THESIS

A HIERARCHICAL FRAMEWORK FOR ENERGY-EFFICIENT RESOURCE MANAGEMENT IN GREEN DATA CENTERS

Submitted by

Eric Jonardi

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2015

Master's Committee:

Advisor: Sudeep Pasricha
Co-Advisor: H. J. Siegel

Adele Howe

ABSTRACT

A HIERARCHICAL FRAMEWORK FOR ENERGY-EFFICIENT RESOURCE MANAGEMENT IN GREEN DATA CENTERS

Data centers and high performance computing systems are increasing in both size and number. The massive electricity consumption of these systems results in huge electricity costs, a trend that will become commercially unsustainable as systems grow even larger. Optimizations to improve energy-efficiency and reduce electricity costs can be implemented at multiple system levels, and are explored in this thesis at the server node, data center, and geo-distributed data center levels. Frameworks are proposed for each level to improve energy-efficiency and reduce electricity costs. As the core count in processors continues to rise, applications are increasingly experiencing performance degradation due to co-location interference arising from contention for shared resources. The first part of this thesis proposes a methodology for modeling these co-location interference effects to enable accurate predictions of execution time for co-located applications, reducing or even eliminating the need to over-provision server resources to meet quality of service requirements, and improving overall system efficiency. In the second part of this thesis a thermal-, power-, and machine-heterogeneity-aware resource allocation framework is proposed for a single data center to reduce both total server power and the power required to cool the data center, while maximizing the reward of the executed workload in over-subscribed scenarios. The final part of this thesis explores the optimization of geo-distributed data centers, which are growing in number with the rise of cloud computing. A geographical load balancing framework with time-of-use pricing and integrated renewable power is designed, and it is demonstrated how increasing the detail of system knowledge and considering all system levels simultaneously can significantly improve electricity cost savings for geo-distributed systems.

ACKNOWLEDGEMENTS

This thesis was possible by the help, guidance, and support of many people. I would like to thank my advisors, Dr. Sudeep Pasricha, Dr. Howard J. Siegel and Dr. Anthony A. Maciejewski for the guidance and wisdom they provided throughout my time at Colorado State University. Thank you also to Dr. Adele Howe for agreeing to be a part of my thesis committee.

Of the students I have worked with at CSU, I would especially like to thank Mark Oxley for being a great research partner, and helping me through my masters in more ways than I can remember. I would also like to thank the remaining members of the CSU robust computing research group that I have worked with, Daniel Dauwe, Ryan Friese, Tim Hansen, Bhavesh Khemka, and Kyle Tarplee, your advice and suggestions have been extremely helpful. Lastly I would like thank my family, Gary, Chris, and Melissa, and my girlfriend Rebecca, for their unwavering love and support.

This research was supported by NSF grants CNS-0905399, CCF-1302693, and CCF-1252500. This research used the CSU ISTeC Cray System supported by NSF Grant CNS-0923386. This research used several servers generously donated by Hewlett Packard.

This thesis is typset in LaTeX using a document class designed by Leif Anderson.

DEDICATION

To my family whose love, caring, and wisdom are beyond compare

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION AND OVERVIEW

The ever increasing scale and computing capacity of data centers and high performance computing (HPC) systems have led to extremely large power consumption by these systems. This increased power consumption has required system operators to expend even more power to cool their systems, and has resulted in increasingly larger electricity costs. The trends will only continue as computing systems grow in size, particularly as HPC systems push towards exascale. To reduce the required cooling resources and resulting electricity costs, many new energy efficient systems, management systems and operating strategies are being developed and implemented.

Modern computer systems rely heavily on multicore processors, and the the number of cores within each processor continues to increase. The cores within each processor, however, share the memory resources within the processor. When a large number of tasks, especially tasks that require large amounts of memory, are run simultaneously, contention for shared memory resources can lead to the slowdown of the applications. Since this performance degradation is difficult to predict, many system administrators limit or disallow entirely the co-location of applications on multicore processors to ensure that Quality-of-Service (QoS) requirements are met.

Reducing or eliminating co-location on multicore processor requires that many more processor be used to accomplish the same workload. Being able to accurately predict the slowdown of co-located applications would enable system administrators to use less computer resources while remaining confident that QoS requirements will be met. Chapter 2 proposes a framework methodology for modeling and predicting performance degradation of applications running on a multicore processor. Using several different processor architectures, many trials were performed using benchmark applications in a variety of co-location scenarios. Detailed application performance

information was collected during each test using the performance counters in the processors. Machine learning techniques, specifically linear regression and neural networks, were trained on the collected data to generate models capable of accurately predicting performance degradation to to co-location. In addition to their usefulness to real systems, these models also enable simulations in other works to accurately represent co-location effects using real data.

Moving up to the data center level, there are many opportunities to improve efficiency and reduce electricity cost. Chapter 3 proposes several resource allocation heuristics that use holistic system knowledge (thermal, power, server heterogeneity, and task co-location effects) to maximize the completed workload for a given energy budget. A detailed thermal model predicts the temperatures within nodes, taking into account the cooling effects of the computer room air conditioning (CRAC) units and the power dissipated by all other nodes. Within each server, dynamic voltage and frequency scaling (DVFS) is used to trade computing speed for power consumption. The proposed resource allocation heuristics are run with and without the co-location effects described in Chapter 2, to demonstrate the difference in the resulting allocation decisions and the importance of including co-location effects in resource allocation techniques.

Chapter 4 extends the efforts of Chapter 3 by seeking to reduce electricity costs for a group of geo-distributed data centers. Many utility companies now use time-of-use (ToU) electricity prices, where the price of electricity is the highest during peak demand hours. This has led to operators of groups of geo-distributed data centers to use geographical load distribution (GLD) to distribute incoming workloads between data centers throughout the day, to take advantage of the cheapest electricity prices. In addition to this, many data centers are installing renewable power generation such as solar panels and wind turbines to reduce electricity bills. Chapter 4 proposes a hierarchical framework for GLD that includes not only ToU pricing and renewable power, but the holistic data

center models described in Chapter 3, providing an unprecedented level of optimization at the geo-distributed system level. Four heuristics are proposed, each with an increasing amount of system knowledge to demonstrate the electricity cost reductions that can be obtained from these methods.

CHAPTER 2

A METHODOLOGY FOR CO-LOCATION AWARE APPLICATION PERFORMANCE MODELING IN MULTICORE COMPUTING[1]

2.1. INTRODUCTION

There is an inherent trade-off in large scale computer systems between reducing the use of system resources by consolidating applications to as few server processor nodes as possible (to reduce system power) and the performance degradation that occurs in these applications as a result of sharing system resources with other applications. Memory interference caused by multiple applications co-located on a multicore processor has been shown to negatively impact application performance (e.g. [4], [5], [6], [7]). Specifically, sharing of system resources such as DRAM and the last-level cache by co-located applications creates contention and increases the memory intensity of all applications running on the multicore processor [5]. This increase in memory intensity results in a corresponding increase in average memory access time, which ultimately contributes to an increase in the application's overall execution time. This increase in execution time is significant, and in some cases can as much as double or triple the execution time of an application as compared to its baseline execution time [8].

The use of multicore processors that experience performance degradation due to co-location is pervasive throughout many kinds of computing systems, but its effects are most likely to be

[1] This work was done jointly with Ph.D. student Daniel Dauwe, and published at the Workshop on Advances in Parallel and Distributed Computational Models in 2015.

prevalent in large-scale server systems and high-performance computers. In these kinds of computing systems, highly parallel applications running on multicore processors result in a sharing of resources in a manner that creates memory interference and causes performance degradation. Having a methodology that is capable of predicting how well a system will run in a particular co-location scenario would be very useful for these systems. For example, the information gained from accurate co-location performance degradation could be integrated into intelligent application scheduling, and may lead to system performance improvement by more fully utilizing hardware and thereby increasing opportunities for server consolidation to save power while still maintaining quality of service constraints. **The work presented in this chapter provides a methodology that can be used to create co-location aware performance models.**

The methodology for analyzing system performance that is described in this work is general enough to be applicable to any set of applications running on any multicore processor. Once application performance information for a particular combination of multicore processor and target applications has been collected, our methodology uses machine learning techniques to construct performance models characterizing that performance information. Once trained, these models require only a single serial baseline measurement of parameters for each application running alone in the system to make predictions about the performance degradation from memory interference that will occur when the application is executing with different types of co-located applications. While it has been shown in [6] that an application's use of memory resources operates in varying phases across its execution, this work shows that going into such a level of detail is not necessary to make accurate predictions.

After describing how our methodology operates, our work validates the theory behind the proposed methodology using real world data collected on two Intel Xeon server-class machines running a collection of scientific application workloads, with some models performing with 98%

accuracy. In addition to creating and demonstrating a methodology that is capable of being ported across processor architectures, this work provides insight into what memory use information is most important to know about a set of applications running in a system.

This work makes the following key contributions: (a) we identify factors that can characterize slowdown during application co-location scenarios, and methods to capture those factors; (b) we propose a novel methodology to integrate these factors into multi-granularity and multi-fidelity performance models that can be used to predict application performance under co-location scenarios; (c) we show that a fine level of detail is not always necessary to achieve reasonable prediction accuracy; and (d) we validate the methodology with real-world data obtained from running co-located scientific workloads on contemporary Intel Xeon server-scale multicore processors with up to 12 cores per processor.

The rest of the paper is organized as follows. The next section discusses related work in this area. Our prediction modeling methodology is presented in Section 2.3. Section 2.4 describes details of the testing environment and data collection. Experimental results validating the models are shown in Section 2.5. The paper concludes with a discussion of practical applications that can be taken from the experimental results in Section 2.6.

2.2. Related Work

Several works have explored the impact of co-location on application performance in multicore environments. Here we briefly summarize the most relevant prior works in this area.

The authors in [7] give an early examination of how co-locating multiple applications on a single multicore processor affects performance. However, their work focuses on a general examination of the effects that co-location has on the system as a whole, and does not examine the effects on specific applications or create co-location performance models the way that our work does.

The study in [6] gives an excellent review of how the architecture that an application is run on can affect the cache use and memory intensity of that application. The paper however does not attempt to make predictions about performance degradation as we do, but it does show the importance of including memory intensity and cache usage information when characterizing performance degradation in the presence of application co-location.

Our work in [5] measures memory interference from application co-location, and its impact on system performance for a single Intel i7 machine. However that work does not create models that predict system performance, and the scope of the work is restricted to only a single consumer class machine.

The work in [9] describes the challenges faced by applications sharing resources, and the need for being able to perform precise predictions of performance degradation. The paper presents its "Bubble-Up" methodology for predicting performance degradation results. However, it does not consider the impact of dynamic voltage and frequency scaling on application performance, and also does not collect experimental data or characterize the memory interference effect of having more than two applications co-located.

The authors in [4] present an extension to the "energy roofline" model that explores the effect of memory intensity (from the perspective of arithmetic intensity) on execution time and power use. The study runs a series of constructed microbenchmarks on twelve machine architectures and provides an analysis of the performance of the systems. While this study collects data about performance degradation from memory interference on a set of real machines, it uses small "microbenchmark" tests on these machines, as opposed to the scientific workloads we use. Moreover their work does not create models to predict performance due to memory interference.

Similar to our work, the work in [10] also looks at creating a portable methodology using machine learning techniques for predicting application performance degradation from shared resources. The authors in this paper also incorporate shared resources beyond the last-level cache. However the addition of incorporating these resources forces the resulting model to be extremely complicated, and their model requires constantly monitoring a large number of processor performance counters, which can cause system-wide slowdown for all running applications. In contrast, our methodology needs to collect performance counter information about each application only a single time, and provides better prediction performance. Additionally, our methodology guarantees a uniform selection of training data over the possible co-location space (allowing for more portability) while the work from these authors selects the vast majority of its training data at random.

The authors acknowledge that work exploring the effect of hyperthreading (SMT) on application performance is an open and active area of research. Papers such as [11] and [12] examine scheduling and resource use of applications utilizing SMT. We chose to focus our study on the interference that applications experience at an inter-core granularity, and for this study we have turned off hyperthreading to remove the possibility of application interference in the L1 cache.

2.3. Modeling Methodology

2.3.1. Overview. Our work uses two types of machine learning techniques, *linear modeling* and *neural networks*, for constructing the predictive models. These techniques have been used in prior work [10], [9] but were limited in attribute selection and scope. For each machine learning technique, we design several models with varying levels of complexity and application features.

2.3.2. MODEL FEATURES. The performance prediction models that we design use up to eight separate features to predict how the target application performance is impacted by co-located applications. The eight features were chosen by performing a principal component analysis (PCA) on the data collected from multicore processors considered in this work. PCA allows all of the features that were gathered to be ranked according to variance of their output, giving an idea of which features were most important to include in the models.

These features are a general set that are present in most multicore processors. We have constructed models that use increasingly complicated combinations of the features. These range from combinations we felt would be most easily available to a resource manager making allocation decisions, to combinations that required more information about the application's baseline information to construct. The eight features are shown in Table 2.1. The table gives the name of the feature in the first column, and the aspect of the processor that it measures in the second column. The features

TABLE 2.1. Model Features

Feature name	aspect of execution measured
baseExTime	baseline execution time of target application at all P-states
numCoApp	number of co-located applications
coAppMem	sum of co-application memory intensities
targetMem	target application memory intensity
coAppCM/CA	sum of co-application last-level cache misses/cache accesses
coAppCA/INS	sum of co-application last-level cache accesses/instructions
targetCM/CA	target application last-level cache misses/cache accesses
targetCA/INS	target application last-level cache accesses/instructions

of Table 2.1 can be combined to create models of various complexities. The "target" application in the table is the one for which we are interested in determining slowdown due to co-location. The baseline execution time seen in Table 2.1 is the execution time of the target application without any co-location present. The model feature sets listed in Table 2.2 represent six possible scenarios, one baseline scenario (model "A") that uses only the *baseExTime* feature for predictions and five

other scenarios. For each of the five other scenarios, the resource management system has a certain amount of baseline information about the system, the target application, and the other applications co-located on the system. The progression from one model to the next simulates a realistic process where the resource management system progressively obtains more detailed information about the system and the executing applications.

TABLE 2.2. Sets of Model Feature Groups

Set name	feature groups within set
A	baseExTime
B	model A + NumCoApp
C	model B + coAppMem
D	model C + targetMem
E	model D + coAppTCM/TCA, coAppTCA/INS
F	model E + targetTCM/TCA, targetTCA/INS

2.3.3. LINEAR MODEL. To predict the impact of application performance due to co-location, six linear models were developed using the six feature sets listed in Table 2.2. Each linear model is the sum of the products of the utilized features and the model coefficients determined during training, plus a constant. A general model for predicting co-located execution time using N features would take the form of Equation 1. Linear regression is used to calculate the values for the coefficients using the linear least squares function in the Python package SciPy.

$$co\text{-}executiontime = \sum_{i=1}^{N} (coefficient_i * feature_i) + constant \tag{1}$$

2.3.4. NEURAL NETWORK MODEL. From observation of the raw data, there are obvious instances of nonlinearity in a few of the features of our data. This was the primary motivation for attempting to create a prediction model using a neural network that can capture non-linearity effects. Neural networks [13] are a machine learning technique that is commonly used when making predictive models. The approach is inspired by attempting to mimic how the human brain

is thought to work by defining a set of "neurons" that are used as nodes for the system. The inputs to these neurons are propagated through the network via a series of functions located at each node in the network. The final output value is determined by the input value's propagation through the network. For our model the input neurons are the features of the data available in each model, and the outputs determine the predicted execution time with performance degradation that the model will experience with co-location. The neural networks used in this work vary in the number of nodes used from ten to twenty depending on the model feature set that is used as inputs to the network. A *scaled conjugate gradient* numerical method was used to determine the co-efficient values at each network node.

2.3.5. MODEL ACCURACY. All of the models are evaluated using Mean Percentage Error and Normalized Root Mean Squared Error as two ways of comparing the predicted application execution time for each test ($predicted_j$) to the actual execution time for each test ($actual_j$).

2.3.5.1. *Mean Percent Error.* Mean Percent Error (MPE) is defined in Equation 2. The magnitudes of the actual values within the data vary greatly (e.g., when modeling execution time, actual values could range from as little as 150 seconds to over 1000 seconds based on the application that is being executed and the state of co-location of the applications in the system), and MPE allows the evaluation of prediction accuracy independent of these magnitudes for each of the M sample points of data. This error value is taken for just the target application's execution time.

$$MPE = 100 * \frac{1}{M} \sum_{j=1}^{M} |\frac{predicted_j - actual_j}{actual_j}| \tag{2}$$

2.3.5.2. *Normalized Root Mean Squared Error.* Normalized Root Mean Squared Error (NRMSE) gives an indication of the variance of our predicted values from the actual values. For M sample points, NRMSE provides a ratio of Root Mean Squared Error and the interval of values that the

actual data can take ($actual_{max} - actual_{min}$). Normalized root mean squared error is defined in Equation 3.

$$NRMSE = \frac{100}{M} * \frac{\sqrt{\sum_{j=1}^{M} (\frac{predicted_j - actual_j}{actual_j})^2}}{actual_{max} - actual_{min}} \tag{3}$$

2.4. Implementation Overview

2.4.1. Testing Environment.

2.4.1.1. *Operating System.* One of the objectives of this research was to design a methodology that could be applied to a wide variety of computing systems. Our testing environment was designed to be portable across many multicore processor architectures to allow for simplicity of gathering test data and ease of recreating the testing environment for future users of this work. To ensure accurate data is collected, the testing environment is run from a "lightweight" command line version of the Ubuntu 14.04 operating system [14] installed on a USB drive. This is done to minimize the effect that the operating system has on application execution. Unessential OS utilities and kernel daemons were removed so that the applications being monitored suffer as little interference from unpredicted events in the OS as possible. Such an environment mimics a large-scale computing platform meant to execute multiple parallel jobs.

2.4.1.2. *Processor Performance Counters.* Modern multicore processors provide the ability for developers to monitor hardware events that occur inside a multicore processor during the execution of an application. Through the use of specialized "performance counters" present in the processor it is possible to track the number of occurrences of certain events that take place, such as the number of instructions executed or last-level cache misses. These performance counters are

architecture dependent, and due to differences between microarchitectures the number and types of performance counters that are available to the system are not consistent (e.g. differences across [15], [16], and [17]). Given the design goal of having portability for our methodology, interfacing directly with these hardware performance counters is not a valid option. Therefore, the testing environment makes use of two tools to facilitate interactions with the hardware.

The first tool is the “Performance Application Programming Interface" (PAPI) [18], an API that was made specifically to provide portability when accessing performance counters across different architectures. PAPI has created a general list of more than 100 standard performance counter “presets” that are likely to be present in a modern processor. PAPI has made it more accessible to interface with these counters across architectures.

The second tool our testing environment utilizes is the HPC toolkit [19]. This suite of applications interface with PAPI and make it easier to monitor and collect information from multiple performance counters in the system. Specifically, HPC toolkit’s “hpcrun-flat” application profiler is used to collect performance counter information because it is able to run with very low overhead.

2.4.1.3. *Measuring Cache Use.* From [5], it is known that applications that need to access data from memory more often experience a larger amount of performance degradation due to co-location. We incorporate these performance degradations into our prediction models by collecting measurements of these effects. We have found that three hardware performance counter measurements can be used to collect the information necessary for deriving the metrics used in our methodology’s models:

- number of last-level cache misses an application experiences (LLC) representing the number of times an application must go to main memory
- number of instructions the application executes (NI)
- total number of last-level cache accesses the application attempts (TCA)

Measured features derived from these measurements were listed in Section 2.3. It should be noted that "last-level" cache misses and accesses are dependent on architecture, and can refer to either the L2 or L3 cache depending on the multicore processor that is being used. It is also important to note that when collecting test results for the execution of applications the values measured in these performance counters suffer a loss of temporal information, so they can only represent an average value across time.

