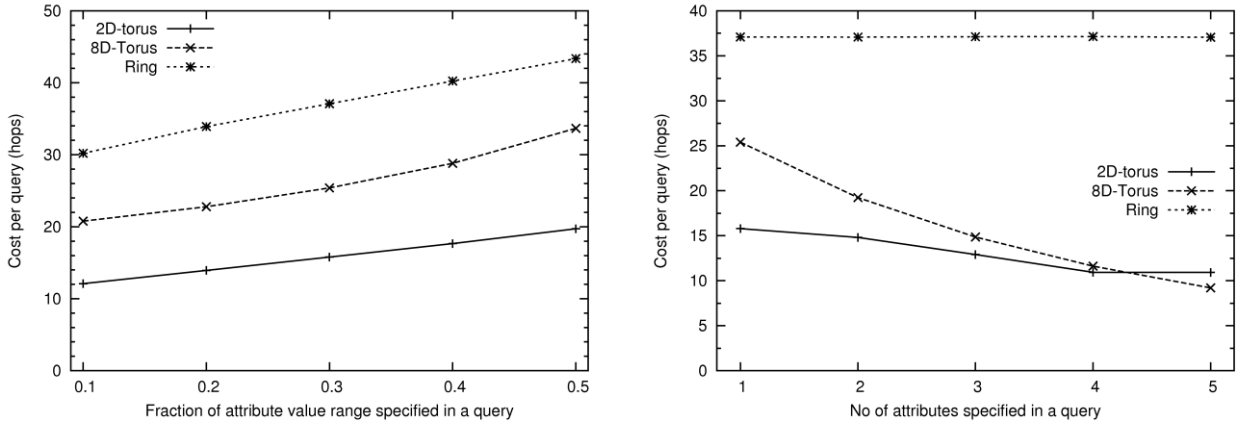
systems with NDN and overlay networks, the bandwidth requirements of the sensors and DCAS system can be reduced while increasing the quality of data delivered to end users.

Next, we analyze the performance of event-specific query resolution using the 2D-torus. Performance is also compared against an 8D-torus (five dimensions for sensor readings and three dimensions for latitude, longitude, and elevation of a weather station) and a ring-like overlay based on [Ca04]. Figure 8.14(a) shows the average number of hops required to resolve a query while increasing the range/fraction of attribute values specified in a query. While the query cost increases with the increasing range of attribute values (as the search space increases), the cost of our solution is 56-65% lower than the 8D-torus and 45-48% lower than the ring. Query cost with increasing number of attributes is shown in Fig. 8.14(b). Query cost of 2D-torus and 8D-torus decreases as the area/volume to search gets smaller when the queries specify more and more attributes. The performance of the ring does not change as the fraction of attribute value range specified in a query was fixed. While the query cost of 8D-torus is gradually approaching our solution, our solution is still 34% lower than the 8D-torus even when five attributes are specified. Figure 8.15 shows the total cost per query (considering both the cost of advertising sensor readings and queries). It can be seen that the 2D-torus is 38-42% and 55-60% more efficient than the 8D-torus and ring, respectively. In practice, queries are more likely to specify a small number of attributes and large range of attribute values (Section 4.4.2). Therefore, the proposed solution is more suitable as it has the lowest cost.



(a) (b)

Figure 8.14 – Query cost with: (a) Varying attribute value ranges. No of attributes = 1; (b) Increasing number of attributes. Fraction of range = 0.3.



(a) (b)

Figure 8.15 – Per query cost with: (a) Varying attribute value ranges. No of attributes = 1; and (b) Increasing number of attributes. Fraction of range = 0.3.

8.6 Summary

A multi-user, multi-application, and multi-sensor DCAS system implemented on top of an overlaid NDN was presented. Ability to name the data based on the geographic location and data type, independent of the sensors that generate them, enables not only bandwidth reduction and load balancing but also increases the quality of data delivered to the end users by reducing the response time and staleness. Proposed subscription scheme and data-generation-time-aware caching policy further reduced the bandwidth requirements, waiting time, and staleness of received data. Resolving event-specific queries using

the proposed 2D-torus is more efficient than building a separate ring-like overlay or a multidimensional torus. While the performance gain is specific to the simulated CASA network, we believe these performance gains are significant enough to justify the applicability of NDN for other DCAS systems.

Chapter 9

SUMMARY

We envision collaborative Peer-to-Peer (P2P) applications that will look for groups of diverse peers that could bring in unique resources and capabilities to a virtual community thereby empowering it to engage in greater tasks beyond what can be accomplished by individual peers. The majority of existing solutions focuses only on discovering individual resources. Moreover, all the solutions rely on many simplifying assumptions due to the absence of data and understanding of the characteristics of real workloads. We bridged this gap by presenting a characterization of real-world resources and queries, and then used the learned behavior to develop a synthetic resource and query generation tool, resource and query aware Resource Discovery (RD) solution, community-aware distributed caching solution, and demonstrated the applicability of Named Data Networking (NDN) for Distributed Collaborative Adaptive Sensing (DCAS) systems. This chapter provides a concluding summary of work presented (Section 9.1) and future directions (Section 9.2).

9.1 Conclusions

In Chapter 4, we analyzed the resource and query characteristics from four different real-world computing environments (1) PlanetLab networking test bed, (2) SETI@home volunteer-computing system, (3) EGI grid computing nodes, and (4) a distributed campus computing facility. Fundamental design choices for P2P-based RD were then qualitatively and quantitatively evaluated using the learned behavior. To our knowledge, this is the first such evaluation using real-life workloads. Findings show real world, multi-attribute resource and query characteristics diverge substantially from conventional assumptions. While real world, less-specific queries are relatively easier to resolve, they introduce significant load balancing issues due to skewed resources and queries. Dynamic attributes contribute to high advertising cost,

and their behavior is attribute-type and system specific hence should not be ignored in performance studies. These findings indicate the need for more efficient, scalable, and robust RD solutions as well as the importance of taking into account the specific characteristics of real-world resources/queries while designing and analyzing such solutions. Hybrid approaches that combine the desirable features of centralized, superpeer, and ring-based architectures have the potential to provide better solutions.

A technique to generate vectors of static attributes and multivariate time series of dynamic attributes while preserving the correlations and temporal patterns observed in operational systems was presented in Chapter 5. A probabilistic finite state machine based technique was also presented to generate multi-attribute range queries. Such synthetic traces of multi-attribute resources and range queries are useful in collaborative P2P and grid/cloud computing for evaluating the scalability of applications, RD solutions, and job schedulers, far beyond that is possible with existing test beds. Its applicability to the four datasets collected in Chapter 4 was demonstrated using statistical tests. Data from any other platform may be used as the basis for trace statistics. A tool is developed to automate the process of data generation and it is made public with the four datasets [CNRL] enabling users to generate synthetic traces using existing datasets or their own ones. Synthetic data from the tool were used to evaluate the fundamental design choices for P2P-based RD in Chapter 4 and the performance of resource and query aware RD solution in Chapter 6.

In Chapter 6, we presented five heuristics to discover multi-attribute resources within a P2P system while alleviating the load-balancing problem identified in Chapter 4. These heuristics were derived based on the properties of resources and queries learned in Chapter 4. Heuristics rely on local statistics to capture the complex characteristics of real-world resources and queries, and try to retain only the nodes that answer queries in the overlay. It is the first RD solution to explicitly take into account the characteristics of real-world resources and queries during the design, runtime, and performance analysis. Resource index is transferred among existing and new nodes to maintain the index size and query load of a node within its capacity. While the heuristics can be executed independently, much better performance can be gained when all five heuristics are executed in the given order. Simulation-based analysis demonstrated

their ability to reduce the query cost, balance the load, and adapt to rapid changes in attribute values. This solution is useful for many systems such as CASA, P2P Clouds, GENI, as well as grid and cloud computing.

Analysis of search clouds from several BitTorrent communities showed P2P communities tend to access the same content hence communities are partially isolated. Furthermore, a survey was conducted to find out the number of communities accessed by BitTorrent users and their frequencies. Our findings show users prefer to access contents from a few primary communities where 89% of the time they accessed at most two communities. These findings were utilized in Chapter 7 to develop a community-aware sub-overlay formation and a distributed caching solution that adapts according to the interest patterns of explicit P2P communities. An analytical model is derived to determine the best cache placement and capacity allocation strategies as well as to provide useful bounds on performance. The proposed caching algorithm is independent of how the communities are formed, utilizes only the local statistics, and works with any skewed distribution of queries. Moreover, the overall solution is adaptive and introduces minimum storage, network, and computational overhead. It is more suitable when users primarily access resources from few communities and when the size of a community is moderate to large with respect to the size of the overall P2P system. Simulations based on Chord overlay, for example, showed 40% reduction in overall query cost (i.e., average path length) with per node cache sizes as low as 20. Less popular communities were able to reduce the query cost by three times compared to system-wide caching. To our knowledge, this is the first caching solution for structured P2P systems that exploits communities to provide better communitywide and system-wide lookup performance. We also demonstrated that query-path-length information is not essential to develop a close to optimal, local-knowledge-based distributed caching solution. The relationships between the asymmetric overlay routing tree and Chord's path length bounded of $O(log_2 N)$, average path length of $\frac{1}{2} \log_2 N$, and bell-shaped distribution of path lengths were also demonstrated.

A proof-of-concept multi-user, multi-application, and multi-sensor DCAS system implemented on top of an overlaid NDN was presented in Chapter 8. A subscription mechanism for periodic queries

and a caching policy based on data generation time that is more suitable for DCAS systems are also presented. The proposed solution also supports sensor-specific and event-specific queries. The ability to name the data based on the geographic location and data type, independent of sensor(s) that generate them, enabled not only bandwidth reduction and load balancing but also increased the quality of data delivered to end users by reducing the response time and staleness. For example, simulation-based analysis using design parameters from the CASA IP1 test bed and reflectivity data from an actual weather event showed 87% and 95% reduction in average bandwidth consumption of radars and latency, respectively. While the performance gain is specific to the simulated CASA network, we believe these performance gains are significant enough to justify the applicability of NDN for other DCAS systems. Moreover, DCAS systems can benefit from multiple features of NDN such as caching, multicasting, duplicate message suppression, ability to deliver interest messages to potential data sources, and enhanced security and mobility hence are even better applications for NDN compared to applications such as web access, streaming, VoIP (Voice over IP) [Ja09b], and text-based chat [Palo] that utilize only a subset of the NDN features. To our knowledge, this is the first demonstration of the applicability of NDN for DCAS systems.

9.2 Future Directions

Presented work can be extended along several directions to further enhance and realize the true potential of collaborative P2P systems, community caching, and multi-sensor data fusion in DCAS systems using NDN. Below we discuss some of the possible future research directions.

Extend resource and query aware resource discovery solution to support all key phases of resource aggregation

Our resource and query aware RD solution (Chapter 6) currently supports only the resource advertising and selecting phases hence needs to be extended to support resource matching and binding phases (Section 3.4). It makes use of a hybrid design that combines a ring-based Distributed Hash Table (DHT) and a superpeer-like two-layer overlay where nodes that are not in the overlay ring advertise and query for resources through the nodes that are in the ring. Superpeers are more suitable for keeping track

of inter-resource relationships of multiple nodes/resources that are directly connected to them (Section 4.5.3, [Ba12b]). They can also provide resource binding on behalf of those connected nodes/resources. Therefore, our hybrid design enables resource matching and binding. However, it is challenging to capture complex inter-resource relationships accurately while introducing low overhead. Constraints such as latency and bandwidth may need to be satisfied among the selected resources as well as between the set of selected resources and the node that is trying to deploy the collaborative application. Satisfying such constraints is nontrivial in superpeer and DHT-based solutions as third-party nodes resolve the queries. Latency or bandwidth measured to a third-party node is not transitive to the selected resources or to the node trying to deploy the application. However, it has been demonstrated that network coordinates [Da04, Le07], measuring performance to a set of landmarks [Za05], and random IP address sampling [Be06] could be used to estimate inter-resource latency without involving the third-party node. However, further research efforts are needed to enhance their accuracy, reduce overhead, as well as to support other inter-resource relationships such as bandwidth, packet loss, and connectivity. Solutions such as P4P [Xi08a] and ALTO [Se09] could be useful in inferring the physical topology and connectivity. Another important extension is to support resource compensation [Ba12b]. For example, distributed data fusion in CASA can compensate for lack of bandwidth between a processing and a storage node by processing data faster (due to inherent parallelism in data fusion) to accommodate the extra delay introduced while transferring data to the storage node. It is useful to identify and support such application-specific compensations within the resource aggregation solution as they can enhance the performance, quality of service, and reliability. Furthermore, solutions designed to support matching and binding phases should take into account the complex characteristics of real-world resources/queries. It is also important to develop analytical models to predict the performance of these solutions. A solution that supports all key phases of resource collaboration can harness the collective power of P2P communities and their underutilized/unused resources to build a globally distributed, virtual datacenter (with computing, storage, and sensing resources) that are useful for limitless number of applications that can yield grater benefits to its contributors/users.

