THESIS

ENABLING AUTOSCALING FOR IN-MEMORY STORAGE IN CLUSTER COMPUTING FRAMEWORK

Submitted by

Bibek Raj Shrestha

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2019

Master's Committee:

Advisor: Sangmi Lee Pallickara

Shrideep Pallickara
Stephen C. Hayne

ABSTRACT

ENABLING AUTOSCALING FOR IN-MEMORY STORAGE IN CLUSTER COMPUTING FRAMEWORK

IoT enabled devices and observational instruments continuously generate voluminous data. A large portion of these datasets are delivered with the associated geospatial locations. The increased volumes of geospatial data, alongside the emerging geospatial services, pose computational challenges for large-scale geospatial analytics. We have designed and implemented STRETCH, an in-memory distributed geospatial storage that preserves spatial proximity and enables proactive autoscaling for frequently accessed data. STRETCH stores data with a delayed data dispersion scheme that incrementally adds data nodes to the storage system. We have devised an autoscaling feature that proactively repartitions data to alleviate computational hotspots before they occur. We compared the performance of STRETCH with Apache Ignite and the results show that STRETCH provides up to 3 times the throughput when the system encounters hotspots. STRETCH is built on Apache Spark and Ignite and interacts with them at runtime.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Sangmi Lee Pallickara, for her enthusiastic support and sincere guidance that have resulted in driving this research project to completion. I would also like to thank my thesis committee members, Dr. Shrideep Pallickara and Dr. Stephen C. Hayne, for their constructive suggestion and remarks. Similarly, I would like to thank my colleague Mr. Saptashwa Mitra for his contribution in this research work. Finally, I would like to thank Ms. Wendy Stevenson and everyone in CS department for creating such a warm and friendly environment in the building.

DEDICATION

I would like to dedicate this thesis to my wife Melissa.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

Advances in remote sensing and IoT technologies allow scientists and researchers to analyze voluminous geospatial data to understand natural and social phenomena. This growth in geospatial data has resulted in the steady increase in the needs for location-based services. For example, in 2018, 242 million users were accessing location-based services [1] on their mobile devices, which was more than two-fold increase over 2013 [2].

To explore voluminous spatial data that is updated at high frequencies, two aspects need to be considered: effective storage and adaptive computing capabilities. In-memory computing over distributed clusters using commodity machines such as Apache Spark [3] is rapidly becoming popular because of its speed, adaptability, and high analytical throughput. This is achieved using a data model called resilient distributed datasets (RDDs) that are stored in memory while being used for the computations instead of loading from the disk. It is tightly integrated with the job execution engine that can optimize computation. Recently, there have been various efforts to enhance in-memory computing capabilities. As for example, Alluxio [4, 5] is an in-memory storage system that targets high throughput writes and reads over Spark without compromising fault tolerance. Apache Ignite [6] provides in-memory storage and computing framework that is compatible with the Spark computations. Similarly, Simba [7] extends the Spark SQL engine to support geospatial dataset with advanced spatial operations.

Large-scale geospatial analytics pose interesting challenges for distributed in-memory analytics. First, the nature of spatial interaction in societal and natural phenomena increases the probability of involving spatially proximate data points with the targeted analytics task. We posit that grouping data points that are geospatially proximate is a fundamental requirement in reducing data movements within the cluster. Second, users' interests are often focused within areas with highlighted events such as national disasters (e.g. earthquakes or hurricanes) or events (e.g. elec-

tions, Olympic Games). Within a shared data hosting environment, this may cause hotspots for computations and data access.

Most existing approaches such as Alluxio and Ignite lack data dispersion schemes that have spatial proximity and autoscaling features. Although there have been approaches such as Simba and Magellan [8] that accounts for the geospatial proximity, they do not cope with unevenly distributed workloads. Indeed, there is a significant gap between the autoscaling mechanism provided by schedulers (that relies on the memory assigned for each executor) and computing with in-memory data storage (stored in the main memory of worker nodes so that executors can share).

To address these aforementioned challenges, we have designed and developed STRETCH, an adaptive auto-scalable in-memory distributed data storage based on storage and computing workloads. STRETCH is built on top of Apache Ignite. It disperses and indexes geospatial data while preserving geospatial proximity to reduce the data movements within the cluster. It tracks resource utilization and data access patterns to proactively scale data nodes before the scheduler scales up the computing resources while preserving data locality.

1.1 Research Questions

Research questions that we explore in this study include the following:

- **[RQ1]** How can we provide a scalable in-memory data structure that can be shared by large number of tasks and jobs?

  Specifically, the data structure must interact with the analytics jobs and their sub-tasks while striking a balance between effective memory utilization and reducing disk I/O (Chapter 3.1).

- **[RQ2]** How do we disperse voluminous data over computing cluster without causing excessive data movements?

  The data dispersion scheme must provide searchable environment to retrieve proximate data points while reducing network I/O over the cluster (Chapter 3.1).

- **[RQ3]** How can the in-memory data structure alleviate possible computing hotspots in the cluster?

  Unlike leveraging existing in-memory database, the data storage must be able to dynamically adapt itself to hotspots during data accesses (Chapter 3.2).

1.2 Overview of Approach

STRETCH is an in-memory distributed data storage for geospatial analytics with adaptive autoscaling capability. The geospatial dataset is stored in a multi-tier distributed hash table [9] comprising the main memories of multiple subgroups of nodes. Each subgroup manages datasets associated with a geospatial coverage to preserve the spatial proximity using geohash algorithm [17]. STRETCH provides a data discovery interface for the scheduler to locate a corresponding node for the computation. Once the dataset is loaded into STRETCH, the subsequent computations are automatically hosted by the node that stores dataset in its memory.

We have designed and developed an autoscaling scheme that proactively enhances the storage and computing capability of each sub-cluster to alleviate the memory and computation hotspots caused by evolving geospatial area of interests. Each sub-cluster has a master that monitors system utilization of its worker nodes in order to detect potential hotspots. Once the hotspot is identified, the master identifies available node from within or other subgroups and employs them to scale out the current hotspot node.

In this study, we propose a delayed data dispersion scheme to adapt the traditional Distributed Hash Table (DHT) for more effective task/data migrations. In this scheme, the number of active nodes that host data and computing is increased incrementally only when the workload exceeds the computing and storage capacity of current active nodes. This allows STRETCH to identify idling nodes easily and perform smaller number of migrations by leveraging sufficient resources from the idle nodes.

1.3 Research Contributions

This research presents our approach for providing a dynamically adaptive in-memory data storage that is tightly coupled with distributed computing cluster to perform a large number of analytics computations for geospatial datasets effectively.

(1) We have designed an in-memory hierarchical DHT that is shared by jobs and their tasks. This data structure allows jobs and tasks to query and re-use the previously loaded data points by other tasks. We are particularly targeting geospatial datasets, and the data is dispersed based on the geospatial proximity of the dataset.