One notable metric derived from this data is application memory intensity. Memory intensity is defined to be a ratio of an application's last-level cache misses to the number of instructions executed by that application. This metric gives an idea of the rate at which an application needs to go to main memory to fetch data. It is useful because it shows whether an application's execution will be more likely to be memory bound relative to another application, meaning that its performance depends more on its memory access speed rather than its computational speed. Memory intensity also gives some idea of how much an application tends to access memory. A highly memory intensive application will utilize the shared cache resource more, and therefore it will tend to affect, and be more affected, by the memory interference from other applications.

2.4.1.4. *Processor Performance States (P-states).* Processor performance states (P-states) are a set of discrete voltage and frequency values in which a multicore processor can operate. P-states utilize dynamic voltage and frequency scaling (DVFS), supported in all contemporary multicore processors. DVFS techniques can reduce the dynamic operating power of a multicore processor to consume less power or to temporarily reduce the operating temperature due to the multicore processor having exceeded a thermal cut-off. However, these benefits come at the cost of having to throttle the multicore processor speed by decreasing the clock frequency. This generally increases the execution time (decreasing system performance) of any application running on the multicore processor. The range and number of P-state frequencies that are available in a system are highly

dependent on the architecture of the multicore processor. Processor P-states are likely to change in high performance computing systems based on the system's need to reduce power or temperature. This work focuses only on how changing P-states affects application execution time. This effect is taken into account through knowledge of the baseline execution time of each application at a given P-state.

2.4.2. Data Collection and Experimental Setup.

2.4.2.1. *Benchmark Applications.* The applications run as testing workloads for our model validation were taken from two scientific benchmark suites. The set of eleven applications that we considered varied in the types of tasks that they perform and are characterized by a wide spread of memory intensity values. Table 2.3 shows the applications, those taken from the PARSEC benchmark suite [20] are denoted with *(P)*, and those from the NAS benchmark suite [21] are denoted with *(N)*. The table also shows the application's associated baseline memory intensity values, where baseline memory intensity values are measured when the applications are executed on a multicore processor by themselves without interference caused by co-location.

As shown in Table 2.3, these applications have been categorized into four memory intensity classes denoted as "Class I" through "Class IV." Class I applications are the most memory intensive applications (meaning that they have the highest number of last-level cache misses per number of instructions executed and are more memory bound), while class IV applications are the least memory intensive (meaning that they experience fewer last-level cache misses per number of instructions executed, and their execution is more CPU bound). Categorizing the applications into groups in this way allows applications from particular groups to be referred to more generally. These groupings allow for more broad use of this methodology for performance prediction. Should a system developer not have detailed memory intensity information about the applications running in the system, but still has a general idea of how memory intensive the applications might

be, then having application class values will allow the developer to still be able to use the model. The developer can still gain some insight as to the expected performance of the system by running the model with average values for that application's class.

It should be noted that the memory intensity values listed in Table 2.3 are from baseline measurements for one specific system. We found that the memory intensity values do not vary widely between the machines we tested, thus we used the memory intensity classes to accurately represent class categories for the Xeon family of multicore processors we consider. It is also important to note that the memory intensity values between application classes tend to differ by orders of magnitude. This allows for clearer distinctions to be drawn between application classes.

TABLE 2.3. Memory Intensity Classification

Applications	classification	baseline memory intensity (LLC / NI)
canneal (P)	Class I	1.84×10^{-2}
cg (N)	Class I	1.56×10^{-2}
ua (N)	Class II	1.63×10^{-3}
sp (N)	Class II	1.50×10^{-3}
lu (N)	Class II	1.11×10^{-3}
fluidanimate (P)	Class III	8.60×10^{-4}
freqmine (P)	Class III	3.47×10^{-5}
blackscholes (P)	Class III	1.88×10^{-5}
bodytrack (P)	Class IV	8.69×10^{-7}
ep (N)	Class IV	6.27×10^{-10}
swaptions (P)	Class IV	4.22×10^{-10}

2.4.2.2. *Multicore Processors Tested.* The specifications of the multicore processors tested during the validation of our methodology are shown in Table 2.4. All multicore processors used are from the Xeon family of multicore processors, with a varying number of available cores (ranging from six to twelve), L3 (last-level) cache size, and frequency ranges. More detailed information about these processors can be found in [16] and [17].

2.4.2.3. *Training Setup.* "Training data" was collected from each multicore processor to construct the models discussed in Section 2.3. The training data for all of the machines was collected

TABLE 2.4. Multicore Processors Used for Validation

Intel processor	num. cores	L3 cache	frequency range
Xeon E5649	6	12MB	1.60-2.53 GHz
Xeon E5-2697v2	12	30MB	1.20-2.70 GHz

in the form of execution time values of various co-locations using all eleven applications as target applications co-located with a subset of only four of the applications available in the testing environment. Specifically, *cg*, *sp*, *fluidanimate*, and *ep* were used as the applications that were co-located with each "target" application to provide the training data with a selection of applications that are representative of each of the four classes of memory intensity. We limited our use of co-location applications for training data to four applications to keep the number of tests that we could run for training tractable.

When measuring application performance, data is collected for only a single "target" application during any given co-location test. Initial baseline tests were run that measured each application's execution without co-location across six P-state frequencies to determine how each application performed without interference from other applications. This baseline test provides a basis of comparison for the effect of interference on application performance degradation. These four applications were then scheduled co-located with each other in a manner that allowed the machine being tested to have a sparsely distributed set of co-location tests that were evenly spread across the number of total possible co-location combinations available to the machine for use as training data.

In particular, training data was collected for each of the eleven target applications by running tests co-locating each application with multiple instances of each of the four co-location applications mentioned earlier. These four co-location applications represent applications from each of the four memory intensity classes. Multiple copies of each of these co-location applications

were run co-located together for each of the number of co-locations denoted in the "number of co-locations" column shown in Table 2.5. Each of these sets of tests where then run once for each of the six selected P-states for each multicore processor. The P-state frequencies are shown in Table 2.5. It should be clarified that each of the columns three to six in the table represent nested loops in the data collection code. Thus each attribute that is included as parameters to the data collection increases the size of the co-location space substantially. For example, the data collection was generally conducted as:

```
for each multicore processor:
    for each frequency:
        for each target application:
            for each co-located application:
                for each co-locations:
                    get_exec_time_of_target()
```

The "num. of co-locations" column in Table 2.5 shows the number of additional applications that were homogeneously co-located with the target application (i.e. all co-located applications are of the same type). The applications ranged on each multicore processor from only a single co-located application occupying one additional core, to co-located applications running on all of the multicore processor's available cores (i.e., one target application plus $n-1$ co-located applications for a multicore processor that has n cores).

Setting up the training data in this way is an attempt to sample the set of all possible co-locations for a given machine in a uniform way that minimizes the amount of training data that is needed to calculate co-efficients for our model. At the same time, the training data is carefully selected to provide a model that is flexible. The data that is collected for training the models is

designed to be able to both predict between the training data's gaps in the sample space, and extend beyond the set of four co-location applications available to the training data and be able to make predictions about applications that it has not seen previously.

TABLE 2.5. Training Schedule

Multicore processor	number CPUs (n)	target applications	co-location applications	frequencies (GHz)	number of co-locations
Intel Xeon E5649	6	all eleven applications	cg, sp, fluidanimate, ep	2.53, 2.40, 2.26, 2.00, 1.73, 1.60	1, 2, 3, 4, 5
Intel Xeon E5-2697v2	12	all eleven applications	cg, sp, fluidanimate, ep	2.70, 2.40, 2.10, 1.80, 1.50, 1.20	1, 3, 5, 7, 9, 11

TABLE 2.6. Effects of Memory Interference on *Canneal*

num. co-located applications	execution time (s)	normalized execution time	linear model F (MPE)	neural network model F (MPE)
0	220	1.00	-	-
1	229	1.04	11.0%	2.5%
3	239	1.09	9.4%	2.2%
5	253	1.15	7.3%	1.8%
7	266	1.21	4.5%	0.4%
9	277	1.26	4.1%	5.0%
11	292	1.33	1.7%	11.4%

2.4.2.4. *Model Testing.* Application testing was done by partitioning the training data described in Section 2.4.2.3 through repeated random sub-sampling validation based on the bootstrapping approach first described in [22]. Thirty percent of the data was randomly selected and withheld from the training process of each model. After training, the withheld data was run through each of the models and measured for accuracy. In this way, each model was tested using data that had not been seen previously during testing. The partitioning process was repeated one-hundred times, each time with a new random selection of points being withheld from training. The error values from each of these one-hundred training and testing partitioning groups was then averaged to determine the overall accuracy of each model.

2.5. EXPERIMENTAL RESULTS

2.5.1. OVERVIEW. This section details the performance results of each of the 12 models we propose (two classes of modeling techniques - linear, and neural network - with six variants each, based on the six model feature groups in Table 2.2). Each of the feature groups offers a trade-off between prediction accuracy and model complexity. Figures 2.1- 2.4 show the prediction accuracy for the training data set and testing data set on the 6-core Intel Xeon E5649 and the 12-core Intel Xeon E5-2697v2 multicore processors, respectively. Model prediction accuracy represented in both MPE and NRMSE is included for each of the machine learning techniques.

Each data point in the figures represents the average training error and average testing error from one-hundred partitions of the data for a particular model. In the case of each model, the error for each partition that was tested did not vary much (at most a quarter of a percent), meaning that each separate model shown has a tight confidence interval for its prediction accuracy.

2.5.2. THE EFFECTS OF MEMORY INTERFERENCE. Table 2.6 provides an example of the performance degradation the *canneal* application experiences from memory interference due to increasing numbers of the *cg* application co-located on cores of a 12-core Intel Xeon E5-2697v2 multicore processor. It can clearly be seen in the table how the application's performance (the execution time of *canneal*) is negatively impacted over *canneal's* baseline execution time of 220 seconds as the number of co-located applications increases. The increase in execution time is shown in the "normalized execution time" column of the table as the co-located execution time normalized by the baseline execution time of the *canneal* application. In this case, increasing application execution time by as much as 33%. The table also shows the accuracy of the execution time predictions for linear and neural network models (using the MPE metric) making execution time predictions using feature set "F."

2.5.3. RESULTS OF LINEAR MODELING. For both multicore processors tested, the more advanced linear models provide only a modest improvement over the baseline linear model (model A) when feature information is added to the models. For the 6-core processor (Figure 2.1) the linear model has an MPE training error and testing error of 8% for its baseline models. With the addition of new features the 6-core processor is able to reduce its error down to an MPE of 6.5% for the linear model. The 12-core processor does not show any improvement from the addition of more model features (Figure 2.2). The linear NRMSE variance results for the 6-core and 12-core processors (Figure 2.3 and Figure 2.4) follow very similar trends to the MPE model except that the variance in predictions reduces from 4.25% to 3.75%.

The complexity of the sample space makes it challenging for the linear models to perform well beyond the baseline model, and in each case the only cache use information that proves to be helpful is information about the co-located applications. Given the fact that the model features do not help to train these models very well, it is not surprising that performance of the testing data very closely matches that of the training data. As mentioned earlier, non-linearity in the data make predictions with these linear models less accurate. The 12-core processor results in particular suffer from having to predict performance degradation for a much greater number of possible co-locations.

2.5.4. RESULTS OF NEURAL NETWORK MODELING. The neural network models provide a clear improvement and very accurate predictions over the linear models (Figures 1-4). It can clearly be seen in the neural network models how the addition of application cache use helps to improve the predictions of each model. Predictably, the most complex neural network models, which utilize the most information, perform the best (operating with only a 2% MPE error on the testing data for both multicore processors), however it appears as though the most important features are the

features measuring the cache use information of the applications that are co-located with the target (coAppMem, coAppCM/CA, and coAppCA/INS from Table 2.1).

As with the linear models, the NRMSE results show that the variance between the predictions and actual values decrease with generally the same trends as the MPE graphs except that for both machines the inclusion of the co-located application's cache access related metrics in model E reduced the NRMSE greater than it reduced the MPE.

2.5.5. MODEL ACCURACY. Figure 2.5(a) shows a detailed view of each application's execution time distribution on a 6-core Intel Xeon E5649 machine. The points inside each application's distribution mark the specific execution time values measured for each test run for each application. Space limitations prevent showing a detailed view of the performance of all models, however Figure 2.5(b) shows a detailed view of the performance of the neural network model using feature set F (i.e. the most accurate model) on the execution time data shown in Figure 2.5(a). Distributions of the percent error between the model's execution time predictions and the application's actual execution time are shown for each application. The lines across each distribution represents the distribution's median (dashed line) and upper and lower quartiles (dotted lines). The figure shows that the model's predictions are generally accurate (there error is close to zero), that the majority of the model's predictions are $\pm 2\%$ from the actual execution time values, and that nearly all of the predictions are within 5% of the actual execution time values.

2.6. CONCLUSIONS

In this chapter, we propose a modeling methodology that allows for predictions to be made about the performance degradation that occurs from multiple applications running co-located on a multicore processor. The methodology is general enough to be applied to any multicore processor or set of applications. To validate the methodology we demonstrate its effectiveness by applying it

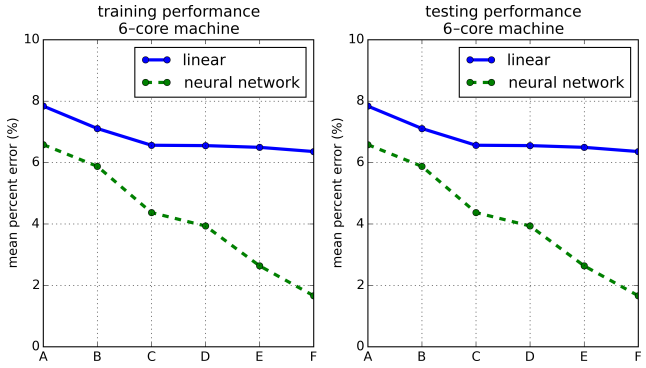

FIGURE 2.1. Model performance per feature set for the 6 core Intel Xeon E5649. (a) MPE for the training data set. (b) MPE for the testing data set. The figure shows results for each of the machine learning techniques: *linear* (solid blue) and *neural networks* (dashed green).

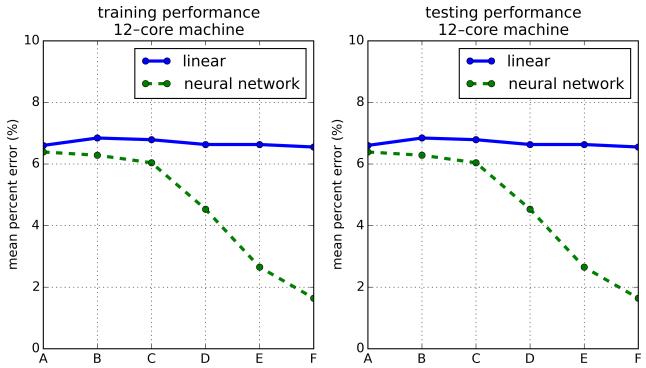

FIGURE 2.2. Model performance per feature set for the 12 core Intel Xeon E5-2697v2. (a) MPE for the training data set. (b) MPE for the testing data set. The figure shows results for each of the machine learning techniques: *linear* (solid blue) and *neural networks* (dashed green).

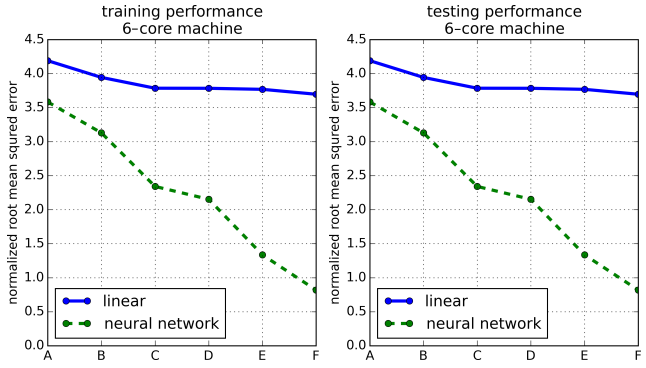

FIGURE 2.3. Model performance per feature set for the 6 core Intel Xeon E5649. (a) NRMSE for the training data set. (b) NRMSE for the testing data set. The figure shows results for each of the machine learning techniques: *linear* (solid blue) and *neural networks* (dashed green).

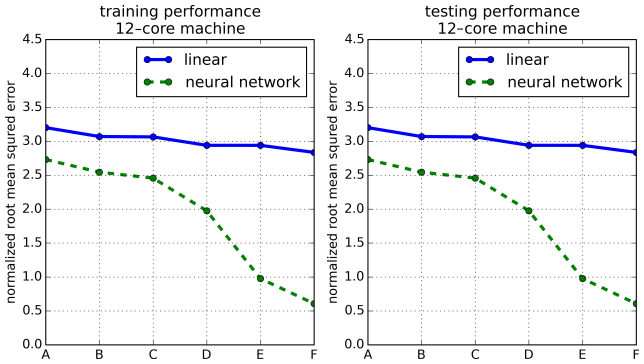

FIGURE 2.4. Model performance per feature set for the 12 core Intel Xeon E5-2697v2. (a) NRMSE for the training data set. (b) NRMSE for the testing data set. The figure shows results for each of the machine learning techniques: *linear* (blue) and *neural networks* (green).

to two server class Intel Xeon multicore processors with up to 12 cores, running real data workloads from two scientific benchmark suites.

Specifically, this work looked at how machine learning techniques can be used to make predictions about application performance degradation due to contention in shared cache and main memory resources when multiple applications were run co-located on the same multicore processor. While simpler linear models do not seem to provide much benefit from the addition of application cache use information, the results from Figure 2.1 and Figure 2.2 show that neural networks can provide quite accurate predictions of application performance. For the neural network even the addition of only some of the application features is capable of producing fairly accurate performance predictions. Using all the features the neural network has only a very small MPE of 2% and an NRMSE of around 1%. Considering that performance degradation due to co-location can extend an application's execution time quite significantly, even a model limited to only knowledge of application memory intensity may be able to provide good enough predictions for performance.