Identify semantic communities and extended performance analysis of community-based caching

Our sub-overlay formation scheme in Chapter 7 directly supports communities based on geographic and organizational interests. It is also important to develop mechanisms to identify communities exhibiting semantic relationships within the P2P system. Our Community-Based Caching (CBC) solution is designed based on the observation in Table 7.1 where real-world P2P communities are neither highly correlated nor completely independent. It is important to investigate the range of correlations for which our solution outperforms other solutions such as clustering which assumes communities are highly correlated and almost all the queries stays within a cluster. For such a comparison, it is important to first extend solutions such as Magnet [Gi10] and/or developing new mechanisms to group nodes into communities based on their semantic relationships. Magnet clusters similar peers to adjacent addresses in the overlay ring and dissimilar peers tend to be well separated. Therefore, prefix bits of the overlay addresses could be used to represent the community identifiers, as nodes mapped to nearby locations tend have the same prefix bits. Once such a mechanism is developed, performance of our CBC solution need be compared against existing semantic-based clustering solutions to determine the applicability of each solution under varying levels of correlations among P2P communities. Once the sub-overlays are created, it may not be necessary to run the same greedy algorithms used by structured P2P systems to forward messages between two community members. Therefore, it is useful to evaluate the possibility of using alternative and/or multiple routing mechanisms within sub-overlays that are more efficient and less complicated. While our solution supports nodes that belong to multiple communities (Section 7.6), our analysis was limited to a single community per node. Therefore, an extensive performance analysis in the presence of multiple communities per node is also necessary. When a node belongs to multiple communities, it may receive a disproportionate number of queries for each of its communities. Thus, it is important to balance the query load while making sure less-dominant communities do not starve due to the dominant ones. While our results show caching also enhances load balancing, further analysis is needed to determine the best cache-capacity allocation strategy within a node when it belongs to multiple communities. It is also of interest to analyze the performance when members of a community have heterogeneous content access

and popularity patterns. Performance of the current solution and all the enhancements need to be evaluated under peer churn as well. The concept of exploiting specific characteristics of communities can be applied to many aspects of collaborative P2P systems including resource aggregation.

Aggregating data from multiple and heterogeneous sensors in named data networking

Distributed and collaborative data fusion provides an attractive implementation choice for CASA real-time weather monitoring because data are constantly being generated, processed, pushed and pulled among groups of radars, storage, and processing nodes. While we demonstrated the suitability of NDN and overlay network based data fusion for DCAS systems (Chapter 8), a lot more work is needed to aggregate groups of heterogeneous, distributed, dynamic, and multi-attribute resources as and when needed [Ba12h]. While the solution proposed to resolve event-specific queries using the 2D-torus outperformed the ring and multi-dimensional-torus based designs, it was not able to benefit entirely from NDN features such as caching and duplicate message suppression. An interest message corresponding to an event-specific query had to first go through a series of zones that overlapped with the query region, based on the order given by the points on the space-filling curve. Once the query reached the last zone, it was resolved by searching the sensor readings stored in the zone controller. Sensor readings stored in the intermediate zones were checked only during the reverse path and matching sensor readings were appended to the data packet. The solution was designed this was as NDN do not allow multiple data packets to be received for the same interest packet and the number of data packets to be generated cannot be determined a priory. However, this also hinders the possibility of benefiting from cached query responses for some of the zones that need to be visited. As a query cannot be resolved until it reaches the last zone (unless an exact match is found in a cache), our solution cannot benefit from cached query results that may be available for a subset of the zones to be visited. The ability to aggregate data from multiple data sources for the same interest packet is also important in many other applications including P2P-based RD and distributed databases. Hence, it is important to develop solutions that can aggregate data from multiple sources (for the same interest packet) while benefiting from multiple features in NDN. Our proof-of-concept solution considered radars and weather stations separately. Hence, it is also important to extend the NDN-based

data fusion solution to integrate multiple heterogeneous sensors with diverse data types, rates, and generation patterns. For example, CASA utilizes pressure sensors and micro-weather stations to increase the accuracy of detecting, tracking, and forecasting tornados [Ba12h, Pe11a, Pe12]. Moreover, integration of mobile sensors is also important, e.g., radars mounted on vehicles that may be deployed during anticipated severe weather events. Building an actual data fusion system using CCNx [Palo] that can be readily deployed on DCAS systems is also of interest.

Supporting incentives, trust, security, and privacy

Solutions designed to support incentives, trust, privacy, and security in conventional P2P systems need to be extended to support interactions among multiple groups of heterogeneous resources in collaborative P2P systems [Ba12b]. Multi-attribute resource discovery/aggregation solutions may treat incentives and reputation values/scores of resources/users as another set of attributes. Incentives and reputation values need to be preserved even after a resource leaves the system (due to failure or churn), as it is costly and time consuming to regain those values when the resource rejoins. Furthermore, security, integrity, and accountability of the nodes as well as maintaining the incentives and trust values are of utmost importance, as they can become easy targets for attacks. Guidelines need to be identified for determining credits/payments and reputation scores for heterogeneous resources. For example, while both a radar and a set of rain gauges are important for weather monitoring, formally evaluating the cost and perceived value of such systems in a consistent manner is difficult. Moreover, while a computed result must be always accurate, accuracy of sensor data is dependent on many dynamic parameters. Formal analysis of incentive schemes such as [Zh12] need to be extended to understand under what conditions a collaborative P2P system will be robust and when will it collapse. Anonymity is in conflict with the incentives, trust, and security; hence, it is important to look for distributed solutions that overcome this limitation. Because the key phases of resource collaboration as well as incentives, trust, privacy and security are essential elements of a collaborative P2P system, their designs and performance should also be evaluated in the context of the overall system. Issues related to incentives, trust, privacy, and security might seem to be overweighting

the benefits of collaborative P2P systems. However, with the right tools and incentives in place, it will be more useful, efficient, and rewarding to accomplish a greater task through the collaboration.

REFERENCES

[Ad00] L. A. Adamic, "Zipf, power-laws, and Pareto – A ranking tutorial," Apr. 2000, Available: http://www.hpl.hp.com/research/idl/papers/ranking/ranking.html

[Ad02] L. A. Adamic and B. A. Huberman, "Zipf's law and the Internet," *Glottometrics*, vol. 3, 2002, pp. 143–150.

[Al08] J. Albrecht, D. Oppenheimer, D. Patterson, and A. Vahdat, "Design and implementation tradeoffs for wide-area resource discovery," *ACM Transactions on Internet Technology*, vol. 8, no. 4, Sep. 2008.

[Am08] Amazon Web Services LLC, "Amazon EC2 service level agreement," Oct. 2008, Available: http://aws.amazon.com/ec2-sla/

[Amaz] Amazon Web Services LLC, "Amazon EC2 instance types," Available: http://aws.amazon.com/ec2/instance-types/

[An06] D. P. Anderson and G. Fedak, "The computational and storage potential of volunteer computing," In Proc. *Int. Symp. on Cluster Computing and the Grid*, May 2006.

[An09] D. P. Anderson and K. Reed, "Celebrating diversity in volunteer computing," In Proc. *Hawaii Int. Conf. on System Sciences*, Jan. 2009.

[An10] A. Andrzejak, D. Kondo, and D. P. Anderson, "Exploiting non-dedicated resources for cloud computing," In Proc. *12th IEEE/IFIP Network Operations and Management Symposium (NOMS '10)*, Apr. 2010.

[An11] B. An, V. Lesser, D. Westbrook, and M. Zink. "Agent-mediated multi-step optimization for resource allocation in distributed sensor networks," In. Proc. *Autonomous Agents and Multi-Agent Systems*, May 2011.

[Ar09] M. Armbrust et al., "Above the clouds: A Berkeley view of cloud," *Technical Report,* No. UCB/EECS-2009-28, Feb. 2009.

[Az09] M. Azua, "The social factor," *IBM Press*, Aug. 2009.
`

[Ba07a] T. Banka, P. Lee, A. P. Jayasumana, and J. Kurose, "An architecture and a programming interface for application-aware data dissemination using overlay networks," In Proc. *2nd IEEE/Create-Net/ICST COMSWARE '07*, Jan. 2007.

[Ba07b] I. Baumgart, B. Heep, and S. Krause, "OverSim: A flexible overlay network simulation framework," In Proc. *10th IEEE Global Internet Symposium*, May 2007, pp. 79–84.

[Ba11a] S. K. Baek, S. Bernhardsson, and P. Minnhagen, "Zipf's law unzipped," *New Journal of Physics*, vol. 13, Apr. 2011.

[Ba11b] A. Bahga and V. K. Madisetti, "Synthetic workload generation for cloud computing applications," *Journal of Software Engineering Applications*, vol. 4, 2011, pp. 396–410.

[Ba11c] H. M. N. D. Bandara and A. P. Jayasumana, "Exploiting communities for enhancing lookup performance in structured P2P systems," In Proc. *IEEE Int. Conf. on Communications (ICC '11)*, June 2011.

[Ba11d] H. M. N. D. Bandara and A. P. Jayasumana, "On characteristics and modeling of P2P resources with correlated static and dynamic attributes," In Proc. *IEEE Global Communications Conference (GLOBECOM '11)*, Dec. 2011.

[Ba11e] H. M. N. D. Bandara and A. P. Jayasumana, "Characteristics of multi-attribute resources/queries and implications on P2P resource discovery," In Proc. *Int. Conf. on Computer Systems and Applications (AICCSA '11)*, Dec. 2011.

[Ba12a] H. M. N. D. Bandara and A. P. Jayasumana, "Evaluation of P2P resource discovery architectures using real-life multi-attribute resource and query characteristics," In Proc. *IEEE Consumer Communications and Networking Conf. (CCNC '12)*, Jan. 2012.

[Ba12b] H. M. N. D. Bandara and A. P. Jayasumana, "Collaborative applications over peer-to-peer systems – Challenges and solutions," *Peer-to-Peer Networking and Applications*, Springer, 2012, DOI: 10.1007/s12083-012-0157-3.

[Ba12c] H. M. N. D. Bandara and A. P. Jayasumana, "Resource and query aware, peer-to-peer-based multi-attribute resource discovery," In Proc. *37th IEEE Conf. on Local Computer Networks (LCN '12)*, Oct. 2012.

[Ba12d] H. M. N. D. Bandara, A. P. Jayasumana, and M. Zink, "Radar networking in collaborative adaptive sensing of atmosphere: State of the art and research challenges," In Proc. *IEEE Globecom Workshop on Radar and Sonar Networks (RSN '12)*, Dec. 2012.

[Ba12e] H. M. N. D. Bandara and A. P. Jayasumana, "Community-based caching for enhanced lookup performance in P2P systems," *IEEE Transactions on Parallel and Distributed Systems*, 2012, DOI: 10.1109/TPDS.2012.270.

[Ba12f] H. M. N. D. Bandara and A. P. Jayasumana, "Multi-attribute resource and query characteristics of real-world systems and implications on peer-to-peer-based resource discovery," To be submitted to *ACM Transactions on Internet Technology*.

[Ba12g] H. M. N. D. Bandara and A. P. Jayasumana, "On characteristics and generation of multi-attribute resources and queries with correlated attributes," To be submitted to *IEEE Transactions on Parallel and Distributed Systems*.

[Ba13] H. M. N. D. Bandara and A. P. Jayasumana, "Distributed multi-sensor data fusion over named data networks," Submitted to *5th Int. Conf. on Communication Systems and Networks (COMSNETS '13)*, Jan. 2013.

