

DISSERTATION

ENERGY- AND THERMAL-AWARE RESOURCE MANAGEMENT FOR
HETEROGENEOUS HIGH-PERFORMANCE COMPUTING SYSTEMS

Submitted by

Mark Oxley

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2016

Doctoral Committee:

Advisor: H. J. Siegel

Co-Advisor: Sudeep Pasricha

Anthony A. Maciejewski

Darrell Whitley

Copyright by Mark Oxley 2016

All Rights Reserved

ABSTRACT

ENERGY- AND THERMAL-AWARE RESOURCE MANAGEMENT FOR HETEROGENEOUS HIGH-PERFORMANCE COMPUTING SYSTEMS

Today’s high-performance computing (HPC) systems face the issue of balancing electricity (energy) use and performance. Rising energy costs are forcing system operators to either operate within an energy budget or to reduce energy use as much as possible while still maintaining performance-based service agreements. Energy-aware resource management is one method for solving such problems. Resource management in the context of high-performance computing refers to the process of assigning and scheduling workloads to resources (e.g., compute nodes). Because the cooling systems in HPC facilities also consume a considerable amount of energy, it is important to consider the computer room air conditioning (CRAC) units as a controllable resource and to study the relationship (and energy consumption impact) between the computing and cooling systems. In this thesis, we present four primary contributing studies with differing environments and novel techniques designed for each of those environments. Each study proposes new ideas in the field of energy- and thermal-aware resource management for heterogeneous HPC systems.

Our first contribution explores the problem of assigning a collection of independent tasks (“bag-of-tasks”) to a heterogeneous HPC system in an energy-aware manner, where task execution times vary. We propose two new measures that consider these uncertainties with respect to makespan and energy: makespan-robustness and energy-robustness. We design resource management heuristics to either: (a) maximize makespan-robustness within an energy-robustness constraint, or (b) maximize energy-robustness within a makespan-robustness constraint.

Our next contribution studies a rate-based environment where task execution rates are assigned to compute cores within the HPC facility. The performance measure in this study is the reward rate earned for executing tasks. We analyze the impact that co-location interference (i.e., the performance degradation experienced when tasks are simultaneously executing on cores that share memory resources) has on the reward rate. Novel heuristics are designed that maximize the reward rate under power and thermal constraints, considering the interactions between both computing and cooling systems.

As part of the third contribution, we design new techniques for a geographical load distribution problem. That is, our proposed techniques intelligently distribute the workload to data centers located in different geographical regions that have varying energy prices and amount of renewable energy available. The novel techniques we propose use knowledge of co-location interference, thermal models, varying energy prices, and available renewable energy at each data center to minimize monetary energy costs while ensuring all tasks in the workload are completed.

Our final contribution is a new energy- and thermal-aware runtime framework designed to maximize reward earned from completing individual tasks by their deadlines within energy and thermal constraints. Thermal-aware resource management strategies often consult thermal models to intelligently determine which cores in the HPC facility to assign workloads. However, the time required to perform the thermal model calculations can be prohibitive in a runtime environment. Therefore, we propose a novel offline-assisted online resource management technique where the online resource manager uses information obtained from offline-generated solutions to help in its thermal-aware decision making.

ACKNOWLEDGEMENTS

I thank my advisors Professors Siegel, Pasricha, and Maciejewski for their valuable insight and counsel for the duration of my Ph.D. studies. Their guidance and support has been extremely generous, useful, and insightful over my Ph.D tenure. I would like to also thank another committee member, Dr. Darrell Whitley, for his input and feedback. A big thanks to all of the members of our research group that have also provided valuable feedback on research ideas, papers, and presentations: Dr. Abdulla Al-Qawasmeh, Dr. Mohsen Amini, Jonathan Apodaca, Daniel Dauwe, Dr. Ryan Friese, Dr. Tim Hansen, Eric Jonardi, Dr. Bhavesh Khemka, Dr. Greg Pfister, Dr. Jerry Potter, Dr. Jay Smith, Dr. Kyle Tarplee, and Dalton Young.