(2) Computations are co-hosted over the STRETCH nodes and the system tracks the computing requirements per node. For the nodes loaded with a high number of analytics jobs, STRETCH provides an adaptive dispersion scheme that allows the jobs to temporarily utilize nodes that are not the original hosts of the targeted data portion.

(3) Our methodology was implemented using the Apache Ignite distributed computing environment. STRETCH supports existing APIs to provide ease-of-use for the existing users. This includes IgniteRDD API that can be used to co-host Spark cluster alongside STRETCH cluster and make use of rich computing libraries that Spark provides.

1.4 Organization

The rest of the document is organized as follows. Chapter 2 describes related work in the context. The first section of our methodology in Chapter 3 describes our delayed data partitioning scheme while the second section describes our autoscaling feature. Chapter 4 go over system architecture that extends our methodology. In Chapter 5, we go over experiments and evaluation of our system against Ignite. Finally, our conclusions are described in Chapter 6.

Chapter 2

Related Work

2.1 In-memory Cluster Computing Frameworks

Apache Spark [3] is a general-purpose, highly efficient large-scale data analytics engine with rich sets of libraries for structured queries, machine learning, streaming analytics, etc. Spark has evolved to become one of the most popular solutions in research and industry. It uses an efficient in-memory fault tolerant data abstraction called Resilient Distributed Datasets (RDDs). Each RDD is a distributed collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. While native Spark RDDs cannot be shared across Spark jobs or applications, STRETCH provides a partitioned view of the distributed store such that it could be used across different Spark jobs, workers, or applications.

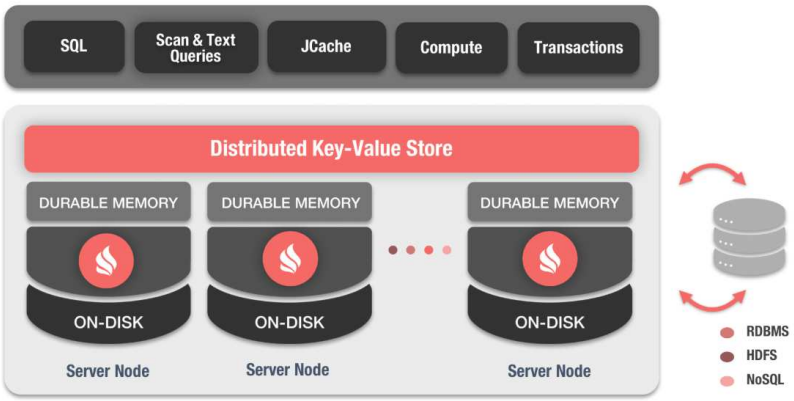


Figure 2.1: Ignite's Data Grid.

Similarly, another popular in-memory solution is Apache Ignite [6], which is a memory-centric distributed database, caching, and processing platform. As depicted in the Figure 2.1, Ignite can be considered a distributed partitioned hash map with every cluster node owning a portion of the overall data set. As is standard in any large-scale engines, Ignite supports collocation of computations with data and minimizes data serialization within network and significantly improve performance

and scalability. STRETCH is implemented on Ignite because of its lightweight indexing scheme and highly available caching platform. STRETCH supports all the existing Ignite API of which Ignite's API to connect to Spark is of interest in our work. Ignite provides an implementation of the Spark RDD which allows any data and state to be shared in memory as RDDs across Spark jobs. IgniteRDD provides a shared, mutable view of the data stored in Ignite's hash table across different Spark jobs, workers, or applications. IgniteRDD is implemented is as a view over a distributed Ignite table. Ignite platform has its own library for basic spatial use-cases that allows querying and indexing geometry data types such as points, lines, and polygons considering the spatial relationship between these geometries. It uses MVR-Tree [10] for spatial indexing. However, STRETCH has its own indexing scheme that is well coupled with geohash on which our whole system is built on.

2.2 Scalable Geospatial Analytics

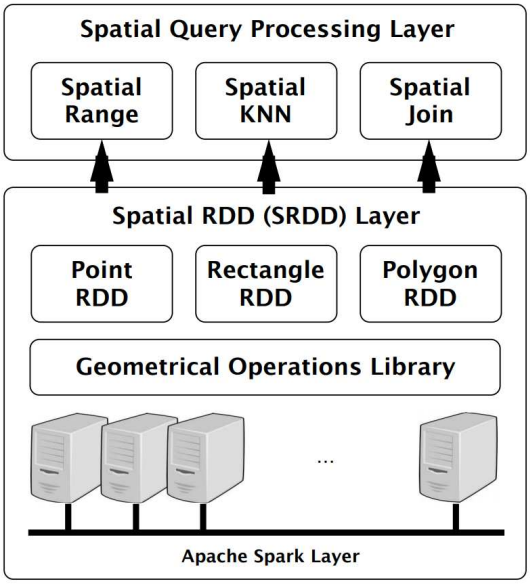


Figure 2.2: Geospark Overview.

With the advances in-memory distributed analytics engines and availability of cloud solutions, there have been dramatic increase in large-scale data analysis. Computing systems like Hadoop

[11] and Spark have revolutionized the big data domain. In this context, one of the most popular categories of big data is geospatial. Therefore, there exist numerous systems that extend the above mentioned data analytic systems for spatial use-cases. Some of the popular systems in this category are Magellan, Simba, GeoSpark [12], GeoMesa [13], etc. all of which extend Spark. These systems essentially indexes the Spark's RDD to suite for spatial queries. The Figure 2.2 gives an overview of one of the systems, namely Geospark, that extends the Spark's core for running spatial queries.

Similarly, there are many systems that extend Hadoop core API to make it suitable for spatial datasets. As evident, there have been extensive works on developing efficient indexing strategies for spatial datasets such as in [14–16]. Also, there are various examples of uses of hash based index such as geohash for effective spatio-temporal disk storage in earlier works such as Galileo [17].

Like Magellan and GeoMesa, STRETCH makes use of geohash to preserve geospatial proximity. Unlike any existing systems, however, STRETCH's indexing scheme is dynamic and is resized on-the-fly in response to hotspots prevalent in the domain of geospatial analytics.

Similarly, there have been several approaches to effective scientific data management [18–22]; several of these involve effective leveraging metadata. Approaches to spatiotemporal data management include Galileo with support for proximity queries [23, 24], ad hoc queries [25], controlled dispersion and collocation [26]. Often how data is managed underpins the effectiveness of analytic operations. The Hadoop Distributed File System supports interoperations with several ecosystems such as Hadoop [11], Mahout [27], and Spark [3] among others. Other approaches to analytic operations closely coupled with storage operations include support for ensemble methods [28] and analytic queries [29, 30]. Another approach to cope with differences in the memory hierarchy include approaches based on sketching [31, 32]. Analytic tasks may involve complex scheduling decisions [33, 34].