[Ba82] J. J. Bartholdi III and L.K. Platzman, "An O(N log N) planar travelling salesman heuristic based on spacefilling curves," *Operations Research Letters*, vol. 1, no. 4, Sep. 1982, pp. 121–125.

[Be06] R. Beverly, K. Sollins, and A. Berger, "SVM learning of IP address structure for latency prediction," In Proc. *ACM SIGCOMM Workshop on Mining Network Data*, Sep. 2006, pp. 299–304.

[Be10] M. Bertier, D. Frey, R. Guerraoui, A. Kermarrec, and V. Leroy, "The Gossple anonymous social network," In Proc. *ACM/ IFIP/USENIX 11th Middleware Conf.*, Dec. 2010, pp. 191–211.

[Bh04] A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting scalable multi-attribute range queries," In Proc. *ACM Special Interest Group on Data Communication (SIGCOMM '04)*, Aug./Sep. 2004.

[Bl58] R. B. Blackman and J. W. Tukey, "The measurement of power spectra, from the point of view of communications engineering," *New York: Dover*, 1958, pp. 95–100.

[Bo04] S. Boyd and L. Vandenberghe, "Convex optimization," *Cambridge University Press*, 2004, pp. 71 and 243.

[Br05] J. Brophy and D. Bawden, "Is Google enough? Comparison of an internet search engine with academic library resources," *Aslib Proceedings*, vol. 57, no. 6, 2005, pp. 498–512.

[Br07] J. Brotzge et al., "CASA IP1: Network operations and initial data," In Proc. *23rd Conf. on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology*, *American Meteorological Society*, vol. 8A.6, 2007.

[Br09] G. Briscoe and A. Marinos, "Digital ecosystems in the clouds: Towards community cloud computing," In Proc. *3rd IEEE Int. Conf. on Digital Ecosystems and Technologies*, June 2009, pp. 103–108.

[Ca02] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, Oct. 2002, pp. 100–110.

[Ca04] M. Cai, M. Frank, J. Chen, and P. Szekely, "MAAN: A multi-attribute addressable network for grid information services," *Journal of Grid Computing*, Jan. 2004.

[Ch02] J. Chu, K. Labonte, and B. N. Levine, "Availability and locality measurements of peer-to-peer file systems," In Proc. *ITCom: Scalability and Traffic Control in IP Networks II*, July 2002, pp. 310–321.

[Ci12] Cisco Systems Inc., "Cisco visual networking index: Forecast and methodology, 2011–2016," May 2012.

[CNRL] Computer Networking Research Laboratory (CNRL), "Collaborative peer-to-peer (P2P) systems," Available: http://www.cnrl.colostate.edu/Projects/CP2P/

[Co02] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," In Proc. *ACM Special Interest Group on Data Communication (SIGCOMM '02)*, Aug. 2002.

[Co09a] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to algorithms," 3rd edition, *MIT Press*, 2009, pp. 1078.

[Co09b] P. Costa, J. Napper, G. Pierre, and M. Steen, "Autonomous resource selection for decentralized utility computing," In Proc. *29th Int. Conf. on Distributed Computing Systems*, June 2009.

[Co10] M. Conti, S. Giordano, M. May, and A. Passarella, "From opportunistic networks to opportunistic computing," *IEEE Communications Magazine*, vol. 48, no. 9, 2010, pp. 126–139.

[Da03] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica, "Towards a common API for structured peer-to-peer overlays," In Proc. *Int. Workshop on Peer-To-Peer Systems*, 2003.

[Da04] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," In Proc. *ACM Special Interest Group on Data Communication (SIGCOMM '04)*, Aug. 2004.

[De09] L. Dekar and H. Kheddouci, "A resource discovery scheme for large scale ad hoc networks using a hypercube-based backbone," In Proc. *Int. Conf. on Advanced Information Networking and Applications*, 2009, pp. 293–300.

[De10] Z. Despotovic, Q. Hofstätter, M. Michel, and W. Kellerer,, "An operator approach to popularity-based caching in DHTs," In Proc. *Int. Conf. on Communications (ICC '10)*, May 2010.

[De78] P. Deheuvels, "Caractérisation complète des lois extrèmes multivariées et de la Convergence des types extrèmes," *Institute of Statistics of the University of Paris*, vol. 23, 1978, pp. 1–36.

[Do05] B. Donovan, D. McLaughlin, J. Kurose, and V. Chandrasekar, "Principles and design considerations for short-range energy balanced radar networks," In Proc. *Int. Geoscience and Remote Sensing Symposium (IGARSS '05)*, 2005, pp. 2058–2061.

[El09] C. Elliott, "GENI: exploring networks of the future," Mar. 2009, Available: http://www.geni.net

[El11] T. Elteto, C. Germain-Renaud, P. Bondon, and M. Sebag, "Towards non-stationary grid models," *Journal of Grid Computing*, vol. 9, no. 4, Dec. 2011, pp. 423–440.

[Fa09] B. Fan, J. C. S. Lui, and D. Chiu, "The design trade-offs of BitTorrent-like file sharing protocols," *IEEE/ACM Transactions on Networking*, vol. 17, no. 2, Apr. 2009, pp. 365–376.

[Fo09] M. Fouquet, H. Niedermayer, and G. Carle, "Cloud computing for the masses," In Proc. *1st ACM workshop on User-provided networking: challenges and opportunities (U-NET '09)*, Dec. 2009, pp. 31–36.

[Ga04a] P. Ganesan, B. Yang, and H. Garcia-Molina, "One torus to rule them all: Multi-dimensional queries in P2P systems," In Proc. *7th Int. Workshop on the Web and Databases (WebDB '04)*, June 2004.

[Ga04b] J. Gao and P. Steenkiste, "An adaptive protocol for efficient support of range queries in DHT-based systems," In Proc. *12th IEEE Int. Conf. on Network Protocols (ICNP '04)*, 2004.

[Ga78] M. R. Garey and D. S. Johnson, "Computers and intractability: A guide to the theory of NP-completeness," *Freeman*, San Francisco, 1978, pp. 65.

[Ge07] M. N. George, "B-A scale-free network generation and visualization," Apr. 2007, Available: www.mathworks.com/matlabcentral/ fileexchange/11947

[Ge11a] C. Germain-Renaud et al., "The grid observatory," In Proc. *11th IEEE Int. Symposium on Cluster, Cloud and Grid Computing*, May 2011.

[Ge11b] C. Germain-Renaud, F. Furst, M. Jouvin, G. Kassel, J. Nauroy, and G. Philippon, "The green computing observatory: A data curation approach for green IT," In Proc. *9th IEEE Int. Conf. on Dependable, Autonomic and Secure Computing*, Dec. 2011.

[Gi10] S. Girdzijauskas, G. Chockler, Y. Vigfusson, Y. Tock, and R. Melamed, "Magnet: Practical subscription clustering for Internet-scale publish/subscribe," In Proc. *4th ACM Int. Conf. on Distributed Event-Based Systems*, July 2010.

[Go04] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in dynamic structured P2P systems," In Proc. IEEE *Int. Conf. on Computer Communications (INFOCOM '04)*, 2004, pp. 2253–2262.

[Gu03] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The impact of DHT routing geometry on resilience and proximity," In Proc. *ACM Special Interest Group on Data Communication (SIGCOMM '03)*, Aug. 2003, pp. 381–394.

[Ha06] S. B. Handurukande, A. M. Kermarrec, F. Le Fessant, L. Massoulié, and S. Patarin, "Peer sharing behaviour in the eDonkey network, and implications for the design of server-less file sharing systems," In Proc. *EuroSys '06*, vol. 40, no. 2, Apr. 2006, pp. 359–371.

[He09] E. M. Heien, D. P. Anderson, and K. Hagihara, "Computing low latency batches with unreliable workers in volunteer computing environments," *Journal of Grid Computing*, Aug. 2009.

[He12] E. M. Heien, D. Kondo, and D. P. Anderson, "A correlated resource models of Internet end hosts," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 6, 2012, pp. 977–984.

[Hu04] K. Y. K. Hui, J. C. S. Lui, and D. K. Y. Yau, "Small world overlay P2P networks," In Proc. *Int. Workshop on Quality of Service*, June 2004.

[Ik09] S. Ikeda, I. Kubo, and M. Yamashita, "The hitting and cover times of random walks on finite graphs using local degree information," *Theoretical Computer Science*, vol. 410, no. 1, 2009, pp. 94–100.

[Io10] A. Iosup and D. Epema, "Grid computing workloads: Bags of tasks, workflows, pilots, and others," *IEEE Internet Computing*, vol. 99, 2010.

[Ir10] D. Irwin, P. Shenoy, E. Cecchet, and M. Zink, "Resource management in data-intensive clouds: Opportunities and challenges," In Proc. *17th IEEE Work. on Local and Metropolitan Area Networks (LANMAN '10)*, May 2010.

[Ja09a] V. Jacobson, D. K. Smetters, J. D. Thornton, M. Plass, N. Briggs, and R. L. Braynard, "Networking named content," In Proc. *5th ACM Int. Conf. on Emerging Networking Experiments and Technologies (CoNEXT '09)*, Dec. 2009, pp. 1–12.

[Ja09b] V. Jacobson et al., "VoCCN: Voice over content-centric networks," In Proc. *ACM Workshop on Re-architecting the Internet (ReArch '09),* Dec. 2009.

[Ja90] H. V. Jagadish, "Linear clustering of objects with multiple attributes," *ACM SIGMOD Record*, vol. 19, no. 2, June 1990, pp. 332–342.

[Je06] M. Jelasity and A. Kermarrec, "Ordered slicing of very large-scale overlay networks," In Proc. *6th IEEE Int. Conf. on Peer-to-Peer Computing*, 2006, pp. 117–124.

[Ka11] I. A. Kash, J. K. Lai, H. Zhang, and A. Zohar, "Economics of BitTorrent communities," In Proc. *6th Work. on Economics of Networks, Systems, and Computation (NetEcon '11)*, June 2011.

[Ka97] D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web," In Proc. *29th Annual ACM Symposium on Theory of Computing*, May 1997, pp. 654–663.

[Ke04] Y. S. Kee, H. Casanova, and A. Chien, "Realistic modeling and synthesis of resources for computational grids," In Proc. *ACM/IEEE Conf. on Supercomputing*, Nov 2004.

[Ke06] Y. Kee, K. Yocum, A. A. Chien, and H. Casanova, "Improving grid resource allocation via integrated selection and binding," In Proc. *ACM/IEEE Conf. on Supercomputing*, Nov. 2006.

[Ki11] W. Kim, A. Roopakalu, K. Y. Li, and V. S. Pai, "Understanding and characterizing PlanetLab resource usage for federated network testbeds," In Proc. *2011 Internet Measurement Conference* (IMC '11). Nov. 2011.

[Kl04] A. Klemm, C. Lindemann, M. K. Vernon, and O. P. Waldhorst, "Characterizing the query behavior in peer-to-peer file sharing systems," In Proc. *4th ACM SIGCOMM Conf. on Internet Measurement*, 2004.

[Ko07] T. Koponen et al., "A data-oriented (and beyond) network architecture," In Proc. *ACM Special Interest Group on Data Communication (SIGCOMM '07)*, Aug. 2007.

[Ko11] I. Konstantinou, D. Tsoumakos, and N. Koziris, "Fast and cost-effective online load-balancing in distributed range-queriable systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 8, Aug. 2011.

[Ku06] J. Kurose et al., "An end-user-responsive sensor network architecture for hazardous weather detection, prediction, and response," In Proc. *Asian Internet Conference (AINTEC)*, Nov. 2006.

[Kw10] S. Kwan and J. K. Muppala, "Bag-of-tasks applications scheduling on volunteer desktop grids with adaptive information dissemination," In Proc. *35th IEEE Conf. on Local Computer Networks (LCN '10)*, Oct. 2010, pp. 560–567.

[La12] G. V. Laszewski et al., "Design of a dynamic provisioning system for a federated cloud and bare-metal environment," In Proc. *Workshop on Cloud Services, Federation, and the 8th Open Cirrus Summit*, Sep. 2012.