Applying this methodology to create models for larger scale systems where the much greater number of cores could cause significantly larger performance degradation due to co-location would enable the design of a new class of interference-aware intelligent scheduling mechanisms that

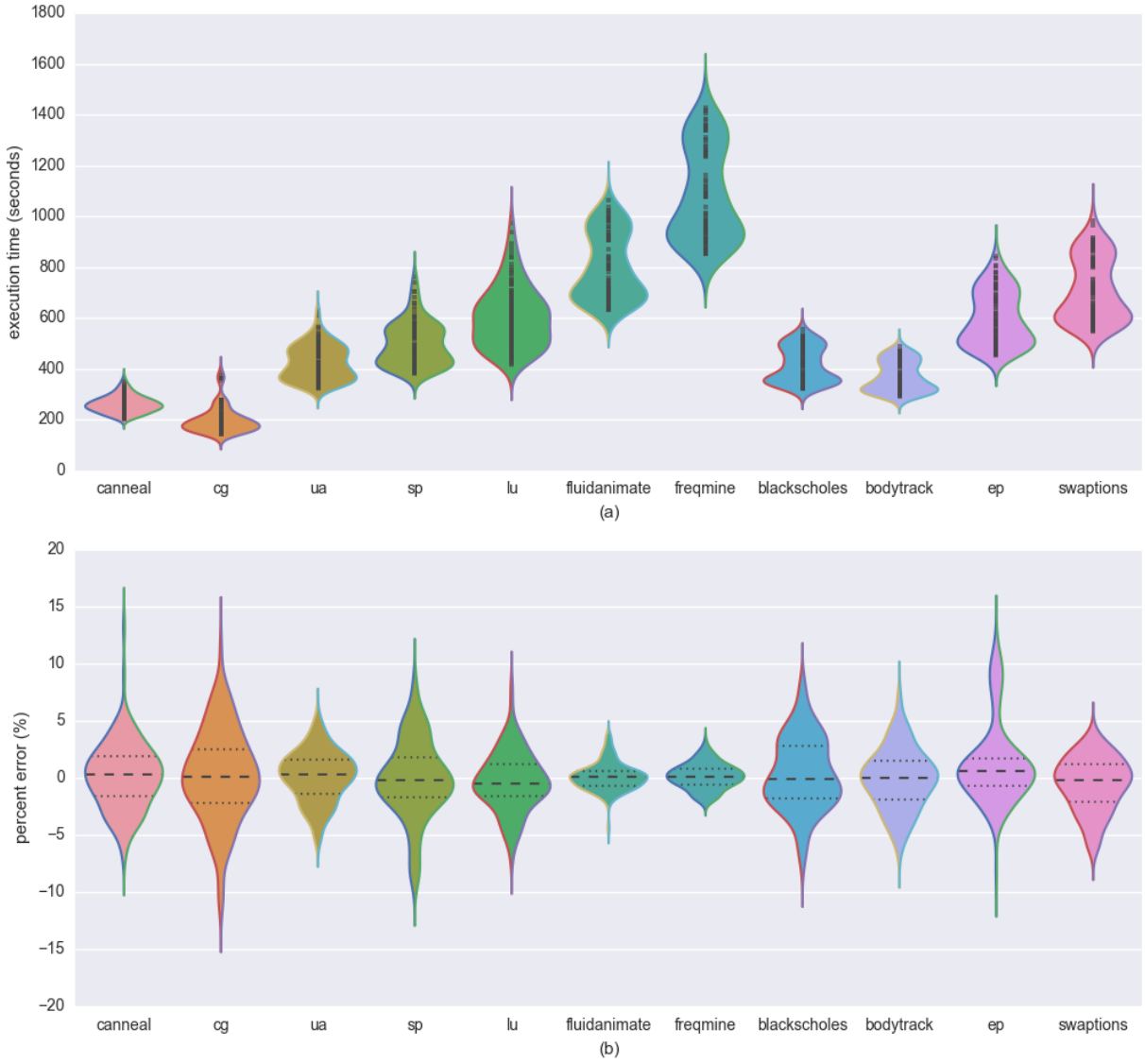

FIGURE 2.5. Distributions of each application's execution time (a), and the accuracy of the neural network model using feature set F on each application (b), for the 6 core Intel Xeon E5649 machine.

could provide substantial performance improvement in those systems. In addition to possibilities for optimizing application scheduling, a system for predicting application performance, such as the method shown here, offers the possibility for use in other areas such as energy modeling. Our next steps are to include monitoring of application power use into the testing environment. The energy use of a system is heavily dependent on the time that the system spends executing applications. Having this methodology that is capable of predicting an application's execution

time when presented with the uncertainty of memory interference from co-location allows this work to lend itself very well to being able to also include the ability to estimate the energy used by the system during execution of a particular application, as well as the increase in energy use that is caused by memory interference. Finally, it would also be interesting to extend this work by examining application performance on families of machines outside of Intel's Xeon architecture.

CHAPTER 3

THERMAL, POWER, AND CO-LOCATION AWARE RESOURCE ALLOCATION IN HETEROGENEOUS HIGH PERFORMANCE COMPUTING SYSTEMS[1]

The power consumption of high-performance computing (HPC) systems and data centers is increasing rapidly, leading to an increase in the amount of cooling resources required to operate these HPC systems at a safe temperature threshold. The increase of power consumption to operate today's facilities has led to the design of power-efficient systems such as the TSUBAME-KFC at the Tokyo Institute of Technology that tops the Green500 list with an efficiency of 4.5 GFLOPS/Watt [23]. Extrapolating the power consumption of the TSUBAME-KFC system to exascale, it would consume 222 MW, which equates to approximately $145 million per year in electricity costs in the United States. The Defense Advanced Research Projects Agency (DARPA) has set the target for an exaflop system to consume no more than 20 MW [24], an order of magnitude of power consumption lower than what can be achieved using today's computing infrastructure.

One technique that can be used to reduce the power consumption and also manage thermal issues in large-scale computing systems, such as data centers, is power-aware and thermal-aware resource allocation. This approach involves exploiting system characteristics such as the heterogeneity among different servers (offering different levels of performance and power consumption),

[1] This work was done jointly with Ph.D. student Mark Oxley, and published at the International Green Computing Conference in 2014.

dynamic voltage and frequency scaling (DVFS) in cores, and the interactions between temperatures of the compute nodes and computer room air conditioning (CRAC) units. By distributing workloads, configuring DVFS, and setting CRAC thermostats in an intelligent manner, it becomes possible to efficiently balance the compute performance with the power consumption and temperature profiles of the HPC facility.

By the law of conservation of energy, the power consumed by servers is dissipated as heat. This heat must be removed by the cooling infrastructure so that compute nodes can safely operate beneath their specified redline temperatures (the maximum safe operating temperatures). The more power that is consumed by compute nodes, the more power CRAC units must consume to remove additional heat and maintain the redline temperatures. Cores within compute nodes are DVFS-enabled, with performance states (P-states) that provide a trade-off between compute performance and power consumed by each core. By intelligently configuring P-states of cores, power consumption can be reduced at the compute nodes, which in turn results in less power that is needed by the CRAC units to cool the facility.

As the number of cores increase in emerging multicore processors, access contention in shared memory (e.g., last-level caches, DRAM) can have a pronounced impact on the execution time of workloads [25]. We model this interference as a performance degradation in the execution rate of tasks when different cores in the same multicore processor are executing tasks at the same time. The performance degradation can be severe when memory-intensive tasks running on separate cores of the same processor access the shared memory simultaneously. Thus, it is intuitive that to minimize interference, one should co-locate memory-intensive tasks with compute-intensive tasks (that have relatively fewer memory accesses), if possible.

We consider a steady-state model of a data center, where task flow rates, temperatures at compute nodes and CRAC units, and the power consumption of the computing system and CRAC

units remain invariant. The performance of the data center is measured by the reward collected from completing tasks by their individual deadlines, where reward represents the worth of completing that task to the system. In the steady-state this is equivalent to the *reward rate* collected. Our resource management techniques mitigate the impact of co-location interference by maximizing *co-located reward rate*, a measure for estimating reward rate when considering co-location interference. By taking a holistic approach to the control of such a facility, we maximize the co-located reward rate earned while ensuring that the compute nodes do not exceed their redline temperatures and the total power consumed by the compute nodes and CRAC units do not exceed a given constraint. To solve this optimization problem, we design a new greedy heuristic, a novel genetic algorithm (GA) with local search, and improve a non-linear programming (NLP) approach from our previous work [26] to consider the effect of co-locating tasks on multicore processors.

In summary, we make the following novel contributions:

- Derivation of a new detailed model of a heterogeneous HPC system that considers the power consumption of both the compute servers and cooling system, thermal constraints, DVFS P-states, memory-intensity of tasks, and co-location interference.
- Design of a greedy heuristic, a genetic algorithm with local search, and an adaptation of a previously proposed non-linear programming approach that is the first work to our knowledge to consider co-location while obeying power consumption and thermal constraints.
- Analysis and comparison of our resource allocation techniques with prior work that does not consider co-location when optimizing for a power and thermal constrained data center.
- Analysis of the performance of our proposed techniques on three different platform sizes.

The rest of the chapter is organized as follows. We discuss related work in Section 3.1. In Section 3.2, we explain our models for compute nodes, CRAC units, and workload as well as how we consider co-location interference. Section 3.3 describes our proposed resource allocation

techniques. Our evaluation setup and results are in Sections 3.4 and 3.5. In Section 3.6, we conclude and discuss ideas for future work.

3.1. Related Work

The power consumption of a data center's cooling system can consume approximately 50% of the total power consumed by the facility [27]. Proper thermal management of a data center can therefore result in large savings in dollar expenditures due to reductions in energy consumption of the compute and cooling infrastructure. Hot spots in the data center can be mitigated in several ways, such as intelligent workload distribution, throttling of compute servers (typically using DVFS), and better management of CRAC units. The techniques proposed in [28] distribute the workload across a data center to minimize inefficiencies in heat removal and hot spot formation, so as to minimize cooling costs. Throttling the performance of servers (or shutting them down) is another technique for controlling the temperatures in an HPC facility. The research in [29] analyzes whether it is more energy efficient to distribute the workload evenly to avoid hot spots, or to concentrate the workload on a small number of servers to be able to deactivate many servers. Methods for controlling the on/off state of CRAC units to minimize hot spots in a data center and save cooling energy by reducing air flow in overprovisioned areas are proposed in [30]. These works perform thermal-aware resource allocation by examining only one of either workload distribution, throttling servers, or managing CRAC units, whereas our work simultaneously considers workload distribution, throttling through DVFS, and CRAC thermostat control.

The thermal-aware approach designed in [31] distributes the workload to servers in a spatial manner such that hot spots are minimized, allowing the CRAC thermostats to be set to higher temperatures so that cooling power is reduced. That work does not consider throttling of servers to reduce the power consumption (and heat generation) of compute nodes. Our previous work

considers managing the workload distribution, throttling of servers, and managing CRAC units simultaneously. The goal of the study in [26] is to maximize the reward rate earned in a data center under power consumption and thermal constraints. A novel non-linear programming technique is designed to solve the resource allocation problem. We build upon this work by enhancing the resource allocation technique to consider co-location interference, and by designing new resource allocation techniques that scale better for larger system sizes.

In situations where tasks are co-located on the various cores of an individual processor, the primary sources of performance degradation are the contention for shared caches and memory bandwidth. Depending on the workload characteristics of the co-located task, the performance degradation due to co-location can range from negligible to severe [25]. Prediction of the impact of co-location requires an understanding of the memory system requirements of each task and how each task would affect and be affected by the other tasks with which it is co-located. This understanding can be obtained by either characterizing individual tasks using synthetic testing structures [32], or experimentally by pairing tasks together [25]. Co-location predictions could be used to reserve memory bandwidth, such as the MemGuard technique described in [33] that predicts the memory interference of an application and uses that knowledge in a scheduler to avoid co-location effects. Our research uses the knowledge and experimental data presented in those works to consider the co-location effects at a larger scale than the single-processor analyses in those works.

3.2. System Model

3.2.1. Overview. Our model of a data center and workload builds on the model proposed in [26]. We assume the data center is configured in a hot aisle/cold aisle fashion (Fig. 3.1). In such a configuration, cold air is supplied from the CRAC units to a cold aisle through perforated floor tiles that face the inlets of the compute nodes. The compute nodes consume power and expel hot

air through the opposite end to a hot aisle. The CRAC units draw the hot air from the hot aisles to cool. Our model can support isolated and non-isolated aisle configurations.

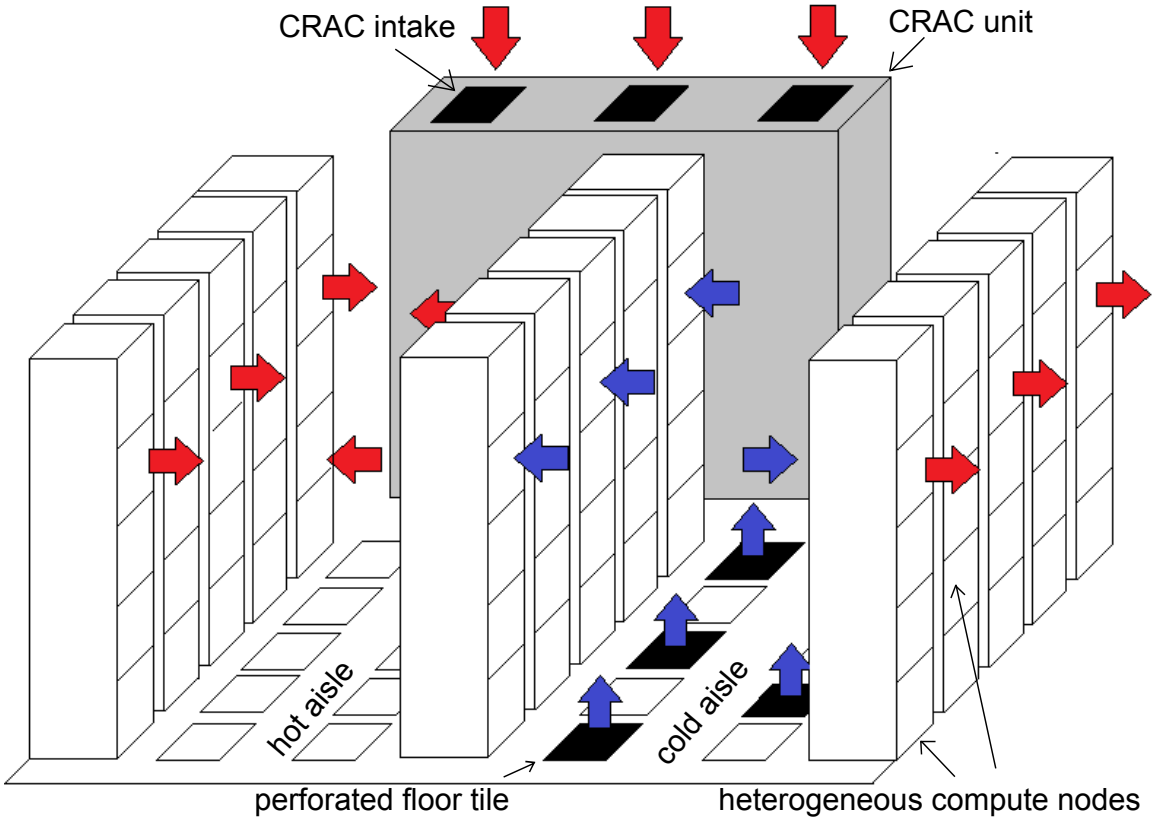

FIGURE 3.1. Data center in hot aisle/cold aisle configuration.

3.2.2. COMPUTE NODES. The number of compute nodes in the data center is NN, and each compute node j belongs to a compute node type $NT(j)$. We assume a heterogeneous data center with the number of compute node types equal to NNT, and compute nodes that belong to a specific compute node type $NT(j)$ are identical and contain the same number of cores, power characteristics, and performance characteristics. Cores within a compute node are homogeneous, and we assume the cores can run in individual P-states that provide a tradeoff between power consumption and performance [34]. The total number of cores in the data center is NC, and $CT(k)$ is the type of the compute node to which core k belongs.

3.2.3. WORKLOAD. We assume that we have a set of T known task types. The arrival rate of tasks of type i is given by λ_i. A reward r_i is obtained for completing a task of type i by its individual deadline d_i, relative to its arrival time.

As our system is heterogeneous, the power and performance characteristics of the compute node types are different. Therefore, tasks of different types can have different execution rates on different node types and P-states. We assume that we know the estimated computational speed (ECS) of any task of type i on a core of type j in P-state k, ECS(i,j,k) (i.e., number of tasks per second). In an actual system, these can be approximated using historical, experimental, or analytical techniques [35–37]; in Section 3.4 we discuss the values used for our evaluation.

Our goal is to assign a *desired fraction* of time each core k will spend executing tasks of type i, denoted $DF(i,k)$, and the P-state each core k is configured to when executing tasks of type i, denoted $PS(i,k)$, to maximize the reward we can obtain and meet the power and thermal constraints of the system. Given $DF(i,k)$ and $PS(i,k)$, we calculate the *execution rate* of tasks of type i on core k, $ER(i,k)$, as

$$ER(i,k) = DF(i,k) \cdot ECS(i, CT(k), PS(i,k)). \tag{4}$$

The total *reward rate* earned, denoted RR, without considering co-location interference is

$$RR = \sum_{i=1}^{T} \left(r_i \cdot min(\sum_{k=1}^{NC} ER(i,k), \lambda_i) \right). \tag{5}$$

Equation (5) states that the reward rate for a given task type i is the product of the reward earned and its execution rate. The min function in Equation (5) enforces the intuitive constraint that if a task is assigned for execution at a faster rate than its arrival rate λ_i, additional reward is not earned.

Reward is earned only when tasks are completed by their deadline. We have no way of guaranteeing deadlines are met in our steady-state model, but we can help minimize the number of deadlines that would be missed by eliminating any task type/core type/P-state combinations from consideration that would result in a missed deadline even if a task of type i starts immediately after its arrival, i.e., we eliminate combinations from consideration that violate

$$\frac{1}{ECS(i, CT(k), PS(i,k) \geq d_i}. \tag{6}$$

Another constraint that must be enforced is to ensure cores do not spend greater than 100% of the time executing tasks. If a core k is spending greater than 100% of the time executing tasks, we normalize the $DF(i,k)$ values on that core to guarantee this constraint. Normalization is performed by summing the desired fractions of time all task types spend executing on a given core k, denoted $SumDF(k)$, and dividing the desired fraction of time values for all T task types by $SumDF(k)$, forcing their sum to equal 100%. This normalization procedure is performed before calculating execution rates (Equations 1, 9, and 10) and the power consumption of a node (Equation 4, Section 3.2.4). Normalization is not performed on a core if the value of $SumDF(k)$ is less than 100%.

3.2.4. POWER AND THERMAL MODEL. We consider the overhead power consumption of a compute node (e.g., from main memory, disks, fans, NICs) in addition to the power consumption of the CPU cores. We assume that CPU cores are able to change P-states over time depending on what task type is currently being executed, and that the time associated with switching P-states is negligible in comparison to the execution time of tasks. The power consumed by cores is a function of the task type being executed in addition to the P-state in which the core is executing a task. Let $O(j)$ be the overhead power consumption of compute node j, let APC(i,$NT(j)$,$PS(i,k)$) be the

average power consumed by cores in a node of type $NT(j)$ executing tasks of type i in P-state $PS(i,k)$, and $NCN(j)$ be the set of cores in node j. In the steady-state, we calculate the power consumption of node j, $PN(j)$, as

$$PN(j) = O(j) + \sum_{k \in NCN(j)} \sum_{i=1}^{T} APC(i, NT(j), PS(i,k)) \cdot DF(i,k). \tag{7}$$

The power consumed at a CRAC unit is a function of the heat removed at that CRAC unit in addition to the Coefficient of Performance (CoP) [38] of the CRAC unit. Let NCR be the total number of CRAC units in the data center, $TC^{in}(i)$ be the inlet temperature of CRAC unit i, $TC^{out}(i)$ be the outlet temperature of CRAC unit i, ρ be the density of air, C be the specific heat capacity of air, and $AFC(i)$ be the air flow rate of CRAC unit i. The power consumed by CRAC unit i, $PC(i)$, is calculated as [38]

$$PC(i) = \frac{\rho \cdot C \cdot AFC(i) \cdot (TC^{in}(i) - TC^{out}(i))}{CoP(TC^{out}(i))}. \tag{8}$$

To calculate the steady-state temperatures at compute nodes and CRAC units, we use the Abstract Heat Flow Model from [39]. Let $TN^{in}(i)$ be the inlet temperature at compute node i and $TN^{out}(i)$ be the outlet temperature at compute node i. The outlet temperature at compute node i is a function of the inlet temperature, the power consumed, and the air flow rate of the node $AFN(i)$, calculated as

$$TN^{out}(i) = TN^{in}(i) + PN(j) / (\rho \cdot C \cdot AFN(i)). \tag{9}$$

Let $\mathbf{T^{out}}$ and $\mathbf{T^{in}}$ be the vectors of outlet and inlet temperatures of CRAC units and compute nodes. Also, let $\mathbf{A}$ be a matrix of cross-interference coefficients where each element $\alpha(i, j)$ represents the percentage of heat transferred from CRAC unit or node i to CRAC unit or node j. We can then calculate our temperatures as [39]

$$\mathbf{T^{in}} = \mathbf{A}\mathbf{T^{out}}. \tag{10}$$

If we let $\mathbf{T^{redline}}$ be the vector of redline temperatures, the thermal constraint is the element-wise inequality

$$\mathbf{T^{in}} \leq \mathbf{T^{redline}}. \tag{11}$$

TABLE 3.1. % Performance Degradation Due to Task Co-Location

memory intensity of task type	heavy	medium	light
heavy	100	30	5
medium	50	20	3
light	6	4	1

3.2.5. CO-LOCATION INTERFERENCE. Tasks competing for shared memory in multi-core processors can cause severe performance degradation, especially when competing tasks are memory-intensive [25]. The memory intensity of a task refers to the ratio of operations performed in the CPU (e.g., floating point) to the number of bytes accessed in main memory. Based on experimental data from [25] that measures the execution time slowdown of some SPEC benchmarks, we create the degradation table shown in Table 3.1 that gives the performance degradation for a task type of a categorized memory intensity in row i when co-located with another task type of a given

memory intensity in column j, given by $PD(i,j)$. For simplicity, we assume that a task type's memory-intensity is represented by one of three categories: heavy, medium, or light. For example, when a "heavy" task type is co-located with a "medium" task type, the PD value of 30% (from Table 3.1) corresponds to a 30% increase in execution time.