2.3 Distributed Key-Value Store

The distributed key-value storage are very popular in many large-scale in-memory caching and memory-centric solutions. They are very efficient, highly scalable, lightweight and easy to

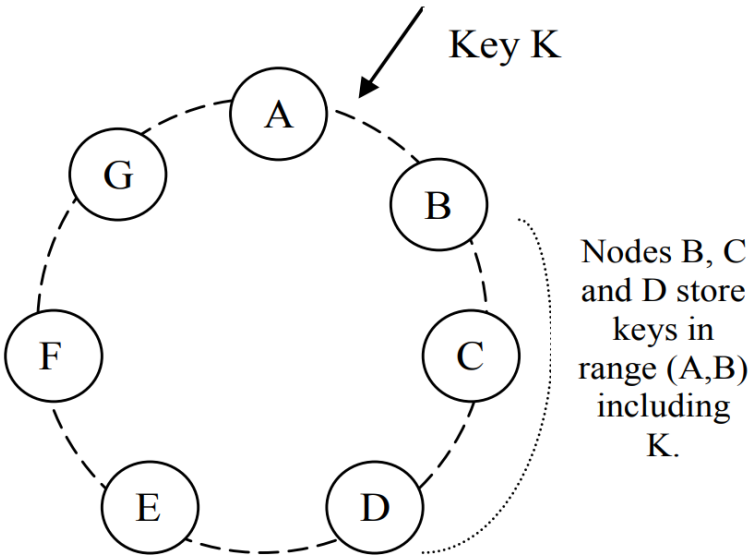


Figure 2.3: Partitioning scheme in Dynamo.

maintain. The core idea behind distributed key-value store is using hash function to uniformly partitioning the keys among the nodes. Figure 2.3 shows the partitioning scheme in one of the popular DHTs, namely Dynamo [35], which is extensively used in Amazon. Some of the very popular systems such as Memcached [36], Ignite, etc. are effectively in-memory key-value store. There have been extensive research in this domain [37–40]. The key-value systems have proven very effective in scalable, petascale analytics.

Because of the above mentioned reasons, STRETCH is implemented as a DHT and leverages the key-value philosophy coupled with effective resource management. This is also essential to support STRETCH's autoscaling feature as addition and/or removal of nodes in the cluster is seamless in a DHT without causing much overhead and reshuffle.

2.4 Distributed Load Balancing

As the cloud solutions are becoming highly scalable, cost-effective and flexible, there has been huge shift towards cloud-based infrastructure in recent time. However, heterogeneity of cloud infrastructure and uneven workload culminates in ineffective use of cloud platform thus causing performance degradation and cost-inefficiency. To that extent, there have been extensive works on

addressing the workload imbalance and effective load balancing [41–44]. Caching solution such as [44] repartitions hot partitions every 12 hours to uniformly distribute the workload. However, for geospatial workloads where the skewness has a very short attention span as it is related to natural and social phenomena that are constantly changing, such periodic reshuffling approach is not effective to address load imbalance. There has been work on dynamic hotspot mitigation in cloud storage [41]. This work is based on basic data read and write queries and addresses the issue of heterogeneity of cloud cluster for load redistribution. Similarly, there are proprietary solutions such as Databricks' Optimized AutoScaling [45] which dynamically autoscale Spark jobs in the cluster at runtime.

A point to note here is that there isn't any existing solution that addresses load imbalance in the domain of geospatial analytics dynamically in cloud platform. This is where STRETCH comes in. It addresses both storage and computation hotspots which preserving geospatial proximity which is very essential for any geospatial-based services.

Chapter 3

Methodology

The topology of the STRETCH cluster is a DHT that includes multiple sub-clusters representing geospatial areas. Each sub-cluster consists of a master node and one or more worker nodes. The master node manages metadata and tracks the resource utilization of each worker node in the sub-cluster. More detailed discussion about the system architecture is available in Chapter 4.

Applications can store and retrieve data using the STRETCH APIs. To store a data object, users can use the `put(K,V)` in their client codebase. Based on the geospatial and temporal information of the data object, the STRETCH API generates a `Key` that is a combination of the geohash of the location and temporal information. To retrieve the data, applications should use the same `Key` to return the matching `Value`.

3.1 Delayed Geospatial Data Partitioning (RQ1 and RQ2)

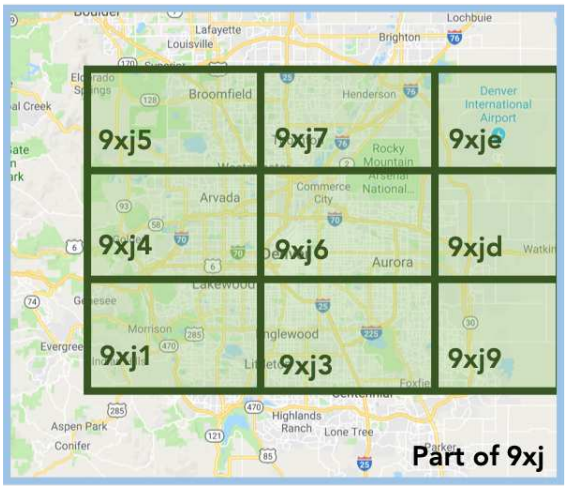


Figure 3.1: Hierarchical Spatial Bounding Boxes with Geohash.

Grouping stored data points with geospatial proximity is critical to reduce potential data movement in the cluster. STRETCH partitions and disperses data using the geohash algorithm. It gen-

erates a spatially-constrained hash space that identifies locations on Earth as Base 32 strings. As a result, Geohash creates a hierarchy of spatial bounding boxes where the hashes sharing a partial prefix represents similar spatial locations. In STRETCH, 12 characters geohash is used which has the resolution of 3.7cm x 1.9cm rectangle at equator. As seen in Figure 3.1, geohashes 9xj1, 9xj3, 9xj7 and 9xj6 are all representing bounding boxes in Denver, Colorado. To support various resolutions, the final hash value can select the length of a geohash prefix. Short prefixes specify larger spatial regions and longer strings define smaller spatial regions. In our previous example, the bounding boxes 9xj6 and 9xj7 are included within the larger bounding box 9xj.

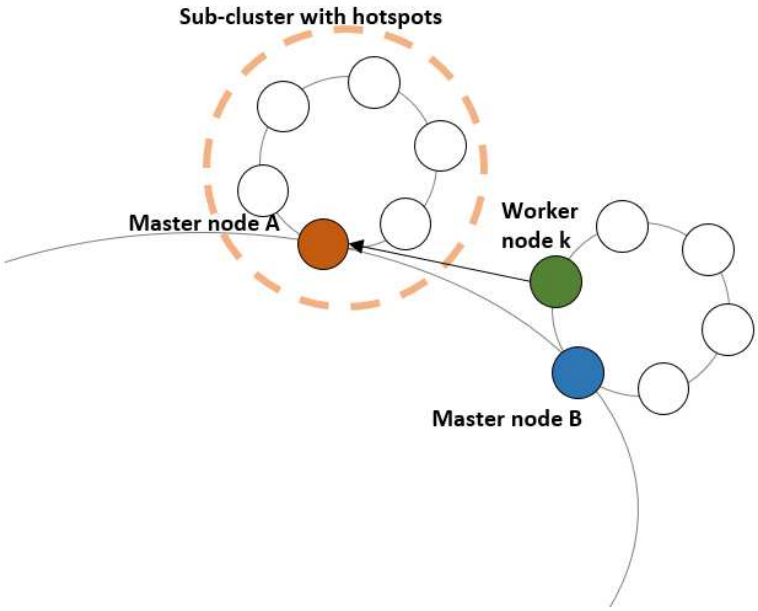


Figure 3.2: STRETCH autoscaling sub-clusters.