[Le07] J. Ledlie, P. Gardner, and M. Seltzer, "Network coordinates in the wild," In Proc. *USENIX NSDI '07*, Apr. 2007.

[Le12] P. Lee, A. P. Jayasumana, H. M. N. D. Bandara, S. Lim, and V. Chandrasekar, "A peer-to-peer collaboration framework for multi-sensor data fusion," *Journal of Network and Computer Applications*, vol. 35, no. 3, May 2012, pp. 1052–1066.

[Li06] D. Li and X. Sun, "Nonlinear integer programming," *Springer*, New York, 2006.

[Li07a] M. Li et al., "Multi-user data sharing in radar sensor networks," In Proc. *5th ACM Conf. on Embedded Networked Sensor Systems (Sensys '07)*, Nov. 2007.

[Li07b] S. Lim, V. Chandrasekar, P. Lee, and A. P. Jayasumana, "Reflectivity retrieval in a networked radar environment: Demonstration from the CASA IP-1 radar network," In Proc. *Int. Geoscience and Remote Sensing Symposium (IGARSS '07)*, July 2007.

[Li07c] L. Liu, N. Antonopoulos, and S. Mackin, "Fault-tolerant peer-to-peer search on small-world networks," *Future Generation Computer Systems*, vol. 23, 2007, pp. 921–931.

[Li09a] L. Liu, J. Xu, D. Russell, and Z. Luo, "Evolution of social models in peer-to-peer networking: Towards self-organising networks," In Proc. *6th Int. Conf. on Fuzzy Systems and Knowledge Discovery*, 2009.

[Li09b] L. Liu, N. Antonopoulos, S. Mackin, J. Xu, and D. Russell, "Efficient resource discovery in self-organized unstructured peer-to-peer networks," *Concurrency and Computation: Practice and Experience*, vol. 21, 2009, pp. 159–183.

[Lu03] D. Lu and P. A. Dinda, "Synthesizing realistic computational grids," In Proc. *ACM/IEEE Conf. on Supercomputing*, Nov. 2003.

[Lu04] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys and Tutorials*, vol. 7, Mar 2004, pp. 72–99.

[Lv02] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," In Proc. *16th Int. Conf. on Supercomputing (ICS '02)*, June 2002, pp. 84–95.

[Ma01] D. Matthys, "Spatial filters," Feb. 2001, Available: http://academic.mu.edu/phys/matthysd/web226/L0205.htm

[Ma02] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric," In Proc. *1st Int. Workshop on Peer-to-peer Systems (IPTPS '02)*, Feb. 2002, pp. 53–65.

[Mc05] D. J. McLaughlin et al., "Distributed collaborative adaptive sensing (DCAS) for improved detection, understanding, and prediction of atmospheric hazards," In Proc. *AMS IIPS for Meteorology, Oceanography, and Hydrology, American Meteorological Society*, 11.3, 2005.

[Mc09] D. McLaughlin et al., "Short-wavelength technology and the potential for distributed networks of small radar systems," *Bulletin of the American Meteorological Society*, vol. 90, Dec. 2009, pp. 1797–1817.

[Me01] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An approach to universal topology generation," In Proc. *Int. Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS '01)*, Aug. 2001.

[Me10] M. Meulpolder et al., "Public and private BitTorrent communities: A measurement study," In Proc. *9th Int. Conf. on Peer-to-Peer Systems*, Apr. 2010.

[Micr] Microsoft, "Windows Azure platform introductory special," Available: http://www.microsoft.com/windowsazure/offers/popup/popup.aspx?lang=en&locale=en-US&offer=MS-AZR-0001P

[Moor] D. Moore, "Fast Hilbert curve generation, sorting, and range queries," Available: http://www.tiac.net/~sw/2008/10/Hilbert/moore/index.html

[Mo01] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz, "Analysis of the clustering properties of the Hilbert space-filling curve," *IEEE Transactions on Knowledge and Data Engineering*, vol.13, no.1, Jan/Feb. 2001, pp. 124–141.

[Mo05] R. Morselli, B. Bhattacharjee, A. Srinivasan, and M. A. Marsh, "Efficient lookup on unstructured topologies," In Proc. *24th ACM Symposium on Principles of Distributed Computing (PODC '05)*, July 2005.

[Ne06] R. B. Nelsen, "An introduction to copulas," 2nd edition, *New York: Springer*, 2006, DOI: 10.1007/0-387-28678-0.

[Ne11] S. Newhouse, "European grid infrastructure – An integrated sustainable pan-European infrastructure for researchers in Europe (EGI-InSPIRE)," *Technical report* EGI-doc-201-v6, Apr. 2011, Available: http://go.egi.eu/pdnon

[NOAAa] National Oceanic and Atmospheric Administration, "Hourly/sub-hourly observational data," Available: http://gis.ncdc.noaa.gov/map/cdo/

[NOAAb] National Oceanic and Atmospheric Administration, "Meteorological assimilation data ingest system (MADIS)," Available: http://madis.noaa.gov/index.html

[NWS] National Weather Service, "National weather service organization," Available: http://www.weather.gov/organization

[Or84] J. A. Orenstein and T. H. Merrett, "A class of data structures for associative searching," In Proc. *3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, 1984, pp. 181–190.

[Palo] Palo Alto Research Center, "CCNx," Available: http://www.ccnx.org/

[Pa06] K. Park and V. S. Pai, "CoMon: A mostly-scalable monitoring system for PlanetLab," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1, 2006.

[Pe11a] D. Pepyne et al., "An integrated radar-infrasound network for meteorological infrasound detection and analysis," In Proc. *91st American Meteorological Society Annual Meeting*, Jan. 2011.

[Pe11b] D. Pepyne et al., "Dense radar networks for low-flyer surveillance," In Proc. *IEEE Conf. on Technologies for Homeland Security*, Nov. 2011.

[Pe12] D. Pepyne and S. Klaiber, "Highlights from the 2011 CASA Infrasound field experiment," In Proc. *92nd American Meteorological Society Annual Meeting*, Jan. 2012.

[Pf11] D. Pfisterer et al.,"SPITFIRE: Toward a semantic web of things," *IEEE Communications Magazine*, vol. 49, no. 11, 2011, pp. 40–48.

[Pi06] T. Pitoura, N. Ntarmos, and P. Triantafillou, "Replication, load balancing and efficient range query processing in DHTs," In Proc. *10th Int. Conf. Extending Database Technology (EDBT)*, 2006, pp. 131–148.

[Plan] PlanetLab, Available: http://www.planet-lab.org/

[Po05] J. A. Pouwelse, P. Garbacki, D. H. J. Epema, and H. J. Sips, "The Bittorrent P2P file-sharing system: Measurements and analysis," In Proc. *4th Int. Workshop on Peer-to-Peer Systems (IPTPS)*, 2005.

[Qi04] D. Qiu and R. Srikant, "Modeling and performance analysis of BitTorrent-like peer-to-peer networks," In Proc. *Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Aug.–Sep. 2004, pp. 367–378.

[Ra01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," In Proc. *ACM Special Interest Group on Data Communication (SIGCOMM '01)*, Aug. 2001.

[Ra04] V. Ramasubramanian and E. G. Sirer, "Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays," In Proc. *USENIX NSDI '04*, vol. 1, 2004, pp. 99–112.

[Ra07] W. Rao, L. Chen, A. W. Fu, and Y. Bu, "Optimal proactive caching in peer-to-peer network: Analysis and application," In Proc. *6th ACM Conf. on Information and Knowledge Management*, Nov. 2007, pp. 663–672.

[Ra08] R. Ranjan, A. Harwood, and R. Buyya, "Peer-to-peer based resource discovery in global grids: A tutorial," *IEEE Communication Surveys*, vol. 10, no. 2, 2008.

[Ra10] W. Rao, L. Chen, A. W. -C. Fu, and G. Wang, "Optimal resource placement in structured peer-to-peer networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 7, July 2010.

[Ri02] M. Ripeanu and I. Foster, "Mapping the Gnutella network: Macroscopic properties of large-scale peer-to-peer systems." In Proc. *1st Int. Workshop on Peer-to-Peer Systems (IPTPS '02)*. 2002, pp. 85–93.

[Ro01] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," In Proc. *IFIP/ACM Int. Conf. on Distributed Systems Platforms*, Nov. 2001, pp. 329–350.

[Ro11] P. Rosenmai, "Lorenz curve graphing tool & Gini coefficient calculator," May 2011, Available: http://www.peterrosenmai.com/lorenz-curve-graphing-tool-and-gini-coefficient-calculator

[Sa68] G. Salton and M. E. Lesk, "Computer evaluation of indexing and text processing," *Association for Computing Machinery*, vol. 15, no. 1, 1968, pp. 8–36.

[Se09] J. Seedorf, S. Kiesel, and M. Stiemerling, "Traffic localization for P2P-applications: The ALTO approach," In Proc. *9th IEEE Int. Conf. on Peer-to-Peer Computing (P2P '09)*, Sep. 2009, pp.171–177.

[SETI] Search for Extraterrestrial Intelligence (SETI@home) Statistics, Available: http://setiathome.berkeley.edu/stats/

[Sh06] H. Shen, C. Xu, and G. Chen, "Cycloid: A constant-degree and lookup-efficient P2P overlay network," *Performance Evaluation*, vol. 63, no. 3, Mar. 2006, pp. 195–216.

[Sh07] H. Shen, A. Apon, and C. Xu, "LORM: Supporting low-overhead P2P-based range-query and multi-attribute resource management in grids," In Proc. *13th Int. Conf. on Parallel and Distributed Systems*, Dec. 2007.

[Sh09] H. Shen and C. Xu, "Performance analysis of DHT algorithms for range-query and multi-attribute resource discovery in grids," In Proc. *Int. Conf. on Parallel Processing*, Sep. 2009. pp. 246–253.

[Sl11] D. E. Slotnik, "Users help a weather site hone its forecasts," *The New York Times*, Mar. 21, 2011.

[So08] M. Sozio, T. Neumann, and G. Weikum, "Near-optimal dynamic replication in unstructured peer-to-peer networks," In Proc. *27th ACM symposium on Principles of Database Systems (PODS '08)*, June 2008, pp. 281–290.

[Sr01] K. Sripanidkulchai, "The popularity of Gnutella queries and its implications on scalability," Feb. 2001, Available: www.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html

[St02] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," In Proc. *ACM Special Interest Group on Data Communication (SIGCOMM '02)*, Aug. 2002.

[St03] I. Stoica et al., "Chord: a scalable peer-to-peer protocol for internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, Feb. 2003, pp. 17–32.

[St08] D. Stutzbach, R. Rejaie, and S. Sen, "Characterizing unstructured overlay topologies in modern P2P file-sharing systems," *IEEE/ACM Transactions on Networking*, vol. 16, no. 2, Apr. 2008, pp. 267–280.

[St09] J. C. Strelen, "Tools for dependent simulation input with copulas," In Proc. *2nd Int. Conf. on Simulation Tools and Techniques*, Mar. 2009.

[Su08a] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya "A toolkit for modelling and simulating data grids: An extension to gridsim," *Concurrency and Computation: Practice & Experience*, vol. 20, no. 13, Sep. 2008, pp. 1591–1609.

[Su08b] X. Sun, Y. Tian, Y. Liu, and Y. He, "An unstructured P2P network model for efficient resource discovery," In Proc. *Int. Conf. on Applications of Digital Information and Web Technologies*, Aug. 2008, pp. 156–161.

[Ta08] Y. Tan, J. Han, and Y. Lu, "Agent-based intelligent resource discovery scheme in P2P networks," In Proc. *Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, Dec. 2008, pp. 752–756.

[Th04] S. A. Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Computing Surveys*, vol. 36, no. 4, Dec. 2004, pp. 335–371.

[Th09] R. Thanawala, J. Wu, and A. Srinivasan, "Efficient resource discovery in mobile ad hoc networks," In Proc. *IEEE Int. Conf. on Communications (ICC '09)*, June 2009.

[Vi05] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco, "Probabilistic finite-state machines – Part I," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol.27, no.7, July 2005, pp.1013–1025.

[Vu09] Q. H. Vu, B. C. Ooi, M. Rinard, and K. -L. Tan, "Histogram-based global load balancing in structured peer-to-peer systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 4, Apr. 2009.