We can probabilistically determine if a task type i would be executing on a core k by considering the $DF(i,k)$ value as the probability that core k is running a given task type i. Using these values, we can calculate the probability that cores within the same processor are executing any combination of task types to help estimate performance degradation caused by co-location. On a dual-core processor with cores k and l, we can estimate the co-located execution time of task type i on core k, $CET^{dual}(i,k)$, as

$$CET^{dual}(i,k) = \frac{1}{ER(i,k)} \left(1 + \sum_{t=1}^{T} \left(DF(t,l) \cdot PD(i,t) \right) \right). \tag{12}$$

Because execution rate is the reciprocal of execution time, the co-located execution rate for a dual-core processor, $CER^{dual}(i,k)$, is $1/CET^{dual}(i,k)$. The summation in Equation 12 represents the the slowdown observed of task i on core k when the co-located core l is executing task types assigned to it, weighted by the desired fractions of time those tasks would execute on core l.

With a processor containing more than two cores, the performance degradation is additive based on the combination of task types executing on the cores co-located with core k. For example, let us assume a processor with three cores k, x, and y and a "heavy" task type is executing on core k. If assigning a "heavy" task type to core x increases the execution time of the task type on core k by 100% (from Table 3.1), then assigning a "medium" task to core y increases the original execution time of the task type on core k by an additional 30% (from Table 3.1), resulting in a total execution time increase of 130%. We observed this trend in experimental data from our recent

work (e.g., the Streamcluster application from the PARSEC benchmark suite) [40]. Thus, the co-located execution time of task type i on core k on a quad-core processor with other cores x, y, and z, $CET^{quad}(i,k)$, is

$$CET^{quad}(i,k) = \frac{1}{ER(i,k)} \cdot \left(1 + \sum_{a=1}^{T}\sum_{b=1}^{T}\sum_{c=1}^{T}(DF(a,x)\cdot DF(b,y)\cdot DF(c,z)\cdot [PD(i,a)+PD(i,b)+PD(i,c)])\right) \tag{13}$$

The summation in Equation 13 represents the probability that task type a is executing on core x, task type b is executing on core y, and task type c is executing on core z multiplied by the sum of the performance degradation values of those particular task types. Again, execution rate is the reciprocal of execution time, therefore the co-located execution rate for a quad-core processor, $CER^{quad}(i,k)$, is $1/CET^{quad}(i,k)$.

Let $CER(i,k)$ be equal to $CER^{dual}(i,k)$ if k is on a dual-core processor and equal to $CER^{quad}(i,k)$ if k is on a quad-core processor. To assist our resource management techniques in estimating actual reward rate under the effects of co-location interference, we introduce a new objective called *co-located reward rate* (CRR), calculated as

$$CRR = \sum_{i=1}^{T}\left(r_i \cdot min(\sum_{k=1}^{NC} CER(i,k), \lambda_i)\right). \tag{14}$$

Thus, $CRR \leq RR$.

The goal of this study is to maximize CRR when subject to power consumption and thermal constraints. The total power consumed by both CRAC units and compute nodes must be less than the power constraint, denoted ϕ. The thermal constraint is defined in (11). In the next section, we describe several approaches to solve this problem.

3.3. HEURISTICS

3.3.1. GREEDY HEURISTIC. We designed a greedy approach similar to those in [41, 42] to assign task types to cores. Our greedy heuristic (Alg. 1) iteratively assigns task types to cores to find the most efficient mapping, where we define efficiency for allocating a particular task type to a core as the ratio of performance (ECS) to the power consumption. For a task of type i on a node of type j in P-state k, we define the efficiency of that mapping, EFF(i,j,k), as

$$EFF(i,j,k) = ECS(i,j,k)/APC(i,j,k). \tag{15}$$

We start by finding the P-states with the highest EFF values for all task types and node types (line 1); all other P-states are not considered. All $T \times NNT$ task type to node type pairings are then sorted by their efficiency in descending order (line 2). At each iteration of the heuristic, the first pairing (i.e., most efficient) is selected and a single core of the chosen node type is assigned the designated task type by assigning that core a 100% desired fraction of time for that task type (lines 4-5). After making an assignment, that core is removed from consideration (line 6). If there are no cores available within the chosen node type (i.e., all cores within nodes of that node type been assigned a task type), that pairing is removed from consideration (lines 7-8). The CRAC outlet temperatures are set to the redline temperature, and then the outlet temperatures of all CRAC units are iteratively decreased by one degree until the thermal constraints are met (line 9). The algorithm repeats using the next pairing until the power constraint is violated or the execution rates for all task types meet or exceed their arrival rates.

3.3.2. GENETIC ALGORITHM.

3.3.2.1. *Overview.* We also designed a genetic algorithm (GA) based on the Genitor GA [41, 43] to solve our optimization problem. Our genetic algorithm in this study operates on a population

Algorithm 1 Pseudo-code for our greedy heuristic

1. find most efficient P-state for all task type/node type pairs
2. sort all task type/node type pairs by efficiency
3. **while** power constraint not violated **do**
4. choose first task type/node type pair
5. assign 100% desired fraction of time for selected task type to a single core from selected node type
6. remove core from future consideration
7. **if** no cores within selected node type available
8. remove task type/node type pair from consideration
9. set CRAC outlet temperatures to hottest temperatures such that thermal constraints are met
10. **end while**

of 200 chromosomes. Each chromosome represents one possible solution (i.e., a complete resource allocation). A chromosome consists of a collection of $|NC \times T|$ genes, where each gene is a pair of $DF(i,k)$ and $PS(i,k)$ values representing the desired fraction of time and P-state of a task type/core combination. The initial population is generated by assigning random desired fractions of time and P-states to each gene within the chromosome, and then normalizing the desired fractions of time so that cores cannot spend greater than 100% of their time executing tasks. After the initial population generation, the chromosomes in the population are evaluated and ranked by co-located reward rate (CRR). We then perform crossover and mutation that alter existing solutions to generate offspring chromosomes. After the offspring are generated, local search is performed on the offspring to meet the power consumption and thermal constraints. The population is then evaluated and ranked, and subsequently the population is trimmed to its original size by eliminating the least-fit chromosomes.

3.3.2.2. *Crossover and Mutation.* Crossover starts by selecting two parent chromosomes using a linear bias [43] and two points are generated such that $x < y \leq |NC|$. All genes for cores ranging from x to y are swapped among parent chromosomes to create two offspring solutions.

Mutation is probabilistically performed on offspring chromosomes to introduce perturbations

Algorithm 2 Pseudo-code for our local search technique

```
1. while power and thermal constraints not met do
2.    set CRAC outlet temperatures to hottest temperatures such that thermal constraints are met
3.    find node with highest temperature (node j)
4.    while node j is hottest node do
5.       choose random core/task type combination from node j
6.       increase P-state assignment by 1 if not already in maximum P-state
7.    end while
8. end while
```

in the genes, allowing a broader search. Offspring chromosomes have a probability p_m of being mutated (empirically set to 0.1). If a chromosome is selected for mutation, each gene has a probability p_g of being mutated (empirically set to 0.05). If a gene is selected for mutation, the desired fraction of time and P-state for its task type on this core are set to random values. Then the chromosome is normalized so that the desired fractions of time for all cores sum to 100%.

3.3.2.3. *Local Search.* We perform a local search on offspring chromosomes that sets CRAC outlet temperatures and P-states such that the power consumption and thermal constraints are met. The pseudo-code for our local search is given in Alg. 2. The step described in line 2 is performed by setting the CRAC outlet temperatures to the redline temperature, and then the outlet temperatures of all CRAC units are iteratively decreased by one degree until the thermal constraints are met. The steps performed in lines 3 to 6 have a two-fold effect to reduce power consumption. First, increasing the P-state assignment (i.e., reducing the power and frequency) directly reduces the power consumption of the node. Second, the CRAC units may be able to run with a higher outlet temperature, reducing the overall power required to maintain redline temperatures.

3.3.3. NON-LINEAR PROGRAMMING APPROACH. We adapt a power and thermal-aware approach from [26] and improve it to include the effects of co-location by maximizing co-located reward rate instead of just maximizing reward rate as done in [26]. The problem in [26] is formulated as a non-linear program and then solved using approximations, heuristics, and linear programming. We refer the reader to [26] for the full details of this technique, but give a brief summary below.

The NLP technique is divided into three steps to solve the formulated mixed integer non-linear program for maximizing reward rate (Equation (5)) while obeying the power and thermal constraints. The first step relaxes the integer constraint on P-states (by assuming continuous P-states) and solves for the CRAC outlet temperatures and power consumption of compute cores such that reward rate is maximized and the power and thermal constraints are met. Using the core power consumption values obtained from the first step, the second step uses a simple heuristic to round the continuous power consumption values of cores to discrete P-states, while maintaining the power and thermal constraints. Lastly, a linear program is solved to maximize reward rate assuming the CRAC outlet temperatures from the first step and the discrete P-state assignments from the second step.

We improve upon this approach in two ways. First, we incorporate the knowledge of the memory intensity of tasks by considering the APC matrix (i.e., a core consumes a different amount of power based on task type) instead of assuming all task types consume the same amount of power in a given P-state. Also, the scaling of ECS values for different P-states is now a function of memory intensity in addition to clock frequency. Second, we incorporate knowledge of co-location interference in this algorithm by modifying the objective function to maximize co-located reward rate instead of reward rate in the first and third steps. We show in Section 3.5 how the modified algorithm (NLPCL) makes significantly different resource allocation decisions than NLP because it considers co-location interference.

3.4. Evaluation Setup

3.4.1. Overview. In our simulations, we consider three heterogeneous platforms of different sizes. The *small* platform consists of three CRAC units, eight task types, and 150 nodes where each node is uniformly selected from two node types. The *medium* platform uses four CRAC

TABLE 3.2. Node Types Used In Simulations

node type	Fujitsu TX140	IBM x3200	Fujitsu RX100	Acer AT110
overhead power (W)	18.9	75.2	35	21.3
number of cores	4	2	4	4
number of P-states	14	4	11	16
power consumption of a core in P-state 0 (W)	9.1	20.9	22.4	18.55

units, 500 nodes (belonging to one of of three node types) and ten task types. The *large* platform uses 60 CRAC units, 5,000 nodes (belonging to one of four node types), and sixteen task types. The power characteristics of our different node types were obtained from SPECpower_ssj2008 results [44] (see Table 3.2), and the workload characteristics were obtained from SPEC_CPU2006 [45] results for the node types listed in Table 3.2. The results for these machines using those benchmarks were collected between the years 2010 and 2012. We set the power constraint (ϕ) equal to 60% of the maximum power of the system, i.e., when all cores are executing the most CPU-intensive task type in P-state 0 with the CRAC units set to highest temperatures such that all nodes maintain redline temperatures. Simulations were implemented in a custom C++ environment on a laptop computer with a Core i7-4700HQ CPU and 8 GB of RAM in a VMware virtual machine running Ubuntu 12.04.

3.4.2. WORKLOAD. The task types and corresponding ECS matrix entries for each node type in P-state 0 are obtained from SPEC_CPU2006 results [45]. We use execution rate results for *lbm*, *bwaves*, *sphinx3*, *milc*,*cactusADM*, *calculix*, *wrf*, and *tonto* for the task types on the small platform size, add *povray* and *deallII* for the medium platform size, and then add *zeusmp*, *gromacs*, *leslie3d*, *namd*, *soplex*, and *GemsFDTD* for the large platform size. These benchmarks were chosen to enable a diverse portfolio of consumer and scientific applications on which to evaluate our optimization frameworks. Experimental data from [33] is used to quantify the memory intensity

of these benchmarks. We scale the ECS values for a task type in other P-states based on the clock frequencies of the P-states and the memory intensity of the task type. Based on experimental data from [46], the performance of the most memory-intensive SPEC_CPU2006 application (*lbm*) scaled approximately 30% to that of the clock frequency. For example, halving the frequency resulted in only a 30% increase in execution time. The most CPU-intensive application (*povray*) scaled at approximately 100% to that of the clock frequency. We scale the ECS values for task types in P-states based on the clock frequency and memory-intensity values from [33].

The reward for task types are generated based on a uniform random variable in the range [0.5,1]. The d_i values, arrival rate λ_i values, and thermal coefficient matrix $\mathbf{A}$ are generated using the same techniques as in [26].

3.4.3. CRAC UNITS. In this study, we assume that the CRAC units are homogeneous. The Coefficient of Performance (CoP) for a CRAC unit is a function of its outlet temperature, τ, given by $CoP(\tau) = 0.0068\tau^2 + 0.0008\tau + 0.458$ [38] . The air flow rate of each CRAC unit is set so that the sum of the air flow rates of the compute nodes is equal to the sum of the flow rates of the CRAC units [26].

3.5. RESULTS

For our first experiment, we examine the value of accounting for co-location in our genetic algorithm and non-linear programming techniques. We denote *GA* as our genetic algorithm that uses reward rate to evaluate chromosomes, and *GACL* as a variation that uses co-located reward rate to evaluate chromosomes. *NLP* represents the technique proposed in [26] and *NLPCL* denotes our modified approach that considers co-located reward rate and the memory intensity of task types. The results in this section that show error bars represent the 95% confidence intervals around the mean for 24 trials, with each trial varying in the ECS, deadline, reward values, and

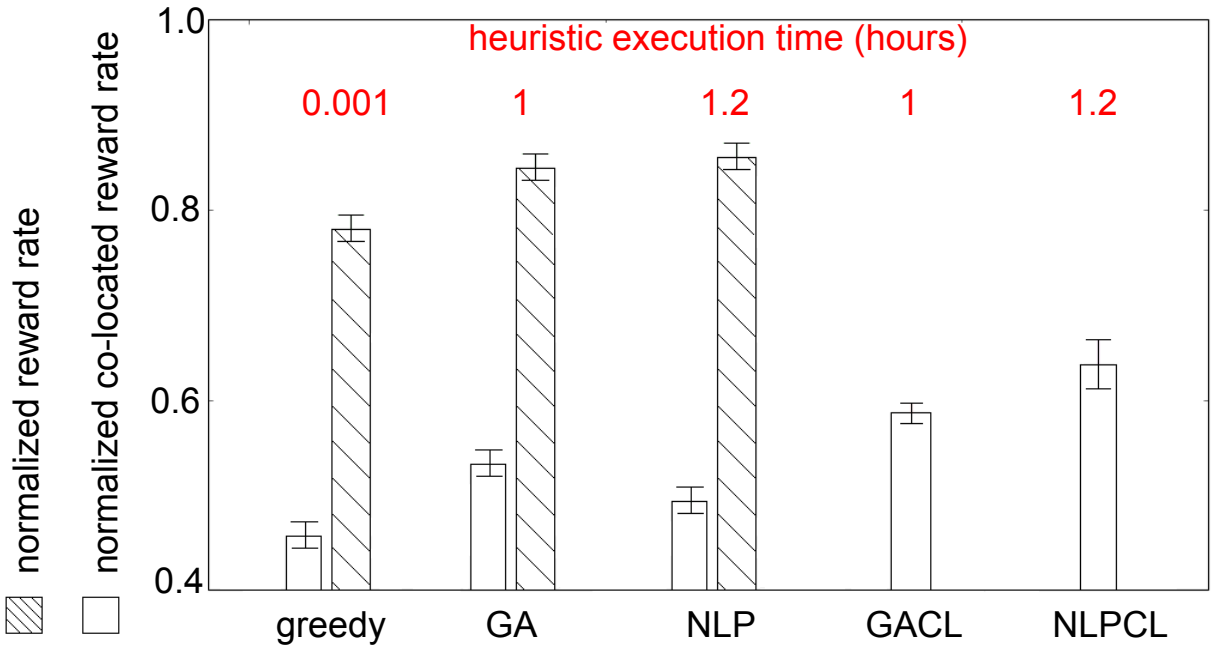

FIGURE 3.2. Comparison of co-located reward rate earned (non-hashed bars) by greedy, GA, and NLP techniques on the small platform size, and reward rate earned (hashed bars) by all techniques on the small platform size

arrival rate values.

Fig. 3.2 shows a comparison of the greedy, GA, and NLP techniques evaluated using the reward rate (RR) measure (hashed bars) that does not consider co-location effects in addition to a comparison of all resource allocation techniques evaluated using our new co-located reward rate (CRR) measure (non-hashed bars). We normalize RR and CRR by the same total reward rate that the system is capable of obtaining, i.e., when all task types are executing at rates equal to their arrival rates. To make a fair comparison, the results for the GA and GACL heuristics are recorded at the same amount of heuristic execution time it takes to perform NLP and NLPCL, respectively (about one hour for the small platform size shown). All techniques are able to meet the power and thermal constraints.

It can be observed in Fig. 3.2 that it is very important to consider the co-location interference,

as evidenced by the performance of GACL compared to GA and NLPCL compared to NLP for co-located reward rate. GACL and NLPCL significantly outperform their counterparts for co-located reward rate. By incorporating the effects of co-location interference into the fitness evaluation, GACL can make intelligent choices for selecting parent chromosomes that reduce the effects of co-location interference as evidenced by the results for co-located reward rate. Similarly, NLPCL avoids allocations that result in extreme performance degradation, such as placing two memory-intensive task types on cores within the same processor. NLP provides a very poor CRR, but NLPCL provides the best CRR.

Fig. 3.2 also shows the normalized reward rate (hashed bars) for the co-location unaware greedy, GA, and NLP approaches. The reward rate for these approaches appears to be high compared the results for co-located reward rate. However, in the presence of co-location interference, the normalized reward rate is an inaccurate metric that does not consider performance degradation from co-location interference. Thus, it overestimates the achievable reward rate.