STRETCH assigns unique geohash to each sub-cluster, which covers a relatively larger area. However, the coverage for a single worker node is not determined when the STRETCH cluster is initialized. Instead, it is dynamically adjusted depending on the workloads within the sub-cluster. As depicted in the Figure 3.2, a sub-cluster contains a master node and worker nodes. In contrast to other key-value storage systems, STRETCH incrementally involves nodes based on the level of demand and resource utilization. We call this process delayed data partitioning.

A node with excessive computing requirements or memory utilization will be considered a resource hotspot. Initially, data is stored only in the master nodes until a master node identifies itself as a resource hotspot. Finally, the master will incrementally involve worker nodes in its sub-cluster. Since the STRETCH cluster will be co-hosted with a Spark cluster, we consider both memory and CPU utilization based on jobs to identify hotspots.

Based on the distributions of workloads, the master nodes will determine how they will migrate current data. To understand the distribution of workloads, the master nodes split their coverage to 32 sub-blocks represented by 32 possible geohash characters and also maintain a routing table to track the frequency of data access per sub-block. We achieved this resolution by adding another character to the geohash string. To consider only recent activities, we track access for last 10 minutes only. This value is configurable.

If a master node identifies a hotspot, it will recruit a worker node and migrate a group of geospatially proximate sub-blocks. This splitting continues recursively until the hotspot is alleviated. However, we have noticed that some of hotspots are caused by very few sub-blocks and the partitioning often results in just relocating hotspots. To address this highly skewed workload distribution, master nodes examine if the workload distribution over the sub-blocks are satisfying the 80-20 rule [46]. For instance, if less than 20% of sub-blocks are responsible for more than 80% of entire workloads in a given node, we consider that as highly skewed workload distribution. To alleviate resource hotspots with highly skewed workload distributions, the master node will recruit extra node(s) and create a replica of a highly popular sub-block. This is a typical strategy in cache based systems and is called selective replication. The routing information will be updated to include this replication information.

3.2 Autoscaling of a sub-cluster for the Area-of-Interests (RQ3)

Geospatially specific events or natural phenomena often create an area of interest, which causes intensive data accesses and processing loads. We have designed and implemented an autoscaling algorithm for sub-clusters to manage the area of interests effectively.

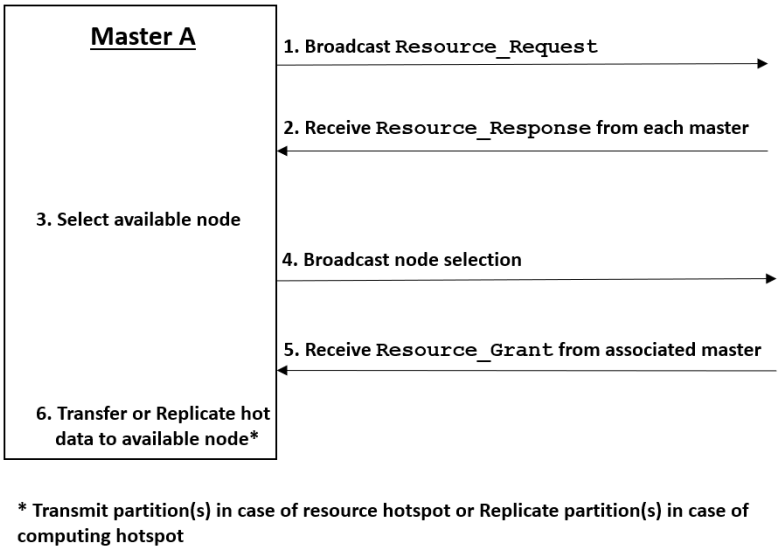


Figure 3.3: Master-to-Master Communication Protocol.

If a master node involves all of the worker nodes and still cannot alleviate resource hotspots, the master node will borrow nodes from other sub-clusters. In STRETCH, the master nodes monitor the resource utilization of their worker nodes. Therefore, the exploration of available nodes will be based on communications between master nodes. The protocol followed by masters for communication is shown in Figure 3.3. Similarly, Figure 3.2 depicts the recruiting of a foreign worker node for workload migration. The master node A with hotspot initiates communications with other master nodes to identify candidates to recruit. Among the candidates, the master node A will select a node (here, node k) and reports this selection to the original master node B. Once the original master node completes update of the routing table, the worker node k will be listed as a worker node of the A's sub-cluster. The master node A then start to migrate the workload to node k. In the meantime, the data stored in node k is still accessible by its primary master node B. The information about the original sub-cluster is also maintained in the routing table. The process of workload migration follows the same algorithm discussed in Chapter 3.1. This process continues until the hotspots are alleviated. Once the workload of the sub-cluster is reduced, the recruited nodes are returned to their respective sub-clusters. If there exist multiple nodes that are idle for a long time, they are shutdown by first saving their data in disk. When these nodes are needed in the

cluster, they are started by loading their previous data stored in disk. This is how node shrinking occurs in STRETCH. As an analogy, when a node is heated, hotspot alleviation procedure kicks in and when a node is idling, it is cooled down.

The delayed data partitioning scheme reduces the complexity and overhead of workload migrations. Incremental involvement of worker nodes prevents worker nodes from being occupied with workloads unless needed. This prevents fragmentation of resources at nodes and reduces the number of nodes to be contacted during migrations. In this way, STRETCH minimizes the overhead during negotiation and workload migration.

Chapter 4

System Architecture

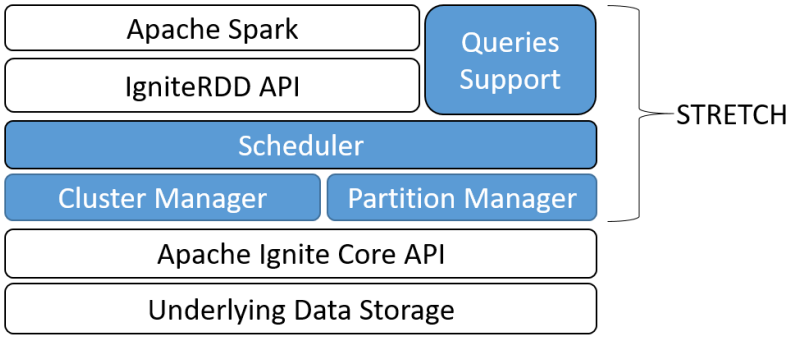


Figure 4.1: STRETCH Architecture.