[Wo04] J. L. Worrall and T. C. Pratt, "Estimation issues associated with time-Series – Cross-section analysis in criminology," *Western Criminology Review*, vol. 5, no. 1, 2004, pp. 35–49.

[Xi08a] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. G. Liu, and A. Silberschatz, "P4P: Provider portal for applications," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, Oct. 2008, pp. 351–362.

[Xi08b] Z. Xiong, Y. Yang, X. Zhang, M. Zeng, and L. Liu, "A grid resource discovery model using P2P technology," In Proc. *Int. Conf. on Intelligent Information Hiding and Multimedia Signal Processing*, Aug. 2008, pp. 1553–1556.

[Xu10] F. Xue, G. Feng, and Y. Zhang, "CommuSearch: Small-world based semantic search architecture in P2P networks," In Proc. *IEEE Global Communications Conference (GLOBECOM '10)*, Dec. 2010.

[Ya06] J. Yao, J. Zhou, and L. Bhuyan, "Computing real time jobs in P2P networks," In Proc. *31st IEEE Conf. on Local Computer Networks (LCN '06)*, Nov. 2006, pp. 107–114.

[Za05] T. Zahn and J. Schiller, "MADPastry: A DHT substrate for practicably sized MANETs," In Proc. *5th Workshop on Applications and Services in Wireless Networks*, June/July 2005.

[Ze03] A. Zeileis, C. Kleiber, W. Krämer, and K. Hornik, "Testing and dating of structural changes in practice," *Journal of Computational Statistics and Data Analysis*, vol. 44, no. 1-2, Oct. 2003, pp. 109–123.

[Ze11] N. Zeilemaker, M. Capotă, A. Bakker, and J. Pouwelse, "Tribler: P2P media search and sharing," In Proc. *19th ACM Int. conf. on Multimedia*, Nov.-Dec. 2011, pp. 739–742.

[Ze96] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to model an internetwork," In Proc. *IEEE Int. Conf. on Computer Communications (INFOCOM '96*). Mar. 1996, pp. 594–602.

[Zh10] B. Zhang, A. Iosup, J. Pouwelse, D. Epema, and H. Sips, "Sampling bias in BitTorrent measurements," In Proc. *Euro-Par 2010*, Aug.-Sep., 2010.

[Zh12] B. Q. Zhao, J. C. S. Lui, and D-M. Chiu, "A mathematical framework for analyzing adaptive incentive protocols in P2P networks," *IEEE/ACM Transactions on Networking*, vol. 20, no. 2, Apr. 2012, pp. 367–380.

[Zi05] M. Zink et al., "Meteorological command and control: An end-to-end architecture for a hazardous weather detection sensor network," In Proc. *ACM Mobisys Workshop on End-to-End, Sense-and-Respond Systems, Applications, and Services*, June 2005, pp. 37–42.

Appendix I

NUMBER OF BITTORRENT COMMUNITIES ACCESSED BY USERS

No measurements are available on the number of communities accessed by P2P users. It is not straightforward to measure such behavior accurately without probing all the possible communities (or a large representative subset of them) and all the resources shared by each of them as BitTorrent-like systems track their users based on the files they access. Moreover, the rate limiting adopted by BitTorrent trackers and use of anonymizing services further impact such a measurement process. Instead, we conducted an online survey to find out the number of communities accessed by BitTorrent users and their frequencies. Survey link was posted on user communities of www.kat.ph, www.forum.suprbay.org, and www.fenopy.eu, and e-mailed to friends between 11/03/2012 and 14/04/2012. We received 238 positive responses (332 attempted the survey) from 42 countries as of 09/05/2012. Survey questions are given in Section I.1 and survey results are given in Section I.2. Survey data are available in [CNRL].

I.1 Survey Questions

Question 1

Do you use BitTorrent, µTorrent, Xunlei, Vuze, etc. to download Videos, Music, Games, Software, e-Books, etc.?

- ○ Yes
- ○ No

Question 2

What types of content **do you search/download** (select all that apply)?

- □ Movies
- □ TV shows or documentaries
- □ Music Video
- □ Anime
- □ Porn
- □ Music

- ☐ Audio Books
- ☐ Games
- ☐ Software
- ☐ e-Books
- ☐ Pictures
- ☐ Other – please specify _____

Question 3

Which of the following features on a BitTorrent search engine **do you use** (select all that apply)?

- ☐ Search
- ☐ Search Cloud
- ☐ Top or most popular torrents
- ☐ Browse torrents
- ☐ Recent/latest Torrents
- ☐ Other – please specify _____

Question 4

Which of the following BitTorrent sites (search engines) **are you aware of** (select all that apply)?

- ☐ 1337x
- ☐ BitGamer.su
- ☐ BitTorrent.com (or search box in client)
- ☐ BitToxic.com
- ☐ BTscene
- ☐ ClearBits
- ☐ Entertane.com
- ☐ EZ-TV
- ☐ GamesTorrents
- ☐ Google
- ☐ Kickass Torrents (KAT)
- ☐ LINUX23
- ☐ Mininova
- ☐ Scrape Torrent
- ☐ SUMO TORRENT
- ☐ ThunderBytes
- ☐ Toorgle
- ☐ TorrentCafe
- ☐ Torrent Downloads
- ☐ torrentGamez
- ☐ torrents.to
- ☐ Torrentz
- ☐ torrentzap
- ☐ uTorrent (or search box in client)
- ☐ YouTorrent
- ☐ Vertor
- ☐ Xunlei (or search box in client)
- ☐ Other – please specify _____
- ☐ Bing
- ☐ BitSnoop
- ☐ BitTorrent Search Engine
- ☐ Blues Brothers
- ☐ Bush Torrent
- ☐ Demonoid.me
- ☐ Extra Torrent
- ☐ fenopy
- ☐ gameupdates.org
- ☐ jamendo
- ☐ isoHunt
- ☐ linux TRACKER
- ☐ NowTorrents
- ☐ Seedpeer
- ☐ The Pirate Bay
- ☐ Torlock
- ☐ Torrentbit
- ☐ Torrent Download
- ☐ Torrent Funk
- ☐ Torrent Root
- ☐ TORRENTScan
- ☐ Torrent Reactor
- ☐ Underground-Gamer
- ☐ VODO
- ☐ YourBittorrent
- ☐ WiiTorrents
- ☐ Yahoo

Question 5

List of options depends on the answers to question four. For example, suppose user selected Bing, BitTorrent.com, fenopy, The Pirate Bay, uTorrent, YouTorrent, YourBittorrent, and Yahoo.

Which of the following BitTorrent sites (search engines) **do you use** (select all that apply)?

- □ Bing
- □ BitTorrent.com
- □ fenopy
- □ The Pirate Bay
- □ uTorrent
- □ YouTorrent
- □ YourBittorrent
- □ Yahoo

Question 6

List of options depends on answers to question five. For example, suppose user selected Bing, fenopy, The Pirate Bay, and YourBittorrent.

If you were to perform **20 searches for files, how many of them would go to each** of the following sites (total should be 20)?

Bing	_____
fenopy	_____
The Pirate Bay	_____
YourBittorrent	_____
Total	========

Question 7

In what **country** do you live (for classification only)?

_____

Comments - anything that you would like to share about use of BitTorrent search engines

_____

I.2 Survey Results

Figure I.1 shows the types of contents accessed by users. As expected, users seem to be accessing a variety of contents. Movies and software are the most popular types of contents. Summary of data is listed in Table I.1.

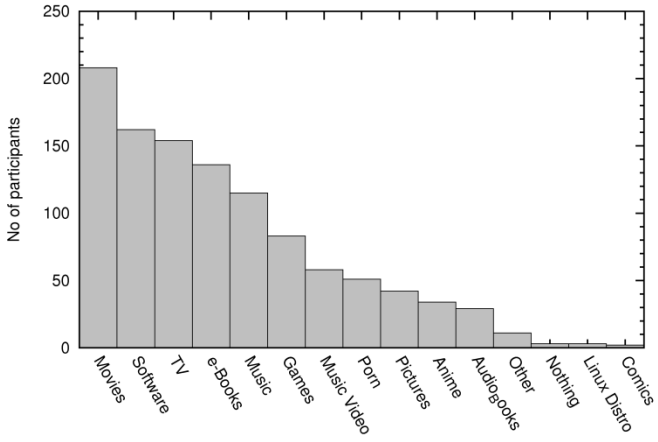


Figure I.1 – Types of contents accessed by users.

Table I.1 – Summary of findings.

	Min	Max	Average	Std.	Mode
Types of content accessed by a user	1	12	4.6	2.3	4
Search engine features	1	6	2.1	1.1	1
No of search engines known to a user	1	36	7.8	6.4	1
No of search engines used	1	19	3.4	2.8	1

Figure I.2 shows what features provided by search engines are frequently used. Majority of the users relies on search and browse options. However, their content access choices seem to be also influenced by what is popular (e.g., Top 50/100 and recent searches lists). Thus, there is a tendency for a popular content to become even more popular. Search clouds are infrequently used when deciding what to access/download however they reflect what is being searched.

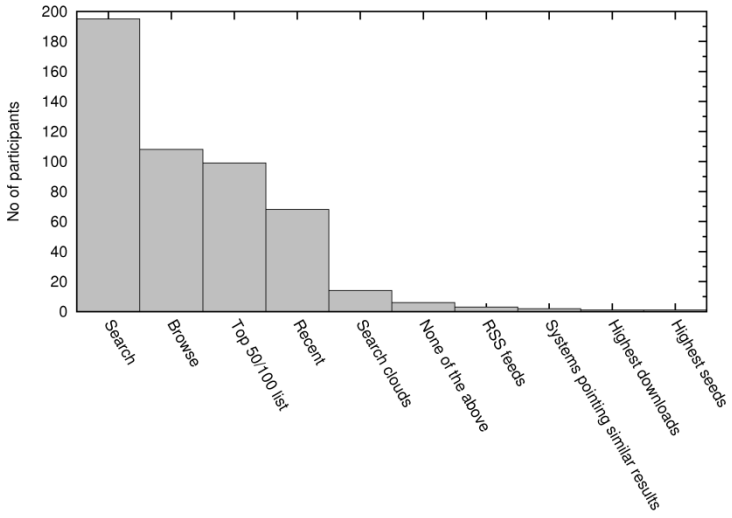


Figure I.2 – Frequently used features provided by search engines.

Search engines (communities) known to users are shown in Fig. I.3 and the ones that are actually used are shown in Fig. I.4. Only the most popular 25 search engines are shown as our focus was on distribution of search engine usage than relative popularity among them. Though users are aware of many search engines, they seem to be using only a small subset of those (average number of search engines used drop from 7.8 to 3.4).

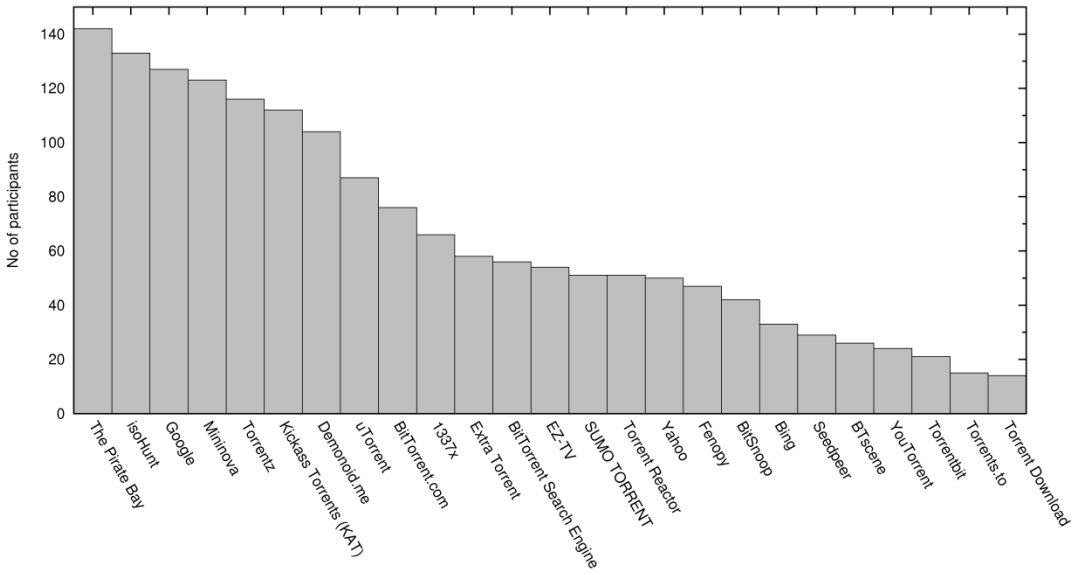


Figure I.3 – Search engines known to users.