Fig. 3.3 provides an in-depth view of the actual allocations provided by NLPCL and NLP across five (out of the 150) nodes from Fig. 3.2. Fig. 3.3a shows an example of how NLPCL allocates the desired fractions of time in cores within the same processor (i.e., those that share memory resources). In the figure, a bar represents the desired fraction of time a core is allocated to different task types. Nodes 1, 2, and 3 have quad-core processors and nodes 4 and 5 have dual-core processors. Task types in the legend are ordered by their memory-intensity. We can observe in Fig. 3.3a that NLPCL mitigates the impact of co-location interference by avoiding allocating memory-intensive task types on cores within the same processor. We can see that the processors in nodes 1, 2, and 3 allocate memory-intensive task types to only one core, and allocate CPU-intensive task types to the remaining cores to reduce performance degradation. Interestingly, one of the cores in node 4 is left to idle (no task type assigned) so the memory-intensive task type

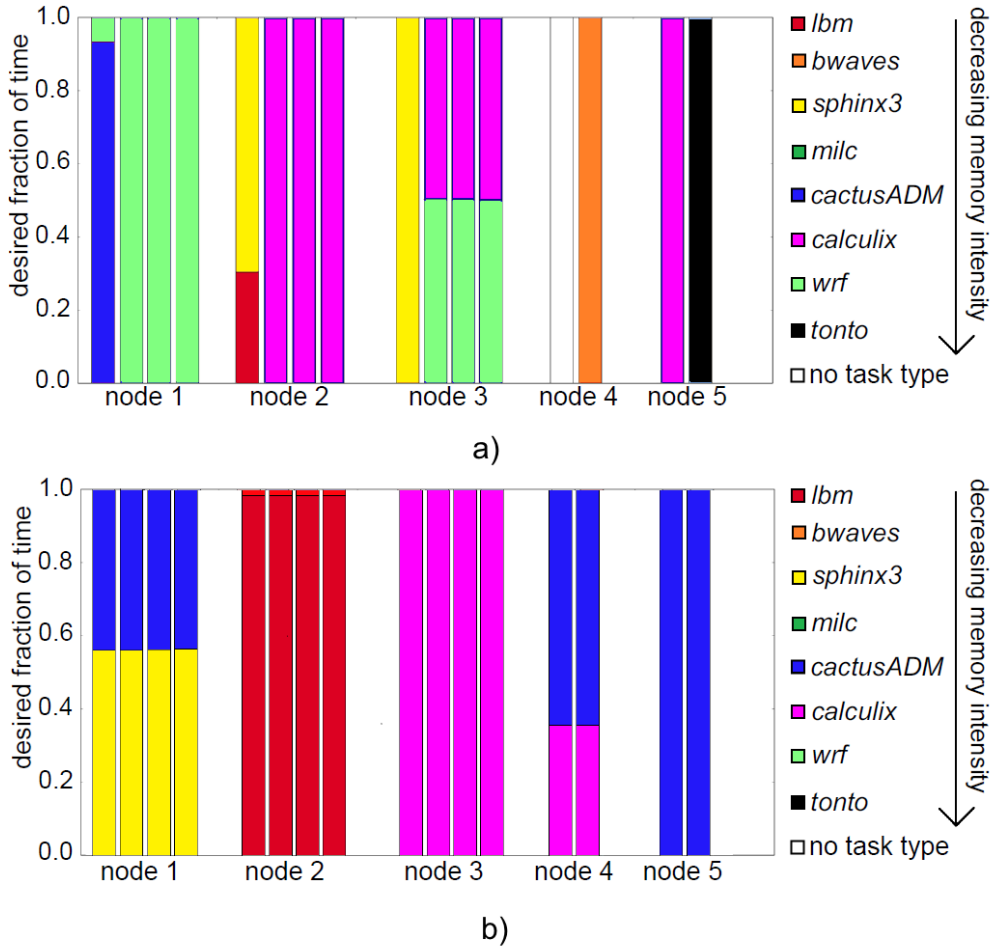

FIGURE 3.3. Desired fractions of time for task types allocated to five nodes (16 cores total) on the small platform size when using (a) NLPCL and (b) NLP.

allocated to the other core in node 4 can execute with no interference.

Fig. 3.3b shows the desired fractions of time allocations on those same nodes provided by the NLP algorithm that does not consider co-location. In our heterogeneous environment, different task types perform better (i.e., execute faster) on different node types. As it lacks knowledge of co-location interference, NLP typically chooses to allocate task types to nodes that can execute them the fastest. For example, in Fig. 3.3b, *bwaves* is the primary task type allocated to node 2, and *calculix* is the only task type allocated to node 3, as nodes of these node types were found to be the best-performing (i.e., greatest ECS values) for those task types. Without considering co-location

interference this may seem to be a "good" choice, however we can see that nodes 1 and 2 have memory-intensive task types allocated to all cores, which causes severe performance degradation. It is obvious from Figs. 3.2 and 3.3 why it is important to consider co-location interference, as NLPCL and GACL significantly outperform the techniques that do not consider co-location.

As HPC facilities become larger, it is important to address the practicality of resource allocation techniques on larger system complexities. Fig. 3.2a showed that NLPCL was able to achieve the best results for the 150 node (small) platform size. However, the primary drawback associated with the NLPCL technique is its poor scalability. Genetic algorithms hold an advantage in that they are able to provide a solution within any given algorithm runtime bounds, though typically better results are obtained the longer they run. Fig. 3.4 shows a comparison of greedy, GACL, and NLPCL on the medium platform. We recorded the performance of GACL at many different heuristic execution times to examine the benefits of running it as long as the NLPCL algorithm takes to execute (twelve hours), and as short as the greedy algorithm takes to execute (just over a minute). Even though the GACL algorithm still does not perform as well as NLPCL in the same amount of time, we can obtain a reasonably good solution from GACL whenever one is needed, e.g., for any system administrator specified heuristic runtime bounds. We can also see that GACL does not perform as well as the greedy heuristic in the same amount of time that greedy takes to execute, because GACL is only barely able to generate an initial population in the same amount of time it takes the greedy heuristic to execute.

Fig. 3.5 compares greedy and GACL at many different heuristic execution time durations. NLPCL results were excluded for the large platform size because we ran the algorithm for a week (168 hours) and it still had not finished. Similar to Fig. 3.4, greedy is able to outperform GACL in a similar amount of time, however the hill climbing capabilities of GACL provide better results given more time.

In summary, for a small platform and problem size, the better results obtained using the non-linear programming (NLPCL) approach motivates its use to generate resource allocations that optimize co-located reward rate. However, as the platform size becomes larger, the NLPCL technique becomes intractable, and GACL offers a solution in a reasonable amount of time. If a resource allocation needs to be found quickly, our greedy heuristic may be the desired approach to take. Over time, however, our GACL heuristic is able to provide better solutions and can be terminated at any point, e.g., when the steady-state of the data center changes and new execution rates and P-states need to be found.

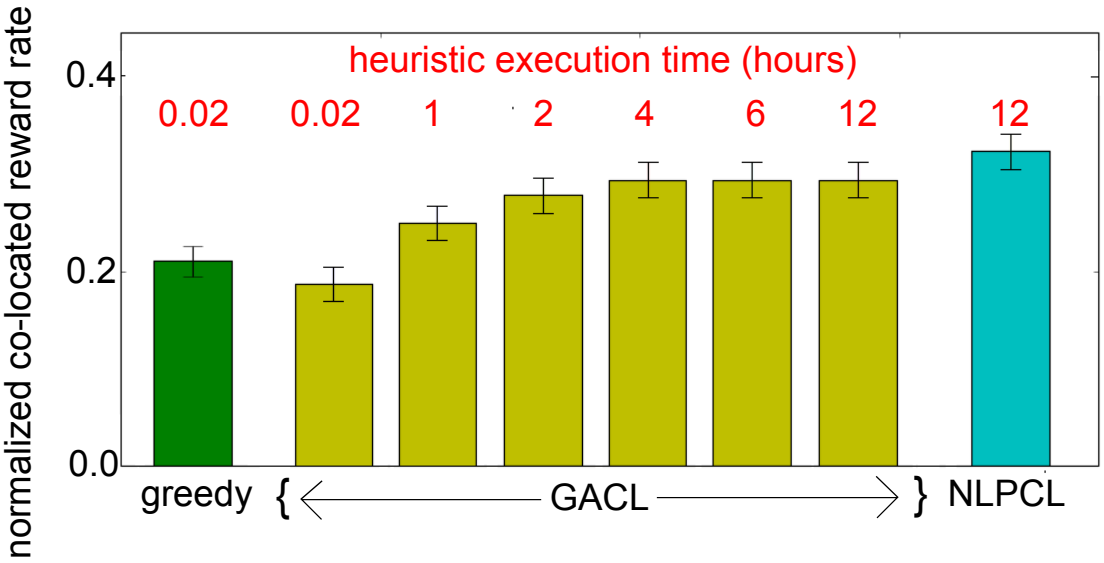

FIGURE 3.4. Co-located reward rate of greedy, GACL (across different heuristic execution time durations), and NLPCL on the *medium* platform.

3.6. CONCLUSIONS

We study the problem of maximizing the reward collected for completing tasks by their deadlines subject to power and thermal constraints. Co-location interference can have a significant impact on the execution speeds of tasks (and thus total reward). We capture these effects with our new performance metric called co-location reward rate. We designed: a greedy technique based on

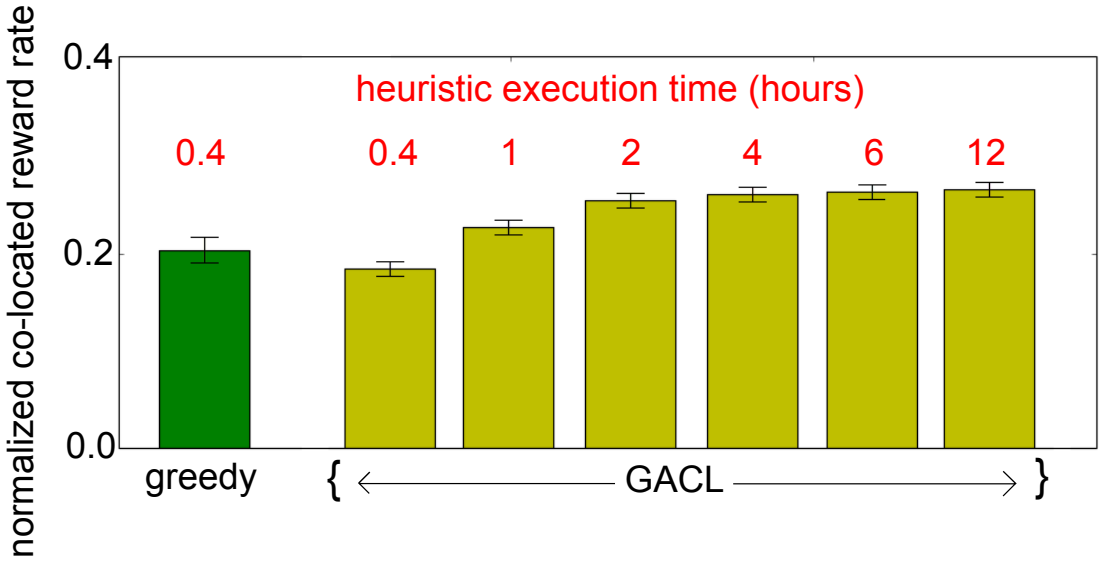

FIGURE 3.5. Co-located reward rate of greedy and GACL (across different heuristic execution time durations) on the *large* platform.

assigning tasks to machines in order of their efficiency (most performance per unit of power), a genetic algorithm combined with a local search technique that maximizes the co-located reward rate and ensures the power and thermal constraints are met, and modified a non-linear programming technique from our prior work to maximize co-located reward rate. We show the importance of considering performance degradation due to co-location interference by analyzing the co-located reward earned by our proposed techniques when they consider co-location interference and when they do not consider it.

The results show that NLPCL offers superior solutions to our other proposed techniques, but NLPCL scales poorly and the time to execute the algorithm increases dramatically as the number of nodes and task types increases. When GACL executes for the same amount of time it takes our greedy to execute, the greedy heuristic provides better results as our GACL heuristic because the GACL is not able to execute very many iterations to improve upon the initial (random) population at that time. However, our GACL algorithm provides better solutions over time and can be terminated when a solution is needed.

We have several directions we wish to pursue going forward with this study. We would like to improve our genetic algorithm by initializing the population with different seeds to help guide the search in an intelligent manner, as well as experiment with other methods of constrained optimization besides the local search technique. We are designing a fast greedy linear programming approximation approach to use as a seed and for comparison, and another greedy heuristic that intelligently avoids co-location effects by assigning desired fractions for task types that avoid interference (much like the allocation for NLPCL shown in Fig. 3.3a). We are also working on creating a more refined model of task memory intensities and co-location interference, and considering a greater variety of workloads for our evaluation. Lastly, we want to perform a sensitivity analysis on how the degree of heterogeneity of the system affects our techniques [47].

CHAPTER 4

ENERGY COST OPTIMIZATION FOR GEOGRAPHICAL LOAD BALANCING AND RESOURCE ALLOCATION WHEN CONSIDERING THERMAL, POWER, AND CO-LOCATION ON HETEROGENEOUS COMPUTING SYSTEMS[1]

4.1. INTRODUCTION

The strong success and extensive growth of cloud computing has resulted in data center operators geographically distributing their data center locations (e.g., Google [48] and Amazon [49]). Distributing data centers geographically offers benefits to the clients (e.g., low latency due to shorter communication distances). However, the primary motivation for operators to geographically distribute their data centers is to reduce electricity costs by exploiting time-of-use (TOU) electricity pricing and free renewable power [50] that varies based on time-of-day and location. Reducing electricity costs is now a focus of data center management as the annual electricity expenditure for powering the average data center has surpassed cost to purchase the equipment itself [51].

Relocating workload among geo-distributed data centers offers several benefits. First, workloads can be shifted to locations in different times zones to concentrate workload in the regions with the lowest electricity prices at that time. Second, an opportunistic distribution of the workload among data centers during periods of peak demand can allow individual servers to run in slower

[1] This work was done jointly with Ph.D. student Mark Oxley

but more energy-efficient performance states (P-states), further reducing electricity costs. Due to the ever-increasing electricity consumption of data centers, the use of on-site renewable energy sources (e.g., solar and wind) has grown in recent years. Adding renewable power to geographical load distribution (GLD) techniques can provide additional opportunities for electricity cost reduction.

The goal of our research is to design techniques for geographical load distribution that will minimize energy cost for executing incoming workloads. We use detailed models of power, temperature, and co-location interference at each data center to provide more accurate information to the geo-distributed workload manager. By considering TOU pricing and renewable power models at each data center, we design workload management techniques that distribute or migrate the workload to low-cost data centers at regular time intervals, while ensuring all of the workload completes. The novel contributions of this work are as follows:

- A hierarchical framework for the GLD problem that considers cost-minimization oriented workload management at both the geo-distributed and local heterogeneous data center level;
- A detailed data center model that considers heterogeneous compute node types, P-states, compute node temperatures, cooling power, renewable power sources, and performance degradation caused by co-location interference;
- Design of four heuristics, three of which possess varying degrees of co-location interference prediction knowledge to demonstrate and motivate the use of detailed models in workload management decisions.

The rest of the paper is organized as follows. In Section II, we review relevant prior work. Our system model is characterized in Section III. Sections IV and V describe our specific problem in

detail and the heuristics we propose to solve it. Finally, we discuss and evaluate the results of our approach in Sections VI and VII, respectively.

4.2. Related Work

There have been many recent efforts that propose methods to optimize electricity costs across geo-distributed data centers, with the fundamental decisions of the optimization problem relying on a TOU electricity pricing model. These models either predict or use traces of real utility prices, where the electricity costs during peak hours of the day (typically 8 A.M. to 5 P.M.) are much higher than off-peak hours. These TOU electricity cost models, sometimes in combination with a model for revenue generation for computation, motivates the use of optimization techniques to minimize energy cost.

Workload distribution for geo-distributed data centers has been studied in [52–58]. Knowledge of TOU pricing is typically used to either minimize electricity costs across all geo-distributed data centers (e.g., [52, 54–58]), or to maximize profits when a revenue model is included for computing (e.g., [53]). A quality of service (QoS) constraint of some form is recognized in most of the aforementioned works, typically as a queuing delay constraint [52, 54, 56]. Others incorporate QoS violations into the cost function, where a monetary penalty is associated with violating queuing delay [53], latency [55], or migration [57] service level agreements (SLAs). The modeling detail varies significantly, some works include dynamic voltage and frequency scaling (DVFS) in decision making [54], some include power consumption of the cooling system in addition to the computing system [55, 56], others consider real-world TOU pricing data [53, 54], and one considers renewable energy sources at each data center location [57]. Our research includes all aforementioned modeling aspects to assist in workload management decisions: DVFS to exploit the power/performance tradeoffs of P-states, cooling system power to include thermal awareness

and reduce cooling cost, TOU pricing data from an actual electric company, and renewable power sources at each data center. To the best of our knowledge, our work is the first to encompass all of these aspects within the GLD problem. In addition, unlike any prior work, we consider performance degradation caused by co-location interference as part of our load distribution technique; a phenomena that occurs when multiple cores within the same multicore processor are executing applications simultaneously and compete for shared resources (e.g., last-level cache or DRAM).

Similar to [57], our study considers a renewable energy source at each geo-distributed data center, a cooling system at each data center, and migration penalties associated with moving already-assigned workloads to a different data center. We differ significantly from [57] by including real TOU electricity pricing traces, consideration of DVFS P-state decisions in our management techniques, and integrating interference effects caused by the co-location of multiple tasks to cores that share resources.

4.3. System Model

4.3.1. Overview. We propose a hierarchical framework for a geo-distributed resource manager (GDRM) that consists of a high-level manager to distribute distribute incoming workload requests and migrate already-allocated requests to geographically distributed data centers. Each data center has its own local workload management system that takes the workload assigned to it by the GDRM and maps requests to processing elements (compute cores) within the data center. We first describe the system model at the geo-distributed level and then provide further details into the models of components at the data center level.

4.3.2. Geo-distributed Level. We assume a static workload management scheme that distributes the incoming workload to geo-distributed data centers. The goal of the GDRM is to minimize the total monetary cost of the system while servicing all requests. We divide time evenly

into intervals called *epochs*. For the sake of simplicity, each epoch represents an hour of time (T^e), and a 24-epoch period represents a full day.

We assume that the beginning of each epoch represents a steady-state scheduling problem where we assign *execution rates* of a set of I workload task types to D data centers. A task type $i \in I$ is characterized by its arrival rate AR_i, and its estimated computational speeds on each of the heterogeneous compute nodes in all P-states. The assignment problem at the geo-distributed level is to map execution rates for each task type i to each data center $d \in D$ such that total energy *cost* across all data centers is minimized, and the execution rates of all task types exceed their arrival rates (i.e., all tasks completed without being dropped).

Real-world electricity prices are not constant, but rather follow a TOU pricing model where the cost of electricity varies based on the time of day (Fig. 4.1). Prices are greater when total electrical grid demand is high, and are reduced during periods when electrical grid demand is low. A group of data centers spread across time zones can take advantage of this by shifting more workload to the data center with the lowest electricity cost at that time. The problem becomes more difficult when considering the amount of renewable power at each data center, the heterogeneity of compute nodes within a data center, and the constraint that the entire workload must complete. Having information about TOU electricity pricing, a prediction of the amount of renewable energy at each data center, the incoming workload, and the execution speeds of task types on the heterogeneous compute nodes lets our GDRM make intelligent decisions for allocating the workload.

For each epoch τ, we assign a desired data center execution rate $ER^{DC}_{d,i}$ for each task type i to each data center d such that the total execution rate for all task types exceed (or equal) the

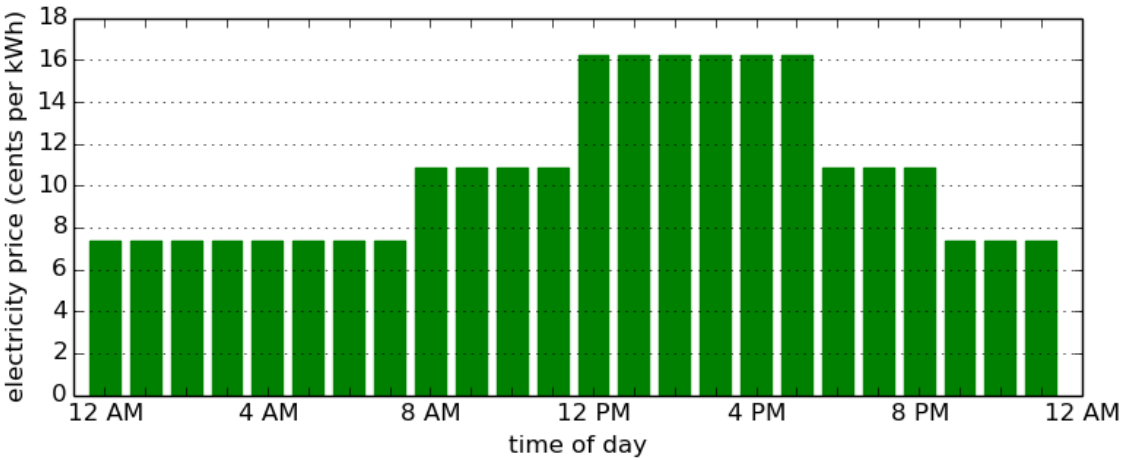

FIGURE 4.1. Time-of-use (TOU) electricity pricing, PG&E Schedule E-19 [1].

corresponding arrival rate, AR_i, ensuring the workload is completed. That is,

$$\sum_{d=1}^{D} ER_{d,i}^{DC}(\tau) \geq AR_i(\tau), \quad \forall i \in I. \tag{16}$$

4.3.3. DATA CENTER LEVEL.

4.3.3.1. *Overview.* Each data center d houses NN_d compute nodes that are arranged in hot aisle/cold aisle fashion (Fig. 4.2), and a cooling system comprised of NCR_d computer room air conditioning (CRAC) units. There exists heterogeneity among compute nodes. Each compute node n is of a compute node type, where node types vary in their execution speeds, power consumption characteristics, and number of cores. Cores within a compute node are homogeneous, and each core is DVFS-enabled to allow independent configuration of its P-states. The number of cores in node n is NCN_n, and NT_k is the compute node type to which core k belongs.