The overview of STRETCH architecture can be seen in Figure 4.1 In STRETCH, geohash based multi-tiered indexing scheme is used to preserve the geospatial proximity of the data points and is dynamically adjusted at runtime. Data is retrieved from the underlying storage only once when accessed for the first time. STRETCH is implemented on top of Apache Ignite and thus allow users to use rich set of existing API, including integration with computing engine such as Apache Spark via IgniteRDD.

4.1 System Components

4.1.1 Master

There is a master per sub-cluster in STRETCH. Master monitors the system utilization among its worker nodes. Each master has a set of local routing tables that it maintains for data indexing: `KeyToPartitionMap` and `PartitionToNodeMap`. The functionality of these tables are explained in detail in Chapter 4.2. Similarly, master orchestrates the autoscaling process within and across sub-clusters. It is also responsible for initiating communication with other master nodes to

identify available nodes and negotiate recruiting foreign nodes. Each master plays role as Cluster Manager for its sub-cluster.

4.1.2 Worker

Each worker is associated with its master and reports its status periodically via heartbeat message. Worker loads the data in its memory and run the computation on the residing data.

4.1.3 Client

Client acts as a gateway between users' applications and the STRETCH cluster. Client as part of a user application load data, fetch data, run spatial queries: polygon query (in our case). It also has a global routing table for data indexing.

4.2 Dynamic Repartitioning for Autoscaling

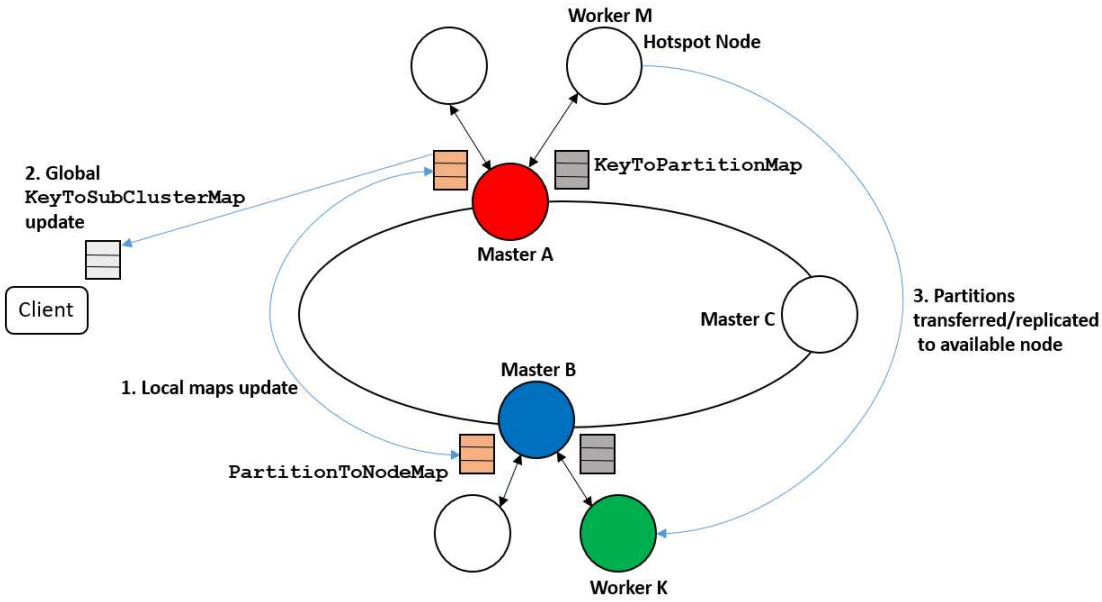


Figure 4.2: Dynamic Indexes Update at Runtime.

The STRETCH's three-tiered indexing scheme is the core concept that enable us to attain dynamic delayed repartitioning and autoscaling at runtime without causing any degradation in system

performance. At the top of the hierarchy is `KeyToSubClusterMap` that resides on heap memory of STRETCH client. This routing table keeps track of data point's primary sub-cluster and its replicated sub-cluster, if any. This is a global index as it gives a rough estimate of data's location in the STRETCH cluster. The other two routing tables are local to sub-clusters, meaning each master maintains its own pair of routing tables. `KeyToPartitionMap` designates key to a unique partition. No key has multiple partitions' residency, instead partitions are replicated to multiple nodes to maintain multiple copies of data point. `PartitionToNodeMap` determines the partition to actual worker node affinity.

4.2.1 Data Repartitioning

By implementing the multi-tiered indexing scheme, we decentralize index resizing and update. As mentioned earlier, to address resource hotspots, nodes split their coverage to 32 sub-blocks by adding another character to the geohash string in the key (key = geohash string + temporal information). This means a partition splits into 32 new partitions. Master updates its `KeyToPartitionMap` to trigger this change in its sub-cluster. A point to note here is that although the partition is split into finer partitions, these new partitions still reside in the same worker node. Neither has the key's affinity to sub-cluster changed; only the key's geohash characters are increased. Thus, we do not require to update `PartitionToNodeMap` and `KeyToSubClusterMap`, respectively. Only time the `PartitionToNodeMap` is updated is when the partition's node affinity changes or is increased to multiple nodes. Similarly, the global index `KeyToSubClusterMap` is updated only when there is an inter-sub-cluster partition transfer or replication.

In Figure 4.2, the steps 1-3 depicts the use-case where worker m becomes hotspot triggering partitions movement to available node k. There could be two possible scenarios for hotspot. Either, partition in node m is transferred to node k (due to resource hotspot) such that the two local `PartitionToNodeMaps` are updated followed by updating global index,

`KeyToSubClusterMap`. Or, partition in hotspot node m is replicated to node k (due to workload hotspot) such that only global index is updated.

The partition split is dynamic, and overhead is minimal as there is no network I/O. Existing systems like SP-Cache repartitions hot files every 12 hours. Our system, on the other hand, dynamically repartitions the popular partitions into 32 finer splits on the go with minimal overhead (less than 1.5 seconds delay on average). Even though the delay could be significant for real-time services, dynamic repartitioning in STRETCH is asynchronous operation that runs in the background without compromising the services provided by the cluster.

4.2.2 Off-heap Buffer

The incremental autoscaling feature involves partitioning hot partitions and updating hash indexes. As we are using minimum possible heap memory, the data stored in the segmented partition is temporarily flushed in off-heap memory which we call Off-heap Buffer. After the `KeyToPartitionMap` is updated, data in the buffer is flushed back to the storage. As the disk I/O is completely avoided, partitioning has minimal overhead. As an example, partitioning a hot partition of size 10GB takes less than 2.5 seconds.

4.3 Fault Tolerance

Each master is responsible for managing fault tolerance among its workers. Each worker periodically pings its master. If the worker doesn't send heartbeat message within a predefined interval, the worker is assumed dead. The master then uses the metadata information of the dead node and recover data partitions into its available worker. Primary partitions are loaded from the underlying data source while replicated partitions are copied from primary worker(s). During the cluster setup, each sub-cluster also starts a secondary master in a randomly selected worker node. A secondary master periodically checkpoints master so that in case of master failure, it could take over. As the resource utilization of a master is very low compared to that of a worker node, maintaining a secondary master does not inherit any serious overhead.