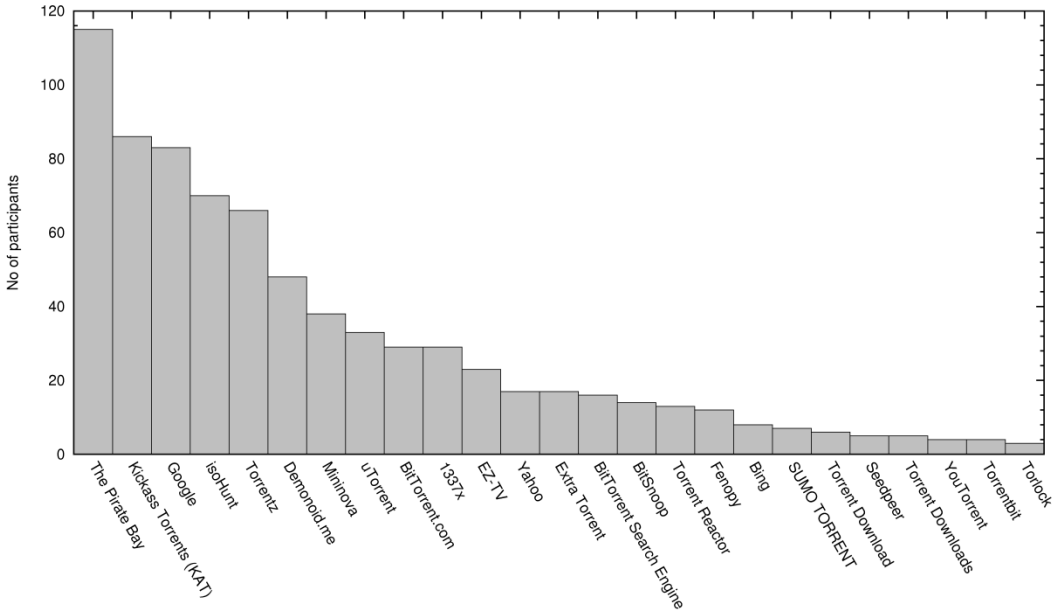


Figure I.4 – Search engines used.

Distribution of the number of communities accessed by a user is shown in Fig. I.5(a). 84% of the time users access up to five search engines. Distribution of the number of searches per search engine when a user performs 20 searches for files is shown in Fig. I.5(b). 73% of the time users prefer to use a single search engine and one to two search engines are accessed 89% of the time. Therefore, though the users tend to access contents from multiple communities (Fig. I.5(a)), they frequently revisit only one or two communities. Consequently, by catering to few most preferred communities of a given user P2P performance and quality enhancement solutions can gain better results. Figure I.6 shows the number of searches per search engine. Country of survey participants is listed in Table I.2.

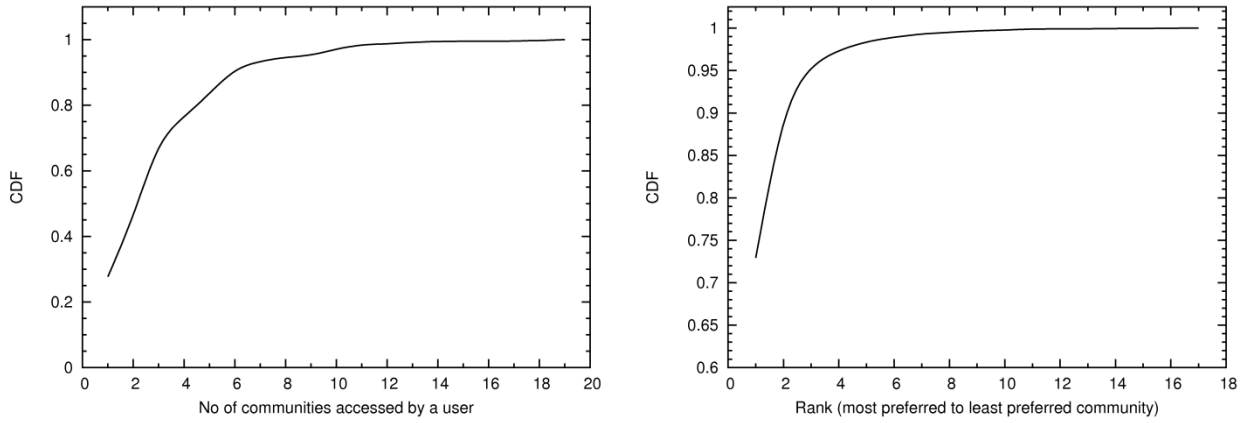


Figure I.5 – Cumulative distribution of: (a) Number of communities accessed by a user; (b) Frequency that a user revisits different communities.

Table I.2 – Country of survey participants.

Country	No	Country	No	Country	No
Sri Lanka	97	Ireland	2	Poland	1
United States	32	Greece	1	Finland	1
United Kingdom	15	Japan	1	Bosnia and Herzegovina	1
India	13	Cyprus	1	Bulgaria	1
Singapore	12	Trinidad	1	New Zealand	1
Australia	10	Italy	1	Bangladesh	1
Canada	7	South Africa	1	Pakistan	1
United Arab Emirates	4	Nepal	1	Romania	1
Norway	3	France	1	Mexico	1
Sweden	4	North Cyprus	1	Egypt	1
Malaysia	2	Ghana	1	Uganda	1
Hungary	2	Brazil	1	China	1
Iran	2	Georgia	1	Netherlands	1
Belgium	2	Spain	1		

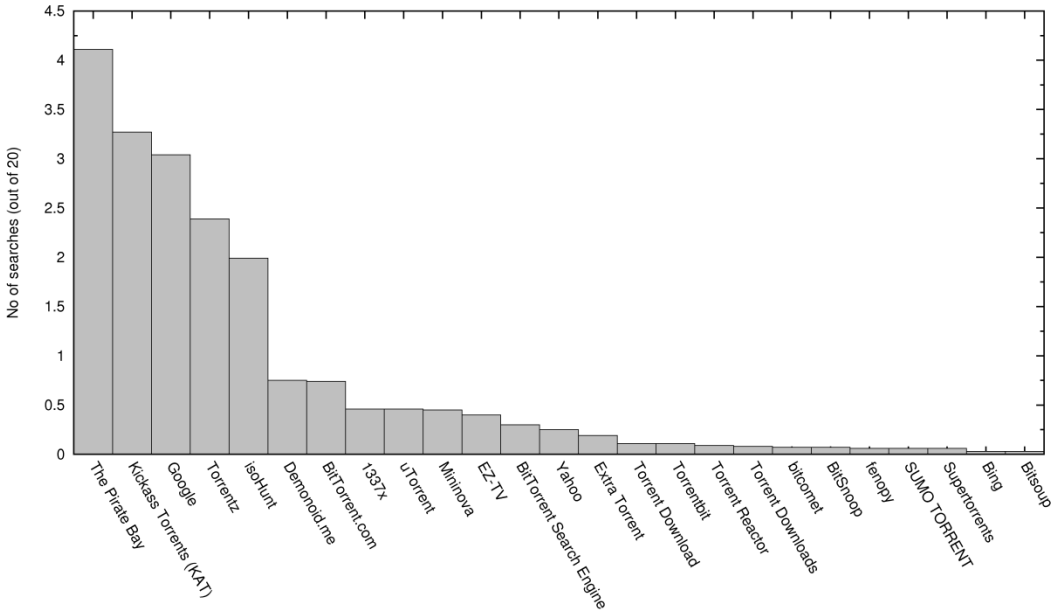


Figure I.6 – Number of searches per search engine (out of 20 searches). Only the first 25 search engines are shown.

Selected Comments

Below is a selected set of comments from survey participants.

1. Used these sites in India frequently not in the US
2. Please make sure they don't ask us to register to be able to download!
3. There aren't many seeders around. People fear using torrent clients, as 95% of downloads are illegal.
4. When I have a torrent that is working, I search other instances of that torrent by hash and add trackers. Additionally, I maintain a list of generic trackers.
5. It is unjustifiable to stop torrents as people in 3rd world countries heavily depend on torrents
6. Help torrent search engine from going down due to government rules. like btjunkie
7. Nice idea to connect the world. It's actually a social network. everyone will benefit from this
8. I didn't aware about such a number of torrent search engines
9. It would be great if possible to seach torrents using a single site. In future using anonymizing technologies like Freenet and GNUNet there will be much pirate stuff can be found in Internet if special laws against such technologies will come into play.
10. I rarely use BitTorrent search function as I use RSS to auto download my torrents.
11. Mostly I use google to search for torrent files
12. They should be made more safe and free from malwares, rootkits and backdoor trojans

13. IF there is a global bittorrent downloader engine which perform global serach on user defined bit-torrent sites & retirves torrents on highest seeds priority level, that would replace searching torrent sites & timing.
14. Indexing is not a crime.
15. Need more reliable torrent uploaders without fakers
16. I don‘t just use the all of the above; there are a few more that are not listed.
17. Regarding question 6: I always start with The Pirate Bay, but if TPB doesn‘t have what I am looking for I move on to other torrent indexes.
18. Most are crap. Some are great!
19. Torrent contents are really useful. We don't need to shut them down due to copy rights violations.
20. prefer private sites (needs login & maintain seed/leech ratio)
21. I use BitTorrent primarily for TV Shows, and I use RSS feeds for automatically download them.
22. It’s just a technology (Peer to peer is awesome). The legal or illegal debate depend on the content we share. That’s it :)
23. Survey does not list many private trackers (though doing such would probably be impractical)
24. in UAE they have blocked KickAss torentz.Before that I have used it most of the time.Then I moved to bit torrent.After using few months I search which is the most populalar torrent client from wikipedea.
25. At the moment my ISP has banned accessing The Pirate Bay. So I have to use proxy to access it.
26. Since North Cyprus is not recognized as a country, it's very easy to pirate, as there are no official rules for online media distribution.
27. Bittorrent search engines are useless and inefficient
28. They're awesome, they're a great way to share so many things, and restrictive bills like SOPA, PIPA, and their counterparts throughout the world, as well as all of the billionaire bullies that support them need to be stopped!
29. Many things that I download are (if they are good) purchased at a later date.
30. Bittorent is good to check if stuff has quality. If it's good it's worth buying.
31. bittorrent is great,it's the perception of a person that matters the most
32. I usually search for things on Demonoid, but then go to The Pirate Bay to actually download them. If I find that a torrent is going slow, I'll search it's hash on Torrenz and add the trackers listed on that site to the tracker list in the torrent.

Appendix II

SIMULATORS

Following is a brief description of simulator designs and configuration parameters. The source code is available in supplementary material.