4.3.3.2. *Core Execution Rates.* Recall that our GDRM determines the distribution of each task type i between all data centers. At each data center d, the sum of execution rates of all cores that are assigned to execute task type i must exceed or equal $ER_{d,i}^{DC}(\tau)$. We assume that we know the *estimated computational speed* (ECS) of any task of type i on a core of node type n in P-state p, ECS(i, n, p), determined using historical, experimental, or analytical techniques [37, 35].

The execution rate of task type i on core k is the product of the assigned desired fraction, $DF_{i,k}(\tau)$, of time core k spends executing tasks of type i and the execution speed that core executes tasks of type i in P-state $PS_{i,k}(\tau)$. That is, the execution rate of task type i, on core k is

$$ER_{i,k}^{core}(\tau) = DF_{i,k}(\tau) \cdot ECS(i, NT_k, PS_{i,k}(\tau)). \tag{17}$$

At the data center level, we assign $DF_{i,k}(\tau)$ and $PS_{i,k}(\tau)$ such that power is minimized (see Section 4.3.3.3), and the execution rates of all task types on cores in data center d exceeds the execution rate assigned by the GDRM, ensuring that the arriving workload is fully executed. That is,

$$\sum_{n=1}^{NN_d} \sum_{k=1}^{NCN_n} ER_{i,k}^{core} \geq ER_{i,d}^{DC} \quad \forall i \in I, \forall d \in D. \tag{18}$$

4.3.3.3. *Power Model.* The power consumption of a compute node consists of the overhead ("base") power consumption and dynamic power consumed by cores executing tasks. We define O_n as the overhead power consumption (fixed power consumption regardless of workload, resulting from system components such as storage, network interfaces, etc.) of compute node n. Let APC(i, NT_k, $PS_{i,k}(\tau)$) be the average power consumed by core k in a node of type NT_k when executing tasks of type i in P-state $PS_{i,k}(\tau)$ during epoch τ. The power consumption of node n during epoch τ, $PN_n(\tau)$, is

$$PN_n(\tau) = O_n + \sum_{k=1}^{NCN_n} \sum_{i=1}^{I} APC(i, NT_k, PS_{i,k}(\tau)) \cdot DF_{i,k}(\tau). \tag{19}$$

The power consumed by a CRAC unit, $PCR_{d,c}(\tau)$, is a function of the heat removed at that CRAC unit and the Coefficient of Performance (CoP) of the CRAC unit [38], calculated using Eq. 5 of [2].

4.3.3.4. *Renewable Energy Model.* Each data center is equipped with and partially powered by a renewable power source. Each location can have either solar power, wind power, or some combination of both. Solar power E_d^{solar} and wind power E_d^{wind} (both kWh) are calculated for each data center d as an average per epoch τ (Equations 20 and 21 are from [59–61]). A_d^{solar} is the total active area of all solar panels, and A_d^{wind} is the total swept rotor area of all wind turbines. The solar-to-electricity and wind-to-electricity conversion efficiency are given by α and β, respectively. Lastly, $s_d(\tau)$ is the average solar irradiance, $v_d(t)$ is the wind speed, and $\rho_d(\tau)$ is the air density, as averaged for data center d during epoch τ. The total renewable energy, $R_d(\tau)$, available at data center d during epoch τ is the sum of the wind and solar energy available at that time. We use these models with historical data to predict the renewable power available at each data center.

$$E_d^{solar}(\tau) = \alpha \cdot A_d^{solar} \cdot s_d(\tau) \cdot T^e \tag{20}$$

$$E_d^{wind}(\tau) = \beta \cdot \frac{1}{2} \cdot A_d^{wind} \cdot \rho_d(\tau) \cdot v_d(\tau)^3 \cdot T^e \tag{21}$$

$$R_d(\tau) = E_d^{solar}(\tau) + E_d^{wind}(\tau) \tag{22}$$

4.3.3.5. *Thermal Model.* Within each data center the compute nodes generate heat, and this heat is removed by the CRAC units. Due to the airflow within a data center, the heat generated by a node can increase the temperature of the air flowing through nearby nodes. Using the notion

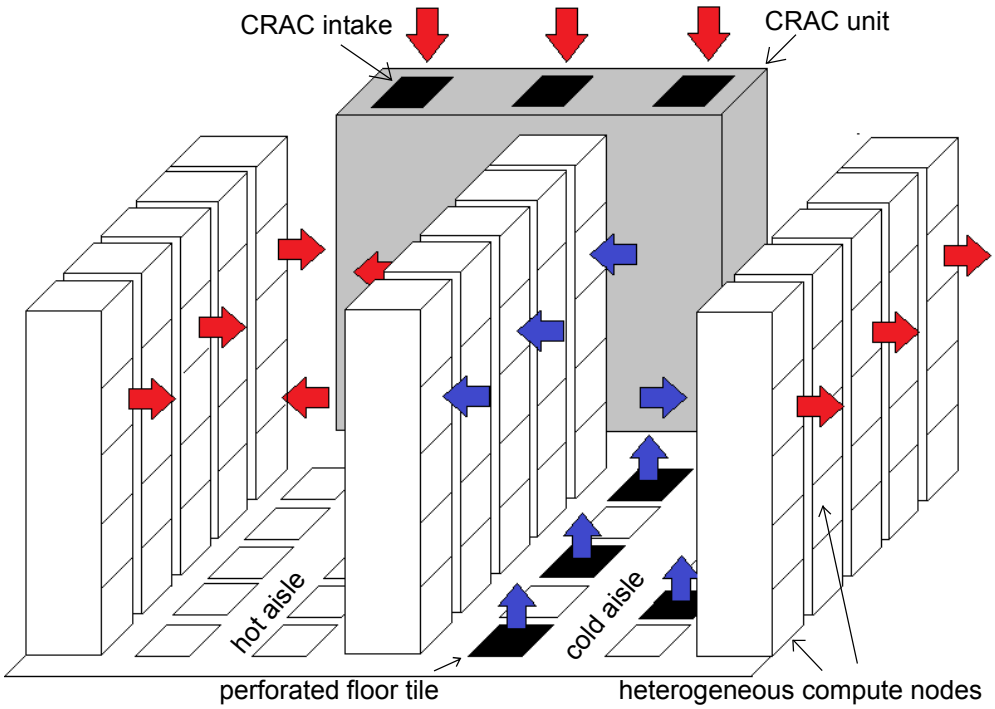

FIGURE 4.2. Data center in hot aisle/cold aisle configuration [2].

of thermal influence indices [62] that we derive using computational fluid dynamics simulations, we can calculate the steady-state temperatures at compute nodes and CRAC units in each data center. Since we assume the same physical layout for each of the data centers (Fig. 4.2), we derive these thermal influence indices for one data center, and assume they are the same for all other data centers.

The outlet temperature of each compute node is a function of the inlet temperature, the power consumed, and the air flow rate of the node. The inlet temperature of each compute node is a function of the outlet temperatures of each CRAC unit and the outlet temperatures of all compute nodes [2]. Lastly, for all nodes the inlet temperature of each node is constrained to be less than or equal to the red line temperature (maximum allowable node temperature).

4.3.3.6. *System Electricity Cost.* The electricity price at data center d during epoch τ is defined as $E_d^{price}(\tau)$. Let Eff_d be the approximation of power overhead in data center d due to the

inefficiencies of power supply units and uninterruptable power supplies. The total electricity cost for data center d during epoch τ, $PC_d(\tau)$, is defined as

$$PC_d(\tau) = E_d^{price}(\tau) \cdot \left[\left(\sum_{c=1}^{NCR_d} PCR_{d,c}(\tau) + \sum_{n=1}^{NN_d} PN_n(\tau) \right) \cdot Eff_d - R_d(\tau) \right]. \tag{23}$$

4.3.3.7. *Server Activation/Deacitvation Cost.* At each data center the number of servers of each server type that are in use changes frequently between epochs. Inactive servers are placed in a sleep state, but entering and exiting this sleep state takes a non-negligible amount of time due to the actions required in both hardware and software to transition the system between states. Each server that is active is considered to be active for the entire epoch, which requires that any server transitioning to/from a sleep state do so during the epoch following/prior the current epoch, respectively.

For each data center d, let $N_{d,j}^{start}(\tau)$ be the number of servers of type j that are inactive during epoch τ and active during epoch $\tau + 1$, and let $N_{d,j}^{stop}(\tau)$ be the number of servers of type j that are active during epoch $\tau - 1$ and inactive during epoch τ. Let P_j^S, P_j^D, and P_j^{Sleep} be the average static power, average peak dynamic power, and average sleep power for server type j, respectively, with the average utilization of server type j defined as ϕ_j^E. Let the coefficient to approximate CRAC unit power at data center d be CUP_d. We assume each data center contains the same number of nodes, however each data center is heterogeneous in the sense that the number of nodes belonging to each node type among data centers varies. Therefore, let J_d be the set of server types in data center d. Let T^S be the time required for a server to transition to/from a sleep state. Recall that T^e is the duration of an epoch. The server assignment cost AC_d for data center d during epoch τ is defined by Eq. 24.

$$AC_d(\tau) = \sum_{j \in J_d} E_d^{price}(\tau) \cdot Eff_d \cdot \left(1 + \frac{1}{CUP_d}\right) \cdot \frac{T^S}{T^e} \cdot \left(\phi_j^E P_j^D + P_j^S - P_j^{Sleep}\right) \cdot \left(N_{d,j}^{start}(\tau) + N_{d,j}^{stop}(\tau)\right) \tag{24}$$

4.3.3.8. *Co-Location Interference Model.* Tasks competing for shared memory in multicore processors can cause severe performance degradation, especially when competing tasks are memory-intensive [25]. The memory-intensity of a task refers to the ratio of last-level cache misses to the total number of instructions executed [63]. We adapt a linear regression model from [63] that uses a set of features (i.e., inputs) based on the current applications assigned to a multicore processor to predict the execution time of a target application *i* on core *k*. A description of the features we use, in addition to the symbols representing each feature, is given in Table 4.1.

In a linear model, the output is a linear combination of all features and their calculated coefficients. We classify the task types into memory-intensity classes, and calculate the coefficients for each memory-intensity class using the linear regression model. If we denote u, v, w, x, and y as the linear model coefficients for feature symbols A, B, C, D, and E, respectively, plus the constant term z, the equation for co-located execution time of a task type i of memory-intensity class m on core k ($CET_{i,k}(\tau)$) is

$$CET_{i,k}(\tau) = u_{m,k} \cdot A_{i,k} + v_{m,k} \cdot B_{i,k} + w_{m,k} \cdot C_{i,k} + x_{m,k} \cdot D_k + y_{m,k} \cdot E_{i,k} + z_{m,k} \, . \tag{25}$$

The execution rate is the reciprocal of the execution time. Therefore the co-located execution rate for task type *i* on core *k*, $CER_{i,k}^{core}(\tau)$, is $1/CET_{i,k}(\tau)$. The total execution rate for task type i in epoch τ is therefore given by

$$CER_i(\tau) = \sum_{d=1}^{D} \sum_{k=1}^{NC_d} CER_{i,k}^{core}(\tau). \tag{26}$$

Because co-location interference degrades the execution rate of task types, it could cause some tasks to be unable to finish if task types are allocated to cores based on execution rates that do not consider degradation effects (Eq. 17). To allocate tasks to cores when considering co-location interference, some of our techniques use knowledge of $CER_{i,k}^{core}$ to judge actual execution rates more accurately than techniques that do not consider co-location interference. When considering co-location the execution rate constraint becomes

$$\sum_{k=1}^{NC_d} CER_{i,k}^{core}(\tau) \geq ER_{d,i}^{DC}(\tau), \quad \forall i \in I, \forall d \in D. \tag{27}$$

TABLE 4.1. description of features for predicting execution time

name	symbol	description
number of co-located applications	$A_{i,k}$	the number of applications co-located with target application i on core k
base execution time	$B_{i,k}$	base execution time of target task i when executing alone on core k in P-state $PS_{i,k}$
frequency of target core	$C_{i,k}$	clock frequency of target core k when running target application i in P-state $PS_{i,k}$
average memory intensity	D_k	average memory intensity of all applications on the multicore processor that contains core k
target memory intensity	$E_{i,k}$	memory intensity of target application i on core k

4.4. PROBLEM FORMULATION

We consider a scenario where there are multiple data centers with a shared workload. The system as a whole is under-subscribed, so all tasks must be completed without dropping. Even though the system is under-subscribed, note that individual data centers may be executing at full capacity. The tasks originate off-site from the data centers, and it is assumed that the transmission time and

cost from a task origin to a data center is equivalent for all data centers.

4.5. ALGORITHM DESCRIPTIONS

The GDRM allocates the incoming workload to not only individual data centers, but also specific servers within each data center. Four GDRM heuristics are proposed, each having different levels of detail of the system available to it.

4.5.1. FORCE DIRECTED LOAD DISTRIBUTION HEURISTICS. Force-directed load distribution (FDLD) is a variation of force-directed scheduling [64], a technique often used for optimizing semiconductor synthesis. FDLD is an iterative heuristic that selectively performs operations to minimize system forces until all constraints are met. Because information for a detailed co-location interference model may not always be available, we design three FDLD heuristics (FDLD-SO, FDLD-TAO, and FDLD-CL) that each possess different amounts of information to solve this problem.

FDLD-SO, inspired by [65], uses simple over-provisioning to compensate for performance degradation due to co-location. This technique over-provisions all task types equally by scaling estimated task execution rates by the factor ϕ^C. The second technique, FDLD-TAO, improves upon FDLD-SO by using task aware over-provisioning to estimate co-location effects for each task type by a factor specific to each task type i, ϕ_i^C. For both FDLD-SO and FDLD-TAO, the degree of over-provisioning (ϕ^C and ϕ_i^C, respectively) is determined empirically. Lastly, the FDLD-CL heuristic uses the co-location models given in Sec. 4.3.3.8 to account for co-location effects when calculating task execution rates. All versions of FDLD consider a system implementation where the computing time of each server core is evenly divided among its assigned tasks.

The fundamental operation of all FDLD variants is described in Algorithm 3. To generate the

initial solution, every server in every data center in every epoch is assigned to execute all task types (step 1). Each operation of the FDLD removes one instance of one task type from a single server, selecting the task to remove that would result in the lowest total system force, F^S (steps 3-20). F^S is the sum of the execution rate forces ($F^{ER}(\tau)$) and cost forces ($F^C(\tau)$) across all epochs.

The execution rate force F^{ER} is the ratio of task execution rate (calculated during steps 6-11) to task arrival rate. Task execution rate is a function of the P-state of the server the task is executing on, but the FDLD is not designed to make DVFS decisions to set the execution rates of task types, and therefore an average execution rate must be determined for all task types using the average server utilization factor ϕ_j^E for each server type j. Let $ER_{j,i}(P_{MAX})$ and $ER_{j,i}(P_0)$ be the execution rates of task type i running on a single core of a server of type j in the highest numbered P-state and and lowest numbered P-state, respectively. Therefore, the equivalent single core execution rate $R_{j,i}$ of task type i on server type j is

$$R_{j,i} = ER_{j,i}(P_{MAX}) + [ER_{j,i}(P_0) - ER_{j,i}(P_{MAX})]\,\phi_j^E \tag{28}$$

Let $N_{d,j}$ be the number of servers of type j in data center d. Let $W_{d,j,m}(\tau)$ be the set of instances of task type i placed on server m of server type j in data center d during epoch τ. Let $Q_{d,j,i}(\tau)$ be the equivalent number of servers of type j running task type i in data center d during epoch τ, given by

$$Q_{d,j,i}(\tau) = \sum_{m=1}^{N_{d,j}} \begin{cases} \frac{1}{|W_{d,j,m(\tau)}|} & \text{if } i \in W_{d,j,m}(\tau) \\ 0 & else \end{cases} \tag{29}$$

The average estimated execution rate $ER_{j,i}^E(P)$ of task type i on machine type j in P-state P during epoch τ, when using either the FDLD-SO or FDLD-TAO versions, is given by Eq. 30.

Note that $ER^E_{j,i}(P)$ is subject to the constraint given in Eq. 31. To compensate for performance degradation due to co-location effects, server over-provisioning is accomplished by the factor F. F is replaced by either ϕ^C or ϕ^C_i in Eq. 30 when using either FDLD-SO or FLDB-TAO, respectively.

$$ER^E_i(\tau) = \sum_{d=1}^{D} \sum_{j \in J_d} K_j \cdot R_{j,i} \cdot F \cdot Q_{d,j,i}(\tau) \tag{30}$$

$$ER^E_i(\tau) \geq AR_i(\tau) \quad \forall i \in I \tag{31}$$

The execution rate force F^{ER} is calculated using Eq. 32. When using the FDLD-CL heuristic, the term Z is replaced by $CER_i(\tau)$, and is replaced by $ER^E_i(\tau)$ when using either FDLD-SO or FDLD-TAO. Observe that $F^{ER}(\tau)$ will decrease to zero as the ratio of Z to $AR_i(\tau)$ decreases to one.

$$F^{ER}(\tau) = \sum_{i \in I} e^{\left(\frac{Z}{AR_i(\tau)} - 1\right)} - 1 \tag{32}$$

Recall that $R_d(\tau)$ is the renewable power available at data center d during epoch τ. For all FDLD variants, let $PC^E_d(\tau)$ be the estimated power cost at data center d during epoch τ, calculated as

$$PC^E_d(\tau) = E^{price}_d(\tau) \cdot \left(\sum_{j \in J^d} \sum_{m=0}^{N_{d,j}} P^E_{d,j,m} \cdot \left(1 + \frac{1}{CUP_d}\right) \cdot Eff_d - R_d(\tau) \right). \tag{33}$$

$$P^E_{d,j,m} = \begin{cases} P^{Sleep}_j \text{ if } |W_{d,j,m}| = 0, \text{ else:} \\ \\ \phi^j P^D_j + P^S_j \end{cases} \tag{34}$$

Let $C^{actual}_d(\tau)$ be sum of power ($PC^E_d(\tau)$) and allocation ($AC_d(\tau)$) costs incurred at data center d during epoch τ. Let $C^{max}_d(\tau)$ be the maximum real power cost possible at data center d, calculated using Eq. 35. The cost force F^C can then be calculated with Eq. 36. Observe that the value of F^C goes to zero as the ratio of $C^{actual}_d(\tau)$ to $C^{max}_d(\tau)$ decreases to zero.

$$C^{max}_d(\tau) = E^{price}_d(\tau) \cdot \sum_{j \in J_d} N_{d,j} \cdot \left[\phi^j P^D_j + P^S_j\right] \tag{35}$$

$$F^C(\tau) = \sum_{d=1}^{D} e^{\left(\frac{C^{actual}_d(\tau)}{C^{max}_d(\tau)}\right)} - 1 \tag{36}$$

Let N^τ be the total number of epochs being considered. The total system force across all epochs, F^S, is calculated as

$$F^S = \sum_{\tau=1}^{N^\tau} F^{ER}(\tau) + F^C(\tau). \tag{37}$$

4.5.2. GENETIC ALGORITHM HEURISTIC. We also designed a fourth heuristic; a genetic algorithm load distribution with full co-location awareness (GALD-CL). The GALD-CL heuristic (Algorithm 4) has two parts: a genetic algorithm based GDRM and a greedy heuristic serving as the fitness function of the genetic algorithm. The GALD-CL assigns fractions of the global task arrival rate to each of the data centers in the simulation (step 3), with the arrival rates of each task type i at each data center d acting as the genes of the chromosomes. Using the task arrival rates

Algorithm 3 Pseudo-code for FDLB heuristics

1. allocate an instance of each task type to every server in every data center in every epoch
2. **while**
3. **for** each server with tasks still allocated to it
4. **for** each task type on the server
5. temporarily remove task type from server
6. **if** FDLB-CL
7. estimate execution rates using Eq. 26 (CER_i)
8. **else if** FDLB-TAO
9. estimate execution rates using Eq. 30 and ϕ_i^C
10. **else if** FDLB-SO
11. calculate execution rates using Eq. 30 and ϕ^C
12. estimate power costs using Eq. 33
13. calculate F^S from F^{ER} and F^C
14. **if** execution rate constraints are not violated (Eq. for FDLB-CL, Eq. 31 FDLB-SO & FDLB-TAO)
15. add to set of possible task removal operations
16. restore task type to server
17. **if** set of possible task removal operations is empty
18. **break**
19. **else**
20. chooses and implement the task type removal operation that would result in the lowest F^S
21. **end while**
22. calculate final execution rates ($CER_i(\tau)$, $\forall i \in I$, $\forall \tau \in N^\tau$)
23. calculate final cost from sum of power costs and allocation costs ($PC_d(\tau)$ and $AC_d(\tau)$, $\forall d \in D$, $\forall \tau \in N^\tau$)

assigned to each data center by the genetic algorithm at the geo-distributed level, the local greedy heuristic assigns tasks types to execute on specific servers (steps 5-15). If the greedy heuristic finds that the task arrival rate assigned to a data center exceeds the capacity of that data center (step 16), the global arrival rates are adjusted slightly and the chromosome is evaluated once again (steps 5-15), with further adjustments made to the global allocations within the chromosome until a valid solution can be reached. The greedy heuristic has full knowledge of the entire system model, including the co-location models and task-node power models, allowing it to make highly optimal placement decisions.