Chapter 5

Empirical Evaluation

5.1 Experimental Results

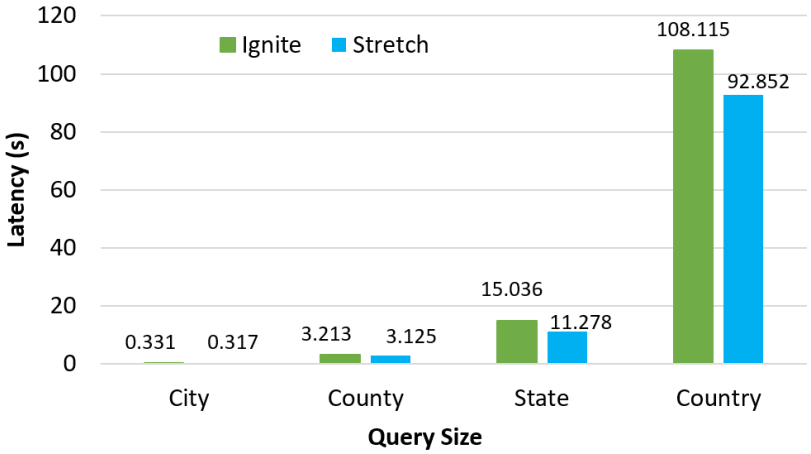
The experiments to compare STRETCH with Ignite were conducted on a cluster consisting of 110 nodes with a 8-core Intel(R) Xeon(R) CPU E5-2560v2 @ 2.10GHz processor, 16GB RAM and 1TB disk. Each machine in the cluster are connected to a Gigabit Ethernet switch and runs Fedora 28 (Server Edition) with Apache Ignite Core 2.7.0. Ignite cluster configuration: 100 Ignite servers with 8GB off-heap memory and 1GB heap memory.

STRETCH cluster configuration: 100 worker nodes with 8GB off-heap memory and 1GB heap memory; 10 master instances with 200MB heap memory and no off-heap memory. Remaining 10 nodes in both systems, were used to run clients to load data and submit polygon queries of different sizes and frequencies.

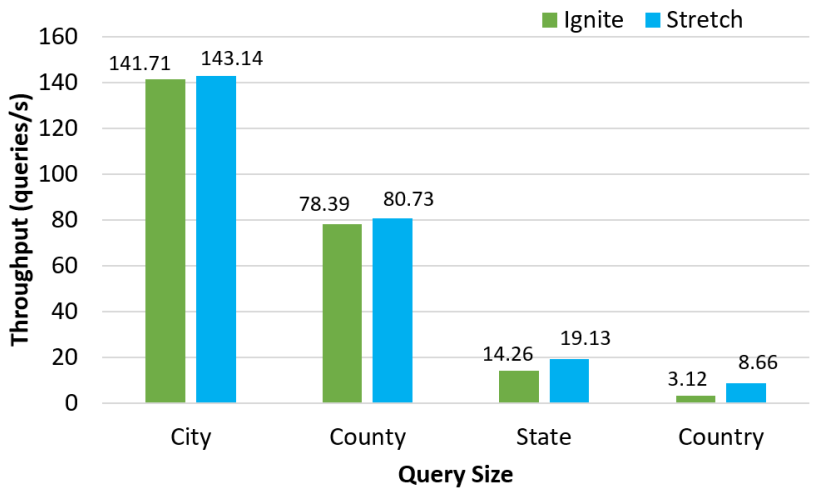
The dataset we have used the perform spatial queries against is The North American Mesoscale Forecast System (NAM) dataset for 2015, which contains atmospheric data collected several times per day globally including features like surface temperature, relative humidity, snow and precipitation. The total size of the dataset in-memory was 750GB.

Figure 5.1a and Figure 5.1b compares latency and throughput Ignite and STRETCH for polygon queries that differ in their latitudinal and longitudinal coverage (spatial extent). In case of the latency calculation, we have taken the average latency of 10 random polygon queries of same size issued over the cluster, for each spatial extent group. In throughput calculation, we have simultaneously issued 1000 same-sized random polygon queries, for each spatial extent group, into random nodes of the cluster. The performance of STRETCH is comparatively similar to that of Ignite when the query size is small. As the size of spatial extent group increases, performance difference becomes fairly significant. Since Ignite also uses a geospatial indexing scheme (R-tree) and it also preserves spatial proximity, evaluating queries for the smaller area does not show any

significant difference. However, for the greater area (e.g. country), STRETCH outperforms Ignite due to the effectiveness of hashing scheme.



(a) Latency



(b) Throughput

Figure 5.1: Performance measure w/o workload hotspot.

Figure 5.2a and Figure 5.2b compares execution time and throughput when there is workload hotspot in the cluster. We simulate workload hotspot by submitting same polygon query (in this case, query group: City) simultaneously using 10 client machines. Here, a node with more than 500 jobs waiting in its task queue is considered to be computing hotspot. The use of 500 jobs is determined by empirical process (execution time of a query and size of thread pool). As we

can see, STRETCH significantly outperforms Ignite as the number of queries increases. Although Ignite provides collocation of computations with data API which is essential for performance in a large-scale system, it does not have provision to handle workload hotspot thus causing a node to overload with jobs. Ignite puts the jobs into its waiting queue and processes only a small number of jobs concurrently which is defined by CPU cores of the node. In the experiment, when 100K queries are submitted, Ignite processes 16 queries at a given time. On the other hand, STRETCH has advanced adaptive scheduling mechanism that mitigates the hotspot at runtime. The master triggers replication of frequently accessed partitions when the number of jobs waiting in the queue is more than a threshold. It replicates the hot partitions to available nodes thus, making use of the idling resources.

Figure 5.3 shows that as the number of queries increases, the replication factor increases by two-folds. For 100K queries the hot partition is replicated to 95 nodes out of 100 in the cluster. In this experiment, STERTCH performed 3x better than Ignite.

5.2 Comparison against Ignite

Data Repartitioning

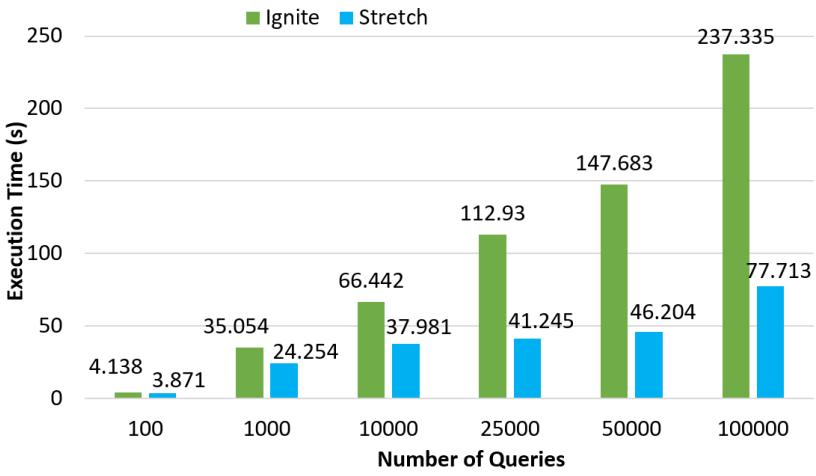
Data Repartitioning: Ignite supports partitioned cache where data is dispersed into partitions based on its default hash-function. It does not, however, support re-partitioning. STRETCH, on the other hand, supports dynamic re-partitioning.