II.1 Resource Discovery Simulators

Seven discrete-event simulators were developed using Python to demonstrate the architectures listed in Table 4.10. Figure II.1 illustrates the architecture of the simulators. A set of multi-attribute resource and query traces was generated using the method described in Section 4.6 and the same datasets were used with all the simulators. Resources/nodes were sampled every 5 minutes and advertisements were sent to the resource index layer, if the value of at least one attribute was significantly different from the previous advertised value. Thresholds listed in Table II.1 were applied to determine whether attribute values changed significantly. Domain of each attribute is listed in Table II.2. Each resource/node issued queries based on a Poisson distribution with a mean inter-arrival time of 2.5 minutes (i.e., two queries per

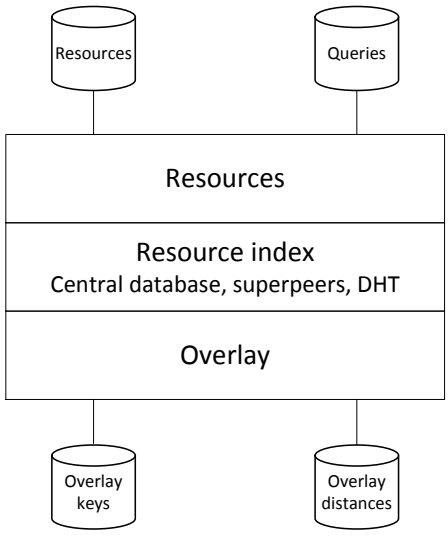


Figure II.1 – Architecture of the resource discovery simulators.

sampling interval per node). Resource index and overlay layers were modified to reflect the appropriate sampling interval per node). Resource index and overlay layers were modified to reflect the appropriate resource discovery solution. A set of topologies were generated a priory and used with appropriate simulators. Both the unstructured and superpeer-based networks were generated using the B-A scale-free network generator [Ge07] with a minimum node degree of two. Structured overlay networks were generated using the Chord [St03] implementation in OverSim [Ba07b], and overlay keys of nodes and the number of hops to send a message from one node to all the other nodes were extracted (this creates the overlay distances dataset). Building the overlay topology outside of the simulator simplified the simulator design, speed up the simulator, and allowed the same topology to be used across different structured P2P-based resource discovery solutions. According to [Co09b], number of cell levels of the *d*-Torus is set to three and nodes in each cell were identified using random sampling. For DHT-based solutions, the overlay key length is set to 160-bits. Attribute values were hashed using the locality preserving hash function proposed in [Ca04]. Queries were issued only after the network was stabilized.

Table II.1 – Thresholds applied while advertising resource attributes.

Attribute Name	Description (Units)	Threshold
1_MIN_LOAD	1-minute EWMA of CPU load	2.0
5_MIN_LOAD	5-minute EWMA of CPU load	2.0
15_MIN_LOAD	15-minute EWMA of CPU load	2.0
CPU_FREE	Free CPU (%)	10.0
DISK_IN	Disk in (GB)	1.0
DISK_FREE	Free disk space (GB)	5.0
DISK_OUT	Disk out (GB)	1.0
DISK_SVC	Disk svc (GB)	1.0
DISK_USED	Disk used (%)	2.0
DISK_UTIL	Disk utilization (%)	2.0
DRIFT	Clock drift (sec)	1.0
MEM_ACTIVE	Memory active (%)	10.0
MEM_FREE	Free memory (%)	10.0
SWAP_IN	Disk swap in (GB)	1.0
SWAP_OUT	Disk swap out (GB)	1.0
SWAP_USED	Disk swap used (%)	5.0
RESP_TIME	Response time (sec)	0.15
RX_RATE	Transmission rate (bps)	1,000
TIMER_AVE	Timer average (milliseconds)	100.0
TIMER_MAX	Timer maximum (milliseconds)	100.0
TX_RATE	Receive rate (bps)	1,000
UPTIME	Uptime of node (sec)	21600.0

Table II.2 – Domains of attribute values.

Attribute Name	Description (Units)	Minimum	Maximum
1_MIN_LOAD	1-minute EWMA of CPU load	0.0	1000.0
5_MIN_LOAD	5-minute EWMA of CPU load	0.0	1000.0
15_MIN_LOAD	15-minute EWMA of CPU load	0.0	1000.0
BOOT	Boot state	0	6
BW_LIMIT	Bandwidth limit (bps)	0	1,000,000
CORES_PER_CPU	No of cores per CPU	0	8.0
CPU_FREE	Free CPU (%)	0.0	100.0
CPU_SPEED	CPU speed (GHz)	0.0	6.0
DISK_IN	Disk in (GB)	0.0	50,000.0
DISK_FREE	Free disk space (GB)	0.0	10,000.0
DISK_OUT	Disk out (GB)	0.0	50,000.0
DISK_SIZE	Disk size (GB)	0.0	10,000.0
DISK_SVC	Disk svc (GB)	0.0	10,000.0
DISK_USED	Disk used (%)	0.0	100.0
DISK_UTIL	Disk utilization (%)	0.0	100.0
DRIFT	Clock drift (sec)	21,600.0	21,600.0
FC_NAMEX	OS (Fedora core) name	0	4
KERN_VER	Kernel version	0	4
LATITUDE	Latitude of node location (degrees)	-90.0	90.0
LOCATION	Location of node (categorical)	1	5
LONGITUDE	Longitude of node location (degrees)	-180.0	180.0
MEM_ACTIVE	Memory active (%)	0.0	100.0
MEM_FREE	Free memory (%)	0.0	100.0
MEM_SIZE	Memory size (GB)	0.0	32.0
NODE_TYPE	Node type (categorical)	0	2
NUM_CORES	No of CPU cores	0	32.0
SWAP_IN	Disk swap in (GB)	0.0	2500.0
SWAP_OUT	Disk swap out (GB)	0.0	2500.0
SWAP_USED	Disk swap used (%)	0.0	100.0
RESP_TIME	Response time (sec)	0.0	100.0
RX_RATE	Transmission rate (bps)	0	100,000
TIMER_AVE	Timer average (milliseconds)	0.0	25,000.0
TIMER_MAX	Timer maximum (milliseconds)	0.0	25,000.0
TX_RATE	Receive rate (bps)	0	100,000
UPTIME	Uptime of node (sec)	0.0	6,3072,000.0

II.2 ResQue – Resource and Query Generator

ResQue is developed using MATLAB. Figure 5.18 illustrates the process of generating multi-attribute resources by combining the empirical-copula-based static attribute generation and time-series-library-based dynamic attribute generation. *pwlCopula* [St09] MATLAB tool is extended to run in the background (without user interaction) and integrated into ResQue. Time series libraries are built using the

strucchange package for *R* and two-halve-window-based derivative filter. Minimal segment size (i.e., minimum gap between two structural changes) is set to 6 hours. PlanetLab queries were preprocessed first to identify the state transitions. These state transitions were used to build the probabilistic finite state machine. See ResQue user guide [CNRL] for more details on data formats and configuration parameters.

II.3 Resource and Query Aware Resource Discovery Simulator

A discrete-event simulator was developed using Python. The architecture of the simulator was same as Fig. II.1. However, the overlay network was built within the simulator, as it needs to be adaptable to the resource and query distributions. Workloads with 100,000 resources were simulated by representing a node in the system as 20 virtual nodes (altogether 5,000 nodes were used) with different identifiers. This was acceptable, as only a few hundred nodes were added to the overlay based on the resource and query distributions. Nodes in the overlay rings acted as a set of proxies for rest of the nodes. A node connected to a randomly selected proxy every 3 minutes or after removing a node from the overlay. Finger tables were updated every 3 minutes to maintain a stable Chord overlay. The length of an overlay key is set to 32-bits, as only a few hundred nodes are added to the overlay ring. Attribute values were hashed using the locality preserving hash function proposed in [Ca04]. The resources were mapped to a node within a clique based on the hash value of resource identifier (SAH1 was used as the hash algorithm). When a node is added/removed to/from a clique, resource index was rearranged within the clique by splitting the address space uniformly among all the nodes in the clique. Packet sizes were calculated based on the size of an IPv4 address and a port number (6 bits), and the number of (attribute, value) pairs (4 bytes per pair) it carries. The maximum packet size was set to 1,500 bytes. Workloads and node capacities are described in Table 6.2 and Section 6.5. Queries were issued only after the network was initially stabilized. Once the resources are indexed and queries are issued network may change in response to their loads.

II.4 Community-Based Caching

II.4.1 Local-Knowledge-Based Distributed Caching and PoPCache Simulators

First, a set of lookup query traces was extracted using the Chord [Sa03] implementation in OverSim [Ba07b] version 20100526 under different random seeds. A python-based simulator was then developed to replay the query traces while assigning cache entries to nodes based on the proposed LKDC algorithm (Fig. 7.7) and PoPCache [Ra07]. Number of nodes in the overlay network, the cache capacity of a node, and Zipf's parameters were varied as discussed in Sections 7.5 and 7.6.1.

II.5.2 Community-Based Caching Simulator

We simulated the community-based caching solution using a 15,000-node network with ten communities using OverSim [Ba07b] version 20100526. The architecture of the solution is illustrated in Fig. II.2. Lists of keys of resources and queries for those keys were generated outside of the simulator. A separate query file was generated for each community as described in Table 7.4. Such a design simplified the implementation while enabling us to reuse the same key and query traces under different simulation

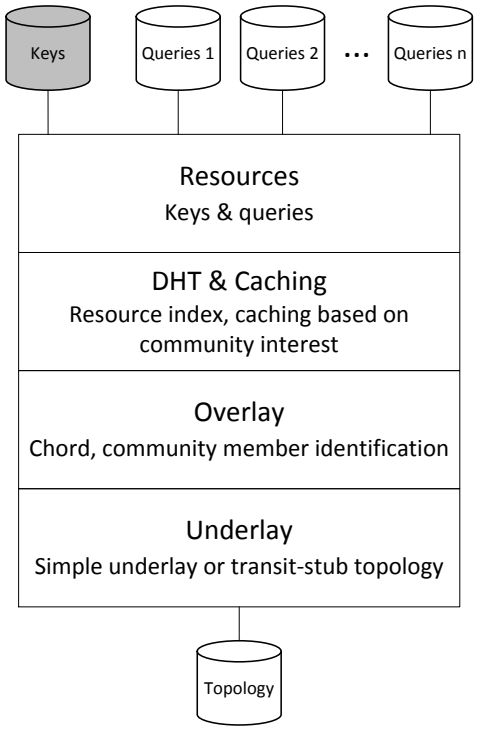


Figure II.2 – Architecture of community-based caching simulator.

parameters.

Multiple key and query traces were generated using different random seeds. Similarity among queries issued by communities was enforced by sharing a subset of the queries (size of the subsets depends on the cosine similarity between the two communities) issued by one community with another. To measure the ability of geographic communities to improve the latency, transit-stub networks with 10 Autonomous Systems (ASs) and 750 routers were generated using BRITE [Me01] while using GT-ITM [Ze96] as the underlying topology generator. Based on [Ca02], overlay node-to-router delay is set to 1 ms and the average delay of the core network links is set to 40 ms. First, the shorted path from each router to all the other

Table II.3 – Simulation parameters for community-based caching.

Parameter	Description (Units)	Value
**.measurementTime	Simulation time (sec)	6,000
**.delayToStart	Delay before keys are indexed (sec)	1,000
**.overlay*.myChord.joinDelay	Delay before joining overlay (sec)	10
**.overlay*.myChord.stabilizeDelay	How frequently to issue overlay stabilize messages (sec)	30
**.overlay*.myChord.fixfingersDelay	How frequently to issue fix finger messages (sec)	150
**.overlay*.myChord.checkPredecessorDelay	How frequently to check the predecessor (sec)	10
**.overlay*.myChord.successorListSize	Type of successor list	1
**.overlay*.myChord.aggressiveJoinMode	Join overlay aggressively	true
**.overlay*.myChord.extendedFingerTable	Use extended finger table	true
**.overlay*.myChord.numFingerCandidates	No of candidates for each finger	1
**.overlay*.myChord.findGroupMembers	Find community members	true
**.overlay*.myChord.givePriorityToGroup	Give priority to community members while finding the next hop to forward a message	false
**.overlay*.myChord.maxFindGroupMemberHops	No of hops to forward a community member discovery message	4
**.overlay*.myChord.proximityRouting	Route based on latency to next hop	false
**.routingType	Routing type	semi-recursive
**.tier2.myDhtTestApp.testInterval	Inter arrival time for queries (sec)	15
**.tier2.myDhtTestApp.maxGroupId	Number of groups/communities	10
**.tier1*.myDht.numReplica	No of replicas in DHT	1
**.tier1*.myDht.numGetRequests	No of get() requests/messages per query	1
**.tier1*.myDht.cacheRefreshTime	Gap between two cache clean ups	0 (no cleaning)
**.tier1*.myDht.maxCacheSize	Cache size	0-28
**.tier1*.myDht.useMsgGroupId	Indicate community ID in get() messages	false
**.tier1*.myDht.alpha	Caching weight (θ) used in LKDC algorithm. In the simulators it is called α.	0.1-0.5
**.tier1*.myDht.cachingThreshold	Caching threshold	$\alpha + 0.02$
**.tier1*.myDht.removeLookupThreshold	Cache entry remove threshold	$(1 - \alpha)^{10}$
**.targetOverlayTerminalNum	No of nodes	1,000-15,000
**.initPhaseCreationInterval	Time delay between addition of two nodes (sec)	0.1

routers and the latency of each route was calculated and dumped to a file. Second, a set of lookup query traces (including the entire path taken by a query), nodes, and their community IDentifiers (IDs) were then extracted while simulating community-based caching. Third, overlay nodes were randomly mapped to the underlay topology generated using BRITE based on their community ID such that nodes in the same community are assigned to the same AS. Finally, the latency for each lookup query was calculated by aggregating the underlay latency for each hop along the overlay. Simulation parameters are listed in Table II.3.