The GALD-CL heuristic addresses two potential shortcomings of the FDLD variants. First, the nature of the decision making within the FDLD variants prevents them from making any kind

Algorithm 4 Pseudo-code for GALD-CL heuristic

1. create an initial population of chromosomes
2. **while** within time limit **do**
3. perform selection, crossover and mutation to create new chromosomes
4. **for** each new chromosome, evaluate:
5. **for** each data center
6. find most efficient P-state for all task type/node type pairs
7. sort all task type/node type pairs by efficiency
8. **while** power constraint not violated **do**
9. choose first task type/node type pair
10. assign 100% desired fraction of time for selected task type to a single core from selected node type
11. remove core from future consideration
12. **if** no cores within selected node type available
13. remove task type/node type pair from use
14. set CRAC outlet temperatures to hottest temperatures such that thermal constraints are met
15. **end while**
16. **if** solution is invalid
17. modify chromosome, return to step 5
18. trim population (with elitism)
19. **end while**
20. take final allocation from the best chromosome
21. calculate final execution rates ($CER_i(\tau)$, $\forall i \in I$, $\forall \tau \in N^\tau$)
22. calculate final cost from sum of power costs and allocation costs ($PC_d(\tau)$ and $AC_d(\tau)$, $\forall d \in D$, $\forall \tau \in N^\tau$)

of DVFS decisions, therefore a single P-state is chosen for each server type regardless of the tasks executing on the server. The greedy heuristic within the GALD-CL approach chooses the most efficient P-state for each task type on each server type. Second, the FDLD variants are susceptible to becoming trapped in local minima. The genetic algorithm portion of the GALD-CL approach intrinsically enables escape from local minima, allowing a more complete search of the solution space. The extra information and decision options available to the GALD-CL heuristic gives it a strong advantage over the FDLD variants.

4.6. Simulation Results

Experiments were conducted for data center groups of three different sizes: four, eight, and sixteen datacenters. Locations from the three test groups were selected from major cities around the continental United States to provide a variety of wind and solar conditions between sites and at different times of the day (Fig. 4.3). Experiments using a group size of four data centers used locations one through four, while experiments using group sizes of eight and sixteen data centers used locations one through eight and one through sixteen, respectively. The sites of each group were selected so that each group would have a fairly even East to West distribution to better exploit TOU pricing and renewable power. Each data center consists of 4,320 servers arranged in 4 aisles, and is heterogeneous within itself, having servers from either two or three of the server types given in Table 4.2, with most locations having three servers types and per-server core counts that range from 4-12 cores depending on the mix of server types used.

The electricity prices used during experiments, as shown in Fig. 4.1, were taken directly from

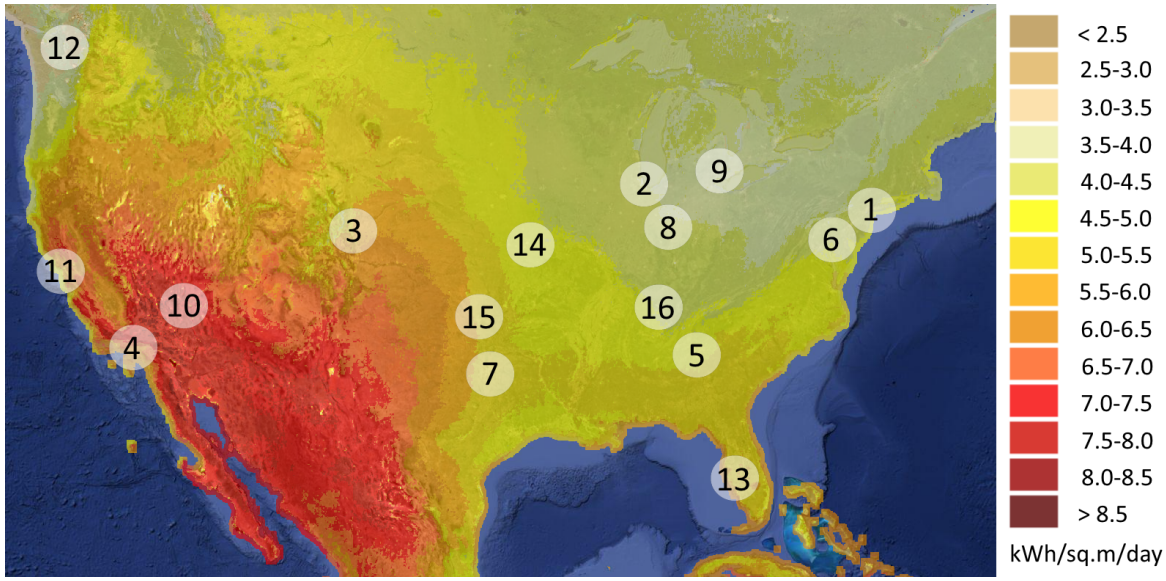

Figure 4.3. Location of simulated data centers overlaid on solar irrandiance intensity map (average annual direct normal irradiance). Wind data collected but not shown [3].

TABLE 4.2. Server Processor Types Used in Experiments

Intel processor	# cores	L3 cache	frequency range
Xeon E3-1225v3	4	8MB	0.8 - 3.20 GHz
Xeon E5649	6	12MB	1.60 - 2.53 GHz
Xeon E5-2697v2	12	30MB	1.20 - 2.70 GHz

Pacific Gas and Electric (PG&E) Schedule E-19, which is for commercial locations consuming between 500kW and 1MW [1]. Each data center had an installed renewable power generating capacity equivalent to 20% of the maximum power consumption of the location during the month with the highest generated power for that location. Renewable power at each location was either wind power, solar power, or a combination of the two. For example, a location with high average solar irradiance but low average wind speed would be restricted to having solar power only. Solar and wind data was obtained from the National Solar Radiation Database [3]. An example of the renewable power available at different locations per epoch is given in Fig. 4.4.

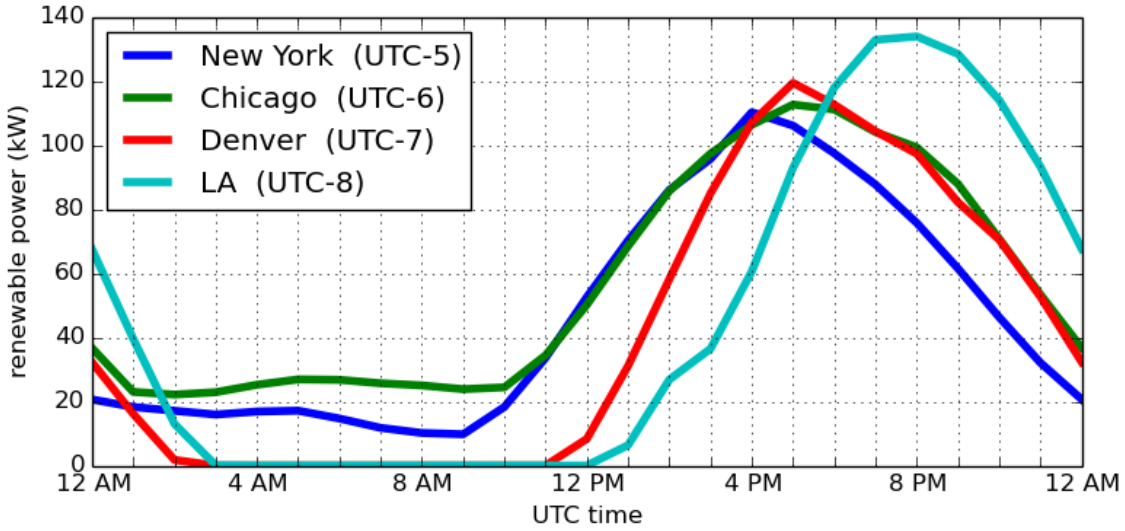

FIGURE 4.4. Example of renewable power available per hour for four locations.

Servers placed in a sleep state by a heuristic are considered to be in the Advanced Configuration and Power Interface (ACPI) server sleep state S3, where RAM remains powered on. Sleep state S3, also commonly referred to to as *suspend* or *standby*, allows greatly reduced power consumption

while still having a small latency to return to an active operating state. Sleep power for all servers is calculated as a fixed percentage of static power for each server type, assumed to be 16% based on a recent study of server power states [66]. The average server utilization factor used during FDLD allocations, ϕ^E, is set as 0.75. The coefficient to approximate CRAC unit power at data center d (CUP_d) was determined empirically by simulating workloads of multiple levels at each data center location, and its value ranged between 1.43 and 2.08 for the different configurations. The time of each epoch τ was set to be one hour. The time required to transition a server to or from a sleep state, T^S, was assumed to be five minutes.

During all tests, the GALD-CL heuristic was limited to a run time of one hour for each epoch it was solving for, to mimic the representative time of each epoch. The FDLD heuristics for four, eight, and sixteen locations completed on average in one, four, and thirteen minutes per epoch simulated, respectively.

Each simulation used a workload of five task types based on real data. Task execution times and co-located performance data were obtained from running benchmark applications on the servers listed in Table 4.2 [63]. Synthetic task arrival rates were constructed and are shown in Fig. 4.5 compared to the TOU prices for the East coast and the West coast.

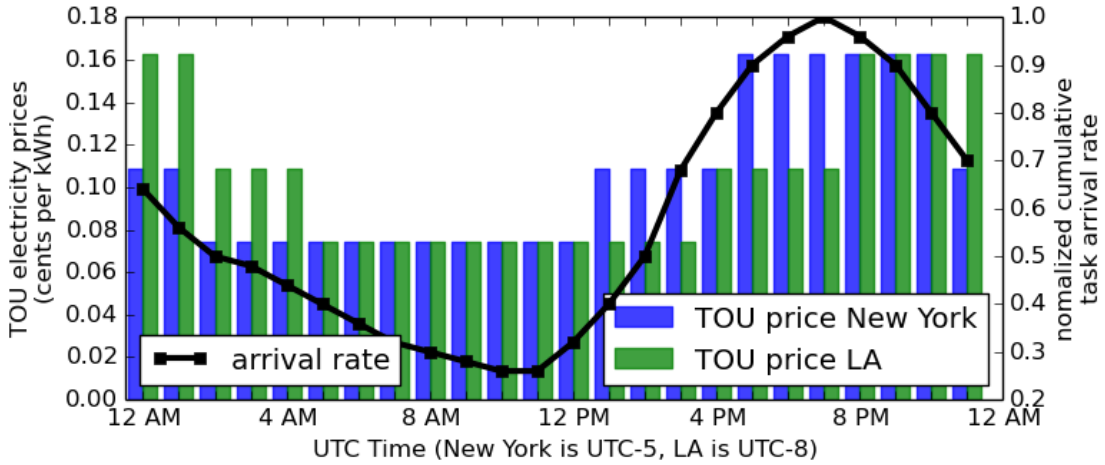

FIGURE 4.5. Comparison of system arrival rate vs. example TOU prices.

4.6.1. COST COMPARISON OF HEURISTICS. Our first set of experiments compared the cost associated with using the four heuristics described in Section V. These experiments used a data center group consisting of four locations and estimated costs over a 24-hour period (Fig. 4.6). It can be observed that the FDLD-CL technique, using the co-location models, performs the best of the FDLD variants for provisioning the minimum number of servers necessary to meet execution rate requirements. The FDLD-SO technique performed the worst, severely over-provisioning servers. The GALD-CL heuristic outperformed all other approaches. In several cases, the performance of the FDLD-CL approached that of the GALD-CL, but did not surpass it. While not performing as well as well as the GALD-CL, the FDLD variants do have the advantage of reaching a solution more quickly, which may be beneficial in some cases.

It can be observed that the epoch-to-epoch cost differences of the FDLD variants were much more irregular compared to the smooth curve of the GALD-CL. The FDLD variants are deterministic with no mechanism for escaping local minima, resulting in the more irregular values of the final solutions. The nature of the GALD-CL heuristic inherently enables it to escape from local minima, allowing for but not guaranteeing more optimal solutions.

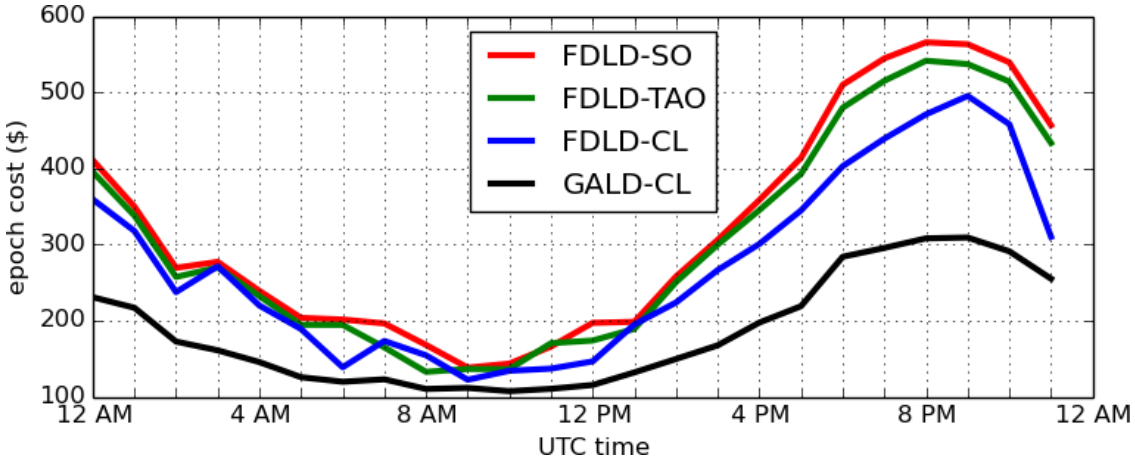

FIGURE 4.6. System costs across twenty four epoch period for each heuristic, four locations.

4.6.2. WORKLOAD TYPE ANALYSIS. The previous experiments presented in this section used a workload that was a mix of memory-intensive and CPU-intensive tasks types. Fig. 4.7 shows experiments for the FDLD-CL and GALD-CL heuristics for a group of four data centers where two additional workload types have been added: one where all of the tasks are highly CPU-intensive, and one where the tasks are highly memory-intensive. The composition of data center workloads can vary greatly and can impact the resource requirements, and these experiments show that the techniques presented in this work will perform well for a variety of workload types.

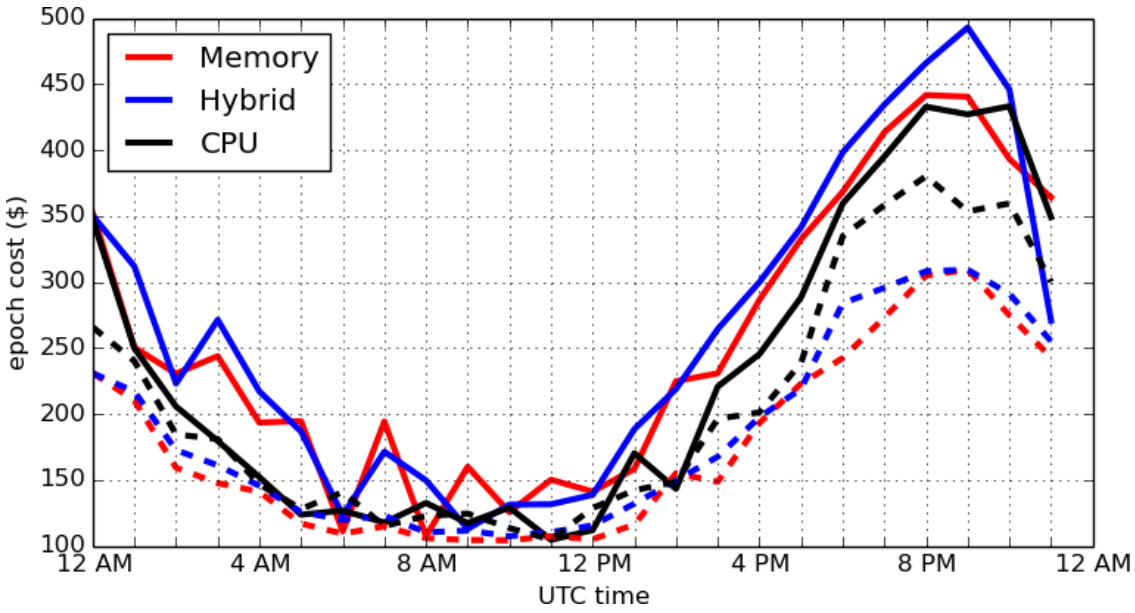

FIGURE 4.7. System costs across twenty four epoch period for different workload types, for a group of four locations. FDLD-CL shown as solid line, GALD-CL shown as dashed line.

4.6.3. TIME-OF-YEAR ANALYSIS. The peak renewable energy generating capacity is fixed for each data center, but the amount of power actually generated at a given time of day will vary depending on the month. All prior experiments use solar and wind information for the month of June. Given the variability of renewable power throughout the year (e.g. solar power in the winter versus summer months), it is important to understand how this would affect geographical load distribution algorithms that incorporate renewable power. Fig. 4.8 compares the GALD-CL and

FDLD-CL methods for the months of June and December for a group of four data centers. The total system cost is higher in December due to reduced available renewable power, but the GDRM is still able to minimize total system cost with either heuristic.

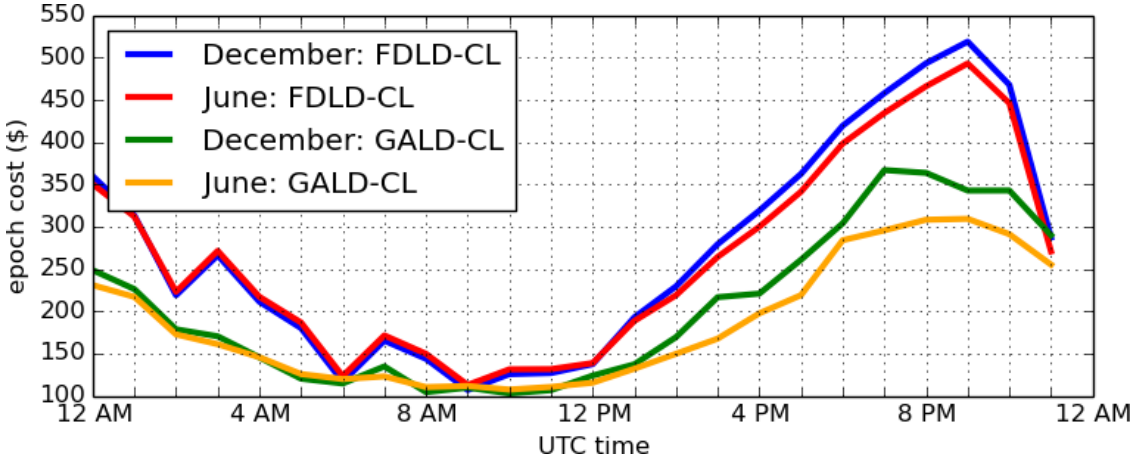

FIGURE 4.8. Comparison of performance during different months, four locations.