Selective Replication

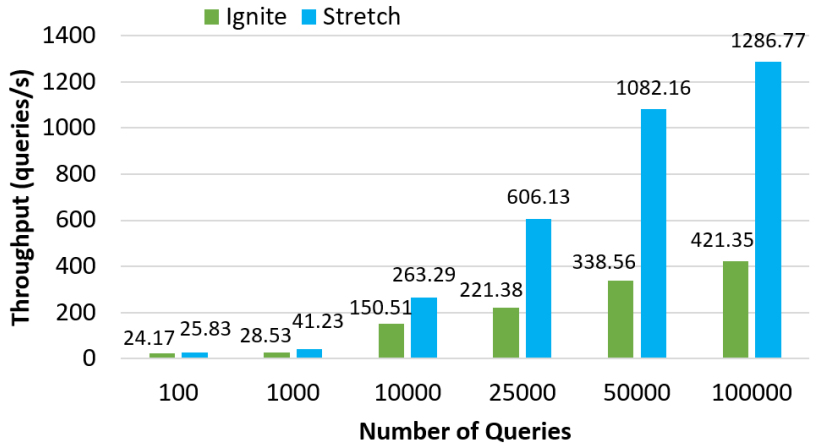
In Ignite, user can define the number of backup copies each partition will have statically during cluster setup. Whereas, STRETCH support selective replication of most popular data partitions dynamically at runtime saving resource by avoiding unnecessary backups of unused data.

Dynamic Indexing

Dynamic Indexing: Ignite has a static global routing table that indexes partitions to nodes in the cluster. A copy of this table is allocated to each node. The table is updated only when there



(a) Execution Time



(b) Throughput

Figure 5.2: Performance measure with workload hotspot.

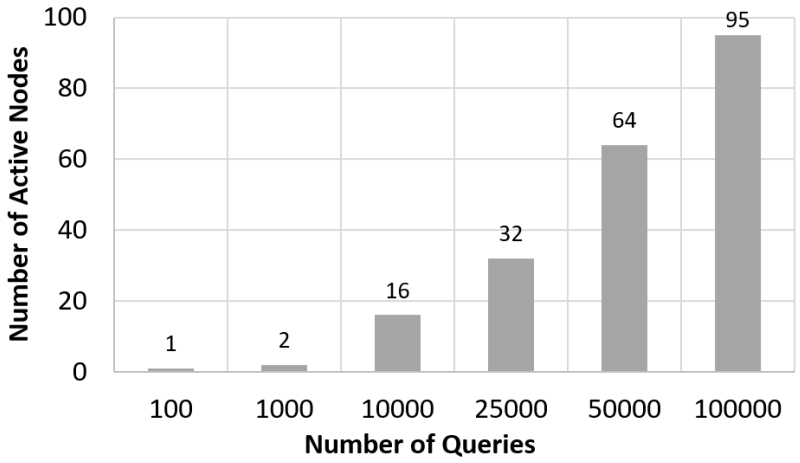


Figure 5.3: Active Nodes Handling Hotspot.

is topology change in the cluster triggered by a node addition, removal, failure, etc. STRETCH has dynamic index mapping feature implemented using three-tier indexing strategy that facilitate delayed partitioning and autoscaling capabilities.

Chapter 6

Conclusions and Future Work

This study describes STRETCH, an in-memory distributed storage system, which alleviates computation hotspots effectively for the large-scale geospatial analysis by: (1) using a delayed data dispersion with geospatial proximity, (2) monitoring data accesses in real-time, and (3) proactively repartitioning and migrating data. The geohash based data dispersion scheme preserves geospatial proximity of the data points and provides flexible resolutions [RQ1, RQ2]. STRETCH's autoscaling feature involves worker nodes within the sub-cluster and foreign nodes from other sub-clusters incrementally and this reduces the overhead for workload migrations [RQ1 and RQ3]. The workload migration scheme exploits distributional characteristics of the workload to cope with different types of hotspots effectively. We have evaluated STRETCH and compared its performance with Apache Ignite. The results showed that STRETCH performs similarly as Ignite when there are no hotspots, and outperforms Ignite by up to 3 times with hotspots.

6.1 Future Work

As part of future work, cloud security is an essential component that could be added so as to prevent unauthorized access to remote clients and other cluster members. Taking approach similar to that in enterprise-grade Ignite version, security mechanism could be implemented. Transmitting data between nodes over the network can be protected with SSL/TLS Encryption. If critical data is stored in distributed memory, securing access is crucial. Thus, by using JAAS-based authentication, any node trying to join the cluster could be authenticated which prevents unauthorized access to the cluster and cached data.

Similarly, the philosophy of autoscaling for hotspots alleviation while preserving geospatial proximity with which this research work was conducted can be extended to non-geographic high dimension data that has locality property. In such scenario, location-sensitive hashing [47] could

replace the geohash based indexing applied in this work. Clustering algorithm such as k-means could also be used to postulate locality among data and index based on its associated cluster group.

Bibliography

[1] Wikipedia Contributors. Location-based service. https://en.wikipedia.org/wiki/Location-based_service, March 2019.

[2] Statista Ltd. Statista. https://www.statista.com, March 2019.

[3] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[4] Alluxio Open Foundation. Alluxio. https://www.alluxio.org, March 2019.

[5] Haoyuan Li, Ali Ghodsi, Matei Zaharia, Scott Shenker, and Ion Stoica. Tachyon: Reliable, memory speed storage for cluster computing frameworks. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, pages 6:1–6:15, New York, NY, USA, 2014. ACM.

[6] The Apache Software Foundation. Apache ignite. https://ignite.apache.org, March 2019.

[7] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, and Minyi Guo. Simba: Efficient in-memory spatial analytics. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 1071–1085, New York, NY, USA, 2016. ACM.

[8] Ram Sriharsha. Magellan. https://magellan.ghost.io/magellan-geospatial-processing-made-easy, March 2019.

[9] Wikipedia Contributors. Distributed hash table. https://en.wikipedia.org/wiki/Distributed_hash_table, March 2019.

[10] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD '84, pages 47–57, New York, NY, USA, 1984. ACM.

[11] Apache Software Foundation. Apache hadoop. https://hadoop.apache.org, March 2019.

[12] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. Geospark: A cluster computing framework for processing large-scale spatial data. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '15, pages 70:1–70:4, New York, NY, USA, 2015. ACM.

[13] Christopher N. Eichelberger Anthony Fox Andrew Hulbert Michael Ronquest James N. Hughes, Andrew Annex. Geomesa: a distributed architecture for spatio-temporal fusion, 2015.