II.5 Named Data Networking for Distributed Multi-Sensor Data Fusion

II.5.1 Multi-Sensor Data Fusion Simulator

A discrete-event simulator is developed in Python and its architecture is depicted in Fig. II.3(a). Section 8.4 explains how the radars, applications, proxies, and end users are placed within the 2D CAN overlay, and queries are generated. Simulation parameters are listed in Table II.4.

II.5.2 Event-Specific Query Simulator

A discrete-event simulator is developed in Python and its architecture is depicted in Fig. II.3(b).

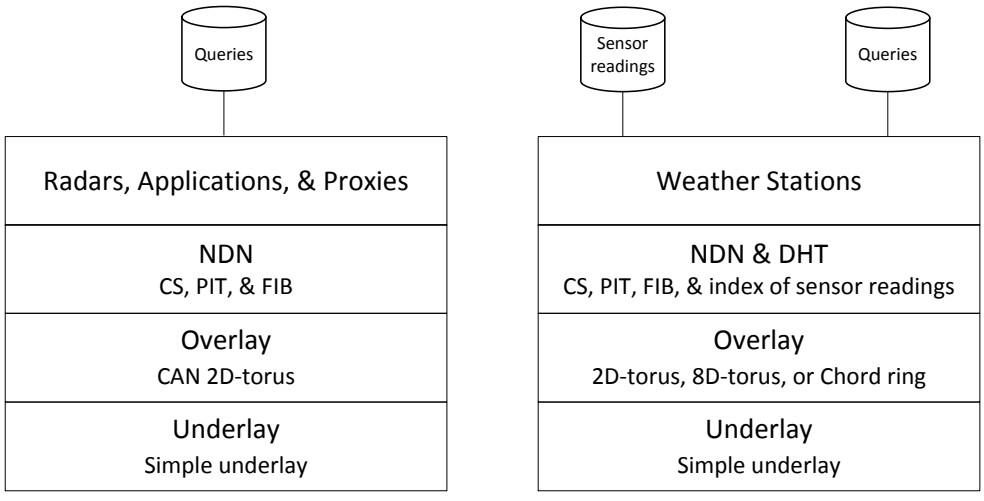


Figure II.3 – Architectures of NDN for DCAS simulators: (a) Multi-sensor data fusion; (b) Event-specific query resolution.

Section 8.4 explains how the sensor readings are collected from weather stations and placement of end users. While advertising sensor readings, thresholds listed in Table II.5 are applied to reduce the number of advertisements due to minor changes in sensor readings. However, sensor readings were advertised at least every 30 or 60 minutes (depending on the sampling interval) even if their values do not change significantly. Domains of sensor readings are listed in Table II.6. Simulation parameters are listed in Table II.4.

Table II.4 – Simulation parameters for NDN for DCAS simulators.

Parameter Name	Description (Units)	Value
CACHING_POLICY	Caching policy	Oldest, LRU, LFU
PIT_TIMEOUT	PIT entry timeout (sec)	120
SIZE_CS	Content store capacity (bytes)	0-100 MB
SIZE_PIT	PIT capacity (bytes)	Unlimited (0)
SIZE_FIB	FIB capacity (bytes)	Unlimited (0)
NUM_DIMENSIONS	No of dimensions of the torus	2, 8
SIZE_TORUS[ ]	Size of each dimension (m)	1,000 km
NUM_BITS	Resolution of an axis on torus. Determine no of segments the torus is split into while using space-filling curves. (bits)	4
NUM_RADARS_X	No of radars along X-axis	11
NUM_RADARS_Y	No of radars along Y-axis	11
RADAR_RANGE	Range of a radar (m)	40,000.0
INTER_RADAR_DIST	Gap between 2 radars (along X & Y arises) (m)	30,000.0
RADAR_SKEW	Maximum time difference between 2 radars' data generation time (sec)	30
RADAR_HEARTBEAT	Heartbeat interval of a radar (sec)	30
NUM_APP_IN_SUB_DFS	No of apps in subset of the data fusion groups	4
NUM_APP_PROXIES	No of proxies	5
APPS_IN_ALL_DFS	Applications in all the data fusion groups. REFL – reflectivity, DDOP – dual Doppler	['REFL', 'DDOP']
APPS_IN_SUB_DFS	Applications in subset of the data fusion groups. QPE - Quantitative precipitation estimation, NBRR – network-based reflectivity retrieval, NCAS – nowcasting	['QPE', 'NBRR', 'NCAS']
TILE_X_PROXY	Length of the smallest area of interest along x-axis (at proxy) (m)	6,000
TILE_Y_PROXY	Length of the smallest area of interest along y-axis (at proxy) (m)	6,000
TILE_X_RADAR	Length of the smallest area of interest along x-axis (at radar) (m)	500
TILE_Y_RADAR	Length of the smallest area of interest along y-axis (at radar) (m)	500
JOIN_DELAY	Delay between 2 nodes that join the network (sec)	0.5
ADD_RADAR_TO_OVERLAY	Are radars part of the overlay network	False
PIXEL_DATA_SIZE	Number of bytes generated for the smallest tile at a radar. 4 × no tiles along x-axis × no tiles along y-axis. (bytes)	100
USE_SUBSCRIPTIONS	Use query subscriptions	True/False
FIRST_QUERY	When to issue first query (sec)	300
SUB_EXPIRE_TIME	Subscriptions expire after this time (sec)	720
NUM_W_STATIONS	No of weather stations	1,081
FRACTION_IN_OVERLAY	Fraction of weather stations in overlay	0.1

Parameter Name	Description (Units)	Value
SENSOR_NAMES	List of names assigned to sensors. X – longitudes, Y – latitudes, DD – wind direction, ELEV – elevation, FF – wind speed, P – pressure, RH – relative humidity, and T – temperature.	'X', 'Y', 'DD', 'ELEV', 'FF', 'P', 'RH', 'T
MIN_VALID_TIME	Sensor data should at least expire after this time (sec)	1,800
USE_THRESHOLDS	Apply thresholds to prevent advertising of minor changes in sensor readings	True
SIZE_SENSOR_READINGS	No of bytes required to represent sensor readings. 1 byte for sensor name & 4 bytes for sensor reading.	5 × no of sensor names.
USERS	End user placement within sensor field. Total users of given type = i * j (if [i, j]) or i if ([i]). NWS – national weather service, EM – emergency managers, RES – researchers, and MED – media.	['NWS', [6, 5]], ['EM', [20, 16]], ['RES', [30]], ['MED', [120]]
JOIN_DELAY	Delay between 2 nodes that join (sec)	0.1
SIM_TIME_AFTER_LAST_QUERY	Simulation time after last query. Defines the simulation end time.	150
BANDWIDTH	Bandwidth (bps)	1 Gbps
PREFETCH_TIME	Time to pre-fetch weather station data & queries (sec)	180
PREFETCH_INTERVAL	How frequently to pre-fetch (sec)	120
SPEED_LIGHT	Speed of light. Use to calculate latency while transferring packets. (m/sec)	299,792,458
FIX_FINGER_INTERVAL	When to fix fingers (sec). Used only for Chord ring.	120
KEY_LENGTH	Overlay key length. Used only for Chord ring.	32

Table II.5 – Thresholds applied while advertising sensor readings.

Attribute Name	Description (Units)	Threshold
DD	Wind direction (deg)	5.0
FF	Wind speed (m/s)	1.5
ELEV	Elevation. Threshold is used as ELEV is a float value	0.0001
P	Station pressure (Pa)	1,000.0
T	Air temperature (K)	1.0
RH	Relative humidity (%)	5.0
X	X coordinate. Threshold is used as X is a float value	0.0001
Y	Y coordinate. Threshold is used as Y is a float value	0.0001

Table II.6 – Domains of sensor readings.

Attribute Name	Description (Units)	Minimum	Maximum
DD	Wind direction (deg)	0.0	360.0
ELEV	Elevation (m)	0.0	2,000.0
FF	Wind speed (m/s)	0.0	25.0
FFGUST	Wind gust (m/s)	0.0	90.0
P	Station pressure (Pa)	25,000.0	110,000.0
T	Air temperature (K)	243.0	320.0
RH	Relative humidity (%)	0.0	100.0
X	X coordinate. Threshold is used as X is a float value	0	1,000 km
Y	Y coordinate. Threshold is used as Y is a float value	0	1,000 km

ABBREVIATIONS

AOI	Area Of Interest
API	Application Programming Interface
AS	Autonomous System
ATM	Automated Teller Machine
CAN	Content Addressable Network
CASA	Collaborative Adaptive Sensing of the Atmosphere
CBC	Community-Based Caching
CCN	Content Centric Networking
CDF	Cumulative Distribution Function
CID	Community IDentifier
CPU	Central Processing Unit
CS	Content Store
CSU	Colorado State University
DCAS	Distributed Collaborative Adaptive Sensing
DF	Data Fusion
DFG	Data Fusion Group
DHT	Distributed Hash Table
DLC	Distributed Local Caching
EGI	European Grid Infrastructure
EM	Emergency Managers
ERD	Efficient Resource Discovery
EWMA	Exponentially Weight Moving Average
FIB	Forwarding Information Base

FIFO	First In First Out
FIR	Finite Impulse Response
FOSS	Free and Open Source Software
FPGA	Field-Programmable Gate Array
GCO	Green Computing Observatory
GEV	Generalized Extreme Value distribution
GENI	Global Environment for Network Innovations
GIS	Geographic Information Systems
GKDC	Global-Knowledge-based Distributed Caching
GPD	Generalized Pareto Distribution
GPS	Global Positioning System
GPU	Graphic Processing Unit
IaaS	Infrastructure as a Service
i.i.d.	Independent and identically distributed
I/O	Input/Output
IP	Internet Protocol
IPTV	Internet Protocol television
ISP	Internet Service Provider
KKT	Karush–Kuhn–Tucker
KS	Kolmogorov-Smirnov
LFU	Least Frequently Used
LKDC	Local-Knowledge-based Distributed Caching
LORM	Low-Overhead, Range-query, and Multi-attribute
LPH	Locality Preserving Hash
LRU	Least Recently Used
MAAN	Multi-Attribute Addressable Network

MC&C	Meteorological Command and Control
MINLP	Mixed Integer NonLinear Programming
MIPS	Million Instructions Per Second
MURK	MUlti-dimensional Rectangulation with Kd-trees
NB	Negative Binomial distributions
NBRR	Network-Based Reflectivity Retrieval
NDN	Named Data Networking
NEXRAD	Next Generation Weather Radar
NLIP	NonLinear Integer Programming
NP	Nondeterministic Polynomial time
NWS	National Weather Service
OFC	Oldest First Caching
ORT	Overlay Routing Tree
P2P	Peer-to-Peer
PaaS	Platform as a Service
PFSM	Probabilistic Finite State Machine
PIT	Pending Interest Table
QoE	Quality of Experience
QoS	Quality of Service
QPE	Quantitative Precipitation Estimation
RD	Resource Discovery
RS	Resource Specification
RST	Range Search Tree
RTT	Round Trip Time
SaaS	Software as a Service
SADQ	Single-Attribute Dominated Querying

SETI	Search for Extraterrestrial Intelligence
SHA	Secure Hash Algorithm
SLA	Service Level Agreement
SFC	Space-Filling Curve
TLS	T Location-Scale distributions
TTL	Time To Live
URL	Uniform Resource Locator
VM	Virtual Machine
VoIP	Voice over Internet Protocol
ZC	Zone Controller