4.6.4. SCALABILITY ANALYSIS. Performing geographical load distribution for groups of data centers with more than four locations is not only a more complex problem, but also allows for more opportunities to take advantage of varying renewable power and TOU pricing. Additional experiments were conducted using groups of eight and sixteen separate data centers. For each of the data center group sizes, the average performance improvement of each technique over the FDLD-SO method is given in Table 4.3. These experiments confirm that all approaches can still perform well even for larger problem sizes. It should be noted that as the number of data centers in the group grows larger, the time for the FDLD variants to reach a solution increases, and the number of GALD-CL generations that can take place within the time limit decreases. As previously mentioned, the increase in the runtime of the FDLD heuristics was very manageable as the number of data centers in the group increased.

TABLE 4.3. Electricity cost reduction compared to FDLD-SO

Heuristic	4 datacenters	8 datacenters	16 datacenters
FDLD-TAO	5.2%	4.6%	5.7%
FDLD-CL	14.2%	15.7%	18.8%
GALD-CL	39.7%	39.2%	36.8%

4.7. CONCLUSION

We propose four workload allocation heuristics for workload allocation across geographically distributed data centers. In this work, we explored adding different levels of knowledge of the system, particularly co-location interference, to geographical workload distribution algorithms. We demonstrated that including additional information in the decision process of the heuristics resulted in a lower energy cost by reducing or eliminating server over-provisioning while still meeting all required workload execution rates. Our FDLD-CL and GALD-CL heuristics resulted on average in 10% and 37%, respectively, lower total cost than the prior work (represented by the FDLD-SO heuristic).

To demonstrate the robustness of our approaches, we performed three additional investigations. First, we compared the performance of the heuristics for three different workload types; CPU-intensive, memory-intensive, and a hybrid combination of CPU-intensive and memory-intensive tasks, demonstrating the ability of the heuristics to perform well for workloads with a range of levels of performance degradation from co-location interference. Second, we tested the sensitivity of the heuristic to variants in renewable power by simulating different months of the year, and showed that heuristics still perform well. Lastly, we tested the scalability of our heuristics by simulating three different data center group sizes of four, eight and sixteen data centers.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1. CONCLUSION

This thesis proposes hierarchical framework for reducing energy consumption and electricity cost, and improving data center throughput efficiency. Each component considers a different level of the system, including inter-processor performance degradation modeling, energy optimization of a data center via power, thermal and co-location aware resource allocation, and cost-minimizing geo-distributed resource allocation. The three components can be used individually or together as complete unit to make more informed system management decisions.

The work of this thesis considers all system levels, each at an unprecedented level of detail, when making optimization decisions. To enhance this detail, real data was used throughout wherever possible, including server power consumption, task execution times for real tasks, co-location performance degradation effects for real tasks, hourly solar irradiance data, hourly wind data, and TOU pricing. This thorough use of real-world data enables more accurate simulations, and ensures the validity and soundness of the presented heuristics.

Rising data center electricity consumption, and as a result increased electricity costs, have led data center operators to seek ways of reducing power consumption by limiting server overprovisioning and more efficiently using active servers. Future efforts to improve efficiency will need to consider all system levels, and will need to support their work using real data. The hierarchical framework proposed within this thesis can serve as an effective template for future efforts seeking to minimize cost and improve energy efficiency.

5.2. FUTURE WORK

The study in Chapter 2 proposes methodologies to model and predict the increase in execution time of applications due to contention for memory resources when they are co-located on different cores of the same processor. Linear regression models and neural networks were designed using information collected from multiple internal performance counters. This data, while useful, gave only indirect insight into the performance degradation. Future efforts could explore other machine learning techniques, and potentially identify useful higher-level trends. Additionally, the tests of this study used only a few architectures, all of which are server-grade systems, and only a handful of benchmark applications were used. Testing on more architectures and with more benchmarks might yield trends previously missed.

The study in Chapter 3 attempts to minimize energy use in an over-subscribed data center by simultaneously considering thermal, power, server heterogeneity, and co-location interference during resource allocation. Several optimization techniques are presented, however all of them optimize for a steady-state scenario. In a real system it is often very difficult to accurately predict the arriving workload. Future extensions to this work could explore dynamic scheduling that still takes all of the aforementioned system features into account. The largest change that would be needed to implement a dynamic version of this work would be a way to model thermal effect in near-real-time that is computationally feasible.

The study in Chapter 4 explores electricity cost minimization via intelligent resource allocation for geo-distributed data centers. Time-of-use electricity pricing and renewable power are both leveraged to reduce cost. Periods of peak renewable power, however, often do not coincide with peak workload and peak electricity pricing, while peak workload and peak electricity pricing often do coincide. The addition of energy storage would add another dimension to the solution space, and allow either renewable energy (or even possibly grid-sourced energy during times of cheap

electricity and minimal renewable energy) to be used during periods of peak workload and peak electricity pricing. Additionally, delay shifting of workloads could be explored to reduce data center load during peak periods.

Chapter 4 could further be expanded by exploring more advanced models for the transmission and transfer of workload and data. In its current state, the study assumes that the workload originates from multiple locations off-site from the considered data centers. More advanced network models, as well as different workload scenarios. Additionally, the current study is constrained to the continental United States. A more advanced data center communication model could allow for simulations of scenarios spanning multiple countries or even multiple continents.

BIBLIOGRAPHY

[1] P. Gas and E. Company, "Electric schedule e-19," Pacific Gas and Electric Company, Tech. Rep., April 2015.

[2] M. A. Oxley, E. Jonardi, S. Pasricha, A. A. Maciejewski, G. A. Koenig, and H. J. Siegel, "Thermal, power, and co-location aware resource allocation in heterogeneous high performance computing systems," in *5th Int'l Green Comp. Conf. (IGCC '14)*, Nov. 2014, 10 pp.

[3] NREL, "National solar radiation database," https://mapsbeta.nrel.gov/nsrdb-viewer/, April 2015.

[4] J. Choi, M. Dukhan, X. Liu, and R. Vuduc, "Algorithmic time, energy, and power on candidate HPC compute building blocks." in *IEEE 28th International Parallel and Distributed Processing Symposium*, May 2014, pp. 447–457.

[5] D. Dauwe, R. Friese, S. Pasricha, A. A. Maciejewski, G. A. Koenig, and H. J. Siegel, "Modeling the effects on power and performance from memory interference of co-located applications in multicore systems," in *The 2014 International Conference on Parallel and Distributed Processing Techniques and Applications(PDPTA-2014)*, Jul. 2014, pp. 3–9.

[6] A. Sandberg, A. Sembrant, E. Hagersten, and D. Black-Schaffer, "Modeling performance variation due to cache sharing," in *IEEE 19th International Symposium on High Performance Computer Architecture (HPCA2013)*, May. 2013, pp. 155–166.

[7] L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M. L. Soffa, "The impact of memory subsystem resource sharing on datacenter applications," in *38th Annual International Symposium on Computer Architecture (ISCA'11)*, Jun. 2011, pp. 283–294.

[8] A. Merkel, J. Stoess, and F. Bellosa, "Resource-conscious scheduling for energy efficiency on multicore processors," in *5th European Conference on Computer systems*. ACM, 2010, pp. 153–166.

[9] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *IEEE/ACM International Symposium on Microarchitecture*, Dec. 2011, pp. 248–259.

[10] T. Dwyer, A. Fedorova, S. Blagodurov, M. Roth, F. Gaud, and J. Pei, "A practical method for estimating performance degradation on multicore processors, and its application to hpc workloads," in *International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012.

[11] M. De Vuyst, R. Kumar, and D. M. Tullsen, "Exploiting unbalanced thread scheduling for energy and performance on a cmp of smt processors," in *International Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006, pp. 10–20.

[12] S. Parekh, S. Eggers, H. Levy, and J. Lo, "Thread-sensitive scheduling for smt processors," in *Technical report, Department of Computer Science and Engineering, University of Washington*, 2000.

[13] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer New York, 2006, vol. 1.

[14] Canonical. (2012) Ubuntu 14 release notes. [Online]. Available: https://wiki.ubuntu.com/TrustyTahr/ReleaseNotes

[15] "Intel 64 and ia-32 architectures software developer's manual volume 3b: System programming guide, part 2," Tech. Rep., Feb. 2014. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-.manual.pdf

[16] (accessed Oct. 2014) Intel xeon e5-2697v2 processor. [Online]. Available: http://ark.intel.com/products/75283/

[17] (accessed Oct. 2014) Intel xeon e5649 processor. [Online]. Available: http://ark.intel.com/products/52581/

[18] (accessed Mar. 2014) Performance application programming interface. [Online]. Available: http://icl.cs.utk.edu/papi/

[19] (accessed Mar. 2014) HPCToolkit. [Online]. Available: http://hpctoolkit.org/

[20] (accessed Oct. 2014) PARSEC benchmark suite. [Online]. Available: http://parsec.cs.princeton.edu/

[21] (accessed Oct. 2014) NAS parallel benchmarks. [Online]. Available: http://www.nas.nasa.gov/publications/npb.html

[22] B. Efron and R. J. Tibshirani, *An Introduction to The Bootstrap*. CRC press, 1994.

[23] W. Feng, "The Green500 list - November 2013," Nov. 2013, accessed 24 Feb. 2014. [Online]. Available: http://www.green500.org/lists/green201311

[24] P. Kogge *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," DARPA, Tech. Rep., Sep. 2008, http://users.ece.gatech.edu/mrichard/ExascaleComputingStudyReports/exascale_final_report_100208.pdf. Accessed 5 Feb. 2014.

[25] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam, "Cuanta: Quantifying effects of shared on-chip resource interference for consolidated virtual machines," in *2nd ACM Symp. on Cloud Comp. (SOCC '11)*, Oct. 2011, pp. 1–14.

[26] A. M. Al-Qawasmeh, S. Pasricha, A. A. Maciejewski, and H. J. Siegel, "Power and thermal-aware workload allocation in heterogeneous data centers," *IEEE Trans. Comput.*, vol. 64, no. 2, pp. 477–491, Feb. 2015.

[27] J. Bruschi, P. Rumsey, R. Anliker, L. Chu, and S. Gregson, "Best practices guide for energy-efficient data center design," US Dept. of Energy, Tech. Rep., Mar. 2011, http://www1.eere.energy.gov/femp/pdfs/eedatacenterbestpractices.pdf. Accessed 10 Oct. 2013.

[28] H. Shamalizadeh, L. Almeida, S. Wan, P. Amaral, S. Fu, and S. Prabh, "Optimized thermal-aware workload distribution considering allocation constraints in data centers," in *IEEE Int'l Conf. on Cyber, Physical and Social Comp. (CPSCom '13)*, Aug. 2013, pp. 208–214.

[29] M. Islam, S. Ren, N. Pissinou, H. Mahmud, and A. Vasilakos, "Distributed resource management in data center with temperature constraint," in *4th Int'l Green Comp. Conf. (IGCC '13)*, June 2013, pp. 31–40.

[30] J. Patel, S. Guercio, A. Bruno, M. Jones, and T. Furlani, "Implementing green technologies and practices in a high performance computing center," in *4th Int'l Green Comp. Conf. (IGCC '13)*, June 2013, pp. 1–8.

[31] A. Banerjee, T. Mukherjee, G. Varsamopoulos, and S. K. S. Gupta, "Cooling-aware and thermal-aware workload placement for green hpc data centers," in *1st Int'l Green Comp. Conf. (IGCC '10)*, Aug. 2010, pp. 245–256.

[32] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *44th Int'l Symp. on Microarchitecture (MICRO '11)*, Dec. 2011, pp. 248–259.

[33] M. Caccamo, R. Pellizzoni, L. Sha, G. Yao, and H. Yun, "MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," in *19th Real-Time and Embedded Technology and Applications Symp. (RTAS '13)*, Apr. 2013, pp. 55–64.

[34] Hewlett-Packard, Intel, Microsoft, Phoenix Technologies, and Toshiba, *Advanced Configuration and Power Interface Specification*, 2011, rev. 5.0, http://www.acpi.info. Accessed 28 Aug. 2012.

[35] M. A. Iverson, F. Ozgüner, and L. Potter, "Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment," *IEEE Trans. Comput.*, vol. 48, no. 12, pp. 1374–1379, Dec. 1999.

[36] Y. A. Li, J. K. Antonio, H. J. Siegel, M. Tan, and D. W. Watson, "Determining the execution time distribution for a data parallel program in a heterogeneous computing environment," *J. Parallel and Distributed Comp.*, vol. 44, no. 1, pp. 33–52, July 1997.

[37] V. Shestak, J. Smith, A. A. Maciejewski, and H. J. Siegel, "Stochastic robustness metric and its use for static resource allocations," *J. Parallel and Distributed Comp.*, vol. 68, no. 8, pp. 1157–1173, Aug. 2008.

[38] J. Moore, J. Chase, P. Ranganathan, and R. Sharma, "Making scheduling "cool": Temperature-aware workload placement in data centers," in *USENIX Annual Technical Conf. (ATEC '05)*, Apr. 2005, pp. 61–75.

[39] Q. Tang, T. Mukherjee, S. K. Gupta, and P. Cayton, "Sensor-based fast thermal evaluation model for energy efficient high-performance datacenters," in *4th Int'l Conf. on Intelligent Sensing and Information Processing (ICISIP '06)*, Dec. 2006, pp. 203–208.

[40] D. Dauwe, R. Friese, S. Pasricha, A. A. Maciejewski, G. A. Koenig, and H. J. Siegel, "Modeling the effects on power and performance from memory interference of co-located applications in multicore systems," 2014, 7 pp., *submitted.*

[41] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, R. F. Freund, D. Hensgen, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *J. Parallel and Distributed Comp.*, vol. 61, no. 6, pp. 810–837, June 2001.

[42] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *J. Parallel and Distributed Comp.*, vol. 59, no. 2, pp. 107–131, Nov. 1999.

[43] D. Whitley, "The GENITOR algorithm and selective pressure: Why rank-based allocation of reproductive trials is best," in *3rd Int'l Conf. on Genetic Algorithms*, June 1989, pp. 116–121.

[44] Standard Performance Evaluation Corporation (SPEC), 2008, SPECpower_ssj2008, http://www.spec.org/power_ssj2008. Accessed 28 Dec. 2013.

[45] ——, 2006, SPEC_CPU2006, http://www.spec.org/cpu2006. Accessed 2 Jan. 2014.

[46] S.-G. Kim, C. Choi, H. Eom, H. Y. Yeom, and H. Byun, "Energy-centric DVFS controlling method for multi-core platforms," in *5th Int'l Workshop on Multi-Core Comp. Systems (MuCoCoS '12)*, June 2012, pp. 685–690.

[47] A. M. Al-Qawasmeh, A. A. Maciejewski, R. G. Roberts, and H. J. Siegel, "Characterizing task-machine affinity in heterogeneous computing environments," in *20th Heterogeneity in Comp. Workshop (HCW '11)*, May 2011, pp. 33–43.

[48] "Data center locations," http://www.google.com/about/datacenters/inside/locations/index.html.

[49] "Global infrastructure," http://aws.amazon.com/about-aws/global-infrastructure/.

[50] "AWS and sustainable energy," http://aws.amazon.com/about-aws/sustainable-energy/.

[51] C. Hsu, "Rack PDU for green data centers," in *Data Center Handbook*, H. Geng, Ed. John Wiley and Sons, Inc., Hoboken, NJ, 2014, ch. 29.

[52] L. Gu, D. Zeng, S. Guo, and B. Ye, "Joint optimization of VM placement and request distribution for electricity cost cut in geo-distributed data centers," in *4th Int'l Conf. Comput., Networking, and Comm. (ICNC '15)*, Feb. 2015, pp. 717–721.

[53] J. Zhao, H. Li, C. Wu, Z. Li, Z. Zhang, and F. C. M. Lau, "Dynamic pricing and profit maximization for the cloud with geo-distributed data centers," in *33rd Int'l Conf. Comp. Comm. (INFOCOM '14)*, Apr. 2014, pp. 118–126.

[54] L. Gu, D. Zeng, A. Barnawi, S. Guo, and I. Stojmenovic, "Optimal task placement with QoS constraints in geo-distributed data centers using DVFS," *IEEE Trans. Comp.*, online preprint Apr. 2014, accepted 2014, online:http://doi.ieeecomputersociety.org/10.1109/TC.2014.2349510.

[55] H. Xu, C. Feng, and B. Li, "Temperature aware workload management in geo-distributed data centers," *IEEE Trans. Parallel and Distributed Systems*, online preprint May 2014, accepted 2014, online:http://dx.doi.org/10.1109/TPDS.2014.2325836.

[56] M. Polverini, A. Cianfrani, S. Ren, and A. V. Vasilakos, "Thermal-aware scheduling of batch jobs in geographically distributed data centers," *IEEE Trans. Cloud Comp.*, vol. 2, no. 1, pp. 71–84, Apr. 2014.

[57] H. Goudarzi and M. Pedram, "Geographical load balancing for online service applications in distributed datacenters," in *6th Int'l Conf. on Cloud Comp. (CLOUD '13)*, June 2013, pp. 351–358.

[58] D. Mehta, B. O'Sullivan, and H. Simonis, "Energy cost management for geographically distributed data centres under time-variable demands and energy prices," in *6th Int'l Conf. on Utility and Cloud Computing (UCC '13)*, 2013, pp. 26–33.

[59] X. Deng, D. Wu, J. Shen, and J. He, "Eco-aware online power management and load scheduling for green cloud datacenters," *Systems Journal, IEEE*, no. 99, pp. 1–10, 2014.

[60] C. Ren, D. Wang, B. Urgaonkar, and A. Sivasubramaniam, "Carbon-aware energy capacity planning for datacenters," in *20th Int'l Symp. Modeling, Analysis Simulation of Comput. and Telecomm. Systems (MASCOTS '12)*, Aug 2012, pp. 391–400.

[61] C. Stewart and K. Shen, *Some Joules Are More Precious Than Others: Managing Renewable Energy in the Datacenter*, 2009.

[62] H. Bhagwat, U. Singh, A. Deodhar, A. Singh, A. Vasan, and A. Sivasubramaniam, "Fast and accurate evaluation of cooling in data centers," *J. Electronic Packaging*, vol. 137, no. 1, Mar. 2015, 9 pp.

[63] D. Dauwe, E. Jonardi, R. Friese, S. Pasricha, A. A. Maciejewski, D. A. Bader, and H. J. Siegel, "A methodology for co-location aware application performance modeling in multi-core computing," in *17th Workshop on Advances on Parallel and Distributed Computational Models (APDCM '15)*, May 2015, accepted, to appear.

[64] P. Paulin and J. Knight, "Force-directed scheduling for the behavioral synthesis of asics," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 8, no. 6, pp. 661–679, Jun 1989.

[65] H. Goudarzi and M. Pedram, "Geographical load balancing for online service applications in distributed datacenters," in *6th Int'l Conf. Cloud Comp. (CLOUD '13)*, June 2013, pp. 351–358.

[66] C. Isci, S. McIntosh, J. Kephart, R. Das, J. Hanson, S. Piper, R. Wolford, T. Brey, R. Kantner, A. Ng, J. Norris, A. Traore, and M. Frissora, "Agile, efficient virtualization power management with low-latency server power states," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 96–107.