[14] Anand Padmanabha Iyer and Ion Stoica. A scalable distributed spatial index for the internet-of-things. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, pages 548–560, New York, NY, USA, 2017. ACM.

[15] Y. Fathy, P. Barnaghi, and R. Tafazolli. Distributed spatial indexing for the internet of things data management. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 1246–1251, May 2017.

[16] A. Fox, C. Eichelberger, J. Hughes, and S. Lyon. Spatio-temporal indexing in non-relational distributed databases. In *2013 IEEE International Conference on Big Data*, pages 291–299, Oct 2013.

[17] M. Malensek, S. L. Pallickara, and S. Pallickara. Galileo: A framework for distributed storage of high-throughput data streams. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, pages 17–24, Dec 2011.

[18] Sangmi Lee Pallickara, Shrideep Pallickara, and Marlon Pierce. *Scientific Data Management in the Cloud: A Survey of Technologies, Approaches and Challenges*, pages 517–533. Springer US, Boston, MA, 2010.

[19] Sangmi Lee Pallickara, Shrideep Pallickara, and Milija Zupanski. Towards efficient data search and subsetting of large-scale atmospheric datasets. *Future Generation Computer Systems*, 28(1):112 – 118, 2012.

[20] S. L. Pallickara, S. Pallickara, M. Zupanski, and S. Sullivan. Efficient metadata generation to enable interactive data discovery over large-scale scientific data collections. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 573–580, Nov 2010.

[21] C. Tolooee, S. L. Pallickara, and A. Ben-Hur. Mendel: A distributed storage framework for similarity searching over sequencing data. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 790–799, May 2016.

[22] Yogesh L. Simmhan, Sangmi Lee Pallickara, Nithya N. Vijayakumar, and Beth Plale. Data management in dynamic environment-driven computational science. In Patrick W. Gaffney and James C. T. Pool, editors, *Grid-Based Problem Solving Environments*, pages 317–333, Boston, MA, 2007. Springer US.

[23] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Polygon-based query evaluation over geospatial data using distributed hash tables. In *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, UCC '13, pages 219–226, Washington, DC, USA, 2013. IEEE Computer Society.

[24] M. Malensek, S. Pallickara, and S. Pallickara. Evaluating geospatial geometry and proximity queries using distributed hash tables. *Computing in Science Engineering*, 16(4):53–61, July 2014.

[25] M. Malensek, S. Pallickara, and S. Pallickara. Fast, ad hoc query evaluations over multidimensional geospatial datasets. *IEEE Transactions on Cloud Computing*, 5(1):28–42, Jan 2017.

[26] M. Malensek, S. L. Pallickara, and S. Pallickara. Expressive query support for multidimensional data in distributed hash tables. In *2012 IEEE Fifth International Conference on Utility and Cloud Computing*, pages 31–38, Nov 2012.

[27] The Apache Software Foundation. Apache mahout. https://mahout.apache.org, March 2019.

[28] Walid Budgaga, Matthew Malensek, Sangmi Pallickara, Neil Harvey, F. Jay Breidt, and Shrideep Pallickara. Predictive analytics using statistical, learning, and ensemble methods to support real-time exploration of discrete event simulations. *Future Gener. Comput. Syst.*, 56(C):360–374, March 2016.

[29] M. Malensek, W. Budgaga, R. Stern, S. Pallickara, and S. Pallickara. Trident: Distributed storage, analysis, and exploration of multidimensional phenomena. *IEEE Transactions on Big Data*, pages 1–1, 2018.

[30] M. Malensek, S. Pallickara, and S. Pallickara. Analytic queries over geospatial time-series data using distributed hash tables. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1408–1422, June 2016.

[31] Daniel Rammer, Walid Budgaga, Thilina Buddhika, Shrideep Pallickara, and Sangmi Lee Pallickara. Alleviating I/O inefficiencies to enable effective model training over voluminous, high-dimensional datasets. In *IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, USA, December 10-13, 2018*, pages 468–477, 2018.

[32] T. Buddhika, M. Malensek, S. L. Pallickara, and S. Pallickara. Synopsis: A distributed sketch over voluminous spatiotemporal observational streams. *IEEE Transactions on Knowledge and Data Engineering*, 29(11):2552–2566, Nov 2017.

[33] S. L. Pallickara and M. Pierce. Swarm: Scheduling large-scale jobs over the loosely-coupled hpc clusters. In *2008 IEEE Fourth International Conference on eScience*, pages 285–292, Dec 2008.

[34] M. Malensek, S. Pallickara, and S. Pallickara. Minerva: Proactive disk scheduling for qos in multitier, multitenant cloud environments. *IEEE Internet Computing*, 20(3):19–27, May 2016.

[35] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 205–220, New York, NY, USA, 2007. ACM.

[36] Memcached Contributors. Memcached. https://memcached.org, March 2019.

[37] Badrish Chandramouli, Guna Prasaad, Donald Kossmann, Justin Levandoski, James Hunter, and Mike Barnett. Faster: A concurrent key-value store with in-place updates. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 275–290, New York, NY, USA, 2018. ACM.

[38] Xingbo Wu, Li Zhang, Yandong Wang, Yufei Ren, Michel Hack, and Song Jiang. zexpander: A key-value cache with both high performance and fewer misses. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, pages 14:1–14:15, New York, NY, USA, 2016. ACM.

[39] Hyeontaek Lim, Dongsu Han, David G. Andersen, and Michael Kaminsky. Mica: A holistic approach to fast in-memory key-value storage. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 429–444, Berkeley, CA, USA, 2014. USENIX Association.

[40] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. Workload analysis of a large-scale key-value store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12, pages 53–64, New York, NY, USA, 2012. ACM.

[41] R. R. Noel and P. Lama. Taming performance hotspots in cloud storage with dynamic load redistribution. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 42–49, June 2017.

[42] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 7:1–7:13, New York, NY, USA, 2012. ACM.

[43] Ganesh Ananthanarayanan, Sameer Agarwal, Srikanth Kandula, Albert Greenberg, Ion Stoica, Duke Harlan, and Ed Harris. Scarlett: Coping with skewed content popularity in mapreduce clusters. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 287–300, New York, NY, USA, 2011. ACM.

[44] Yinghao Yu, Renfei Huang, Wei Wang, Jun Zhang, and Khaled Ben Letaief. Sp-cache: Load-balanced, redundancy-free cluster caching with selective partition. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '18, pages 1:1–1:13, Piscataway, NJ, USA, 2018. IEEE Press.

[45] P. Chockalingam, E. Liang, T. Das, and J.Y. Stephan. Introducing databricks optimized autoscaling on apache spark. https://databricks.com/blog/2018/05/02/introducing-databricks-optimized-auto-scaling.html, May 2018.

[46] Wikipedia Contributors. Power law. https://en.wikipedia.org/wiki/Power_law, March 2019.

[47] Wikipedia Contributors. Locality-sensitive hashing. https://en.wikipedia.org/wiki/Locality-sensitive_hashing, March 2019.