A special thanks to my family and fiancée for their considered support through the trials and tribulations that come with being a Ph.D student. My parents, Dotty and Jay Oxley, have provided kind words to keep me motivated. My brother Kevin Oxley and his family (Chris, Katie, Nikki, and Kaliyah) have provided entertainment and happiness. Last, a very special thank you to Amber Riester. She has put up with me through times of stress, anxiety, and frustrations, but has kept me motivated and calm.

Parts of the research in this dissertation were supported by the National Science Foundation under grant numbers CNS-0615170, CNS-0905339, CCF-1302693, and CCF-1252500, and by the Colorado State University George T. Abell Endowment. Parts of this research used the CSU ISTeC HPC System supported by National Science Foundation under grant number CNS-0923386. Parts of this research used systems donated by Hewlett Packard.

DEDICATION

To my amazing family and partner Amber, whose compassion and patience are without equal.

TABLE OF CONTENTS

Abstract	ii
Acknowledgements	iv
Dedication	v
List of Tables	ix
List of Figures	xi
Chapter 1. Introduction and Overview	1
Chapter 2. Energy-Aware Robust Resource Allocation*	7
2.1. Introduction	7
2.2. Related Work	10
2.3. System Model	11
2.4. Stochastic Measures	14
2.5. Heuristics	19
2.6. Constrained Optimization	28
2.7. Results	31
2.8. Conclusions	41
Chapter 3. Rate-based Thermal, Power, and Co-location Aware Resource Management for Heterogeneous High Performance Computing Systems [†]	43
3.1. Introduction	43
3.2. Related Work	47
3.3. System Model	51
3.4. Heuristics	58

3.5.	Evaluation Setup	63
3.6.	Results	66
3.7.	Conclusions	75
Chapter 4. Energy Cost Optimization for Geographically Distributed Heterogeneous		
	Data Centers [†]	77
4.1.	Introduction	77
4.2.	Related Work	78
4.3.	System Model	80
4.4.	Heuristic Descriptions	86
4.5.	Simulation Results	92
4.6.	Conclusion	96
Chapter 5. Online Resource Management in Thermal and Energy Constrained		
	Heterogeneous High Performance Computing Systems [§]	97
5.1.	Introduction	97
5.2.	System Model	100
5.3.	Problem Statement	106
5.4.	Proposed Resource Manager	106
5.5.	Comparison Techniques	116
5.6.	Simulation Setup	118
5.7.	Results	120
5.8.	Related Work	126
5.9.	Conclusions	129
Chapter 6. Summary of Thesis		
		130

Chapter 7. Future Work	133
Bibliography	137
Appendix A. Power and Performance Data Collection and Calculations For Node Types from SPECPower	151
Appendix B. Global and Local Search in Tabu Search for Robust Energy-aware Resource Management	153
Appendix C. Limiting the Search Space Technique for Robust Energy-aware Resource Management	155
Appendix D. Table of Notations for Robust Energy-aware Resource Management	157
Appendix E. NLP Formulation for Rate-Based Resource Management	158
Appendix F. Simulation Parameters for Rate-Based Resource Management	160

LIST OF TABLES

3.1	Description of features for predicting execution time	56
3.2	Node-Types Used In Simulations.....	64
4.1	Node Processor Types Used in Experiments.....	94
4.2	Monetary cost reduction compared to FDL-D-SO [1]	95
5.1	Node types used in simulations	118
A.1	Data From SPECpower.....	151
D.1	Table of Notations	157
F.1	Idle power consumption (watts) of each node type	161
F.2	$u(m, k)$ Coefficients	161
F.3	$v(m, k)$ Coefficients	161
F.4	$w(m, k)$ Coefficients.....	161
F.5	$x(m, k)$ Coefficients	161
F.6	$y(m, k)$ Coefficients	161
F.7	$z(m, k)$ Coefficients	161
F.8	Execution times (seconds) of PARSEC benchmarks on our lab servers at highest frequency P-state	162
F.9	Execution times (seconds) of PARSEC benchmarks on our lab servers at lowest frequency P-state	162
F.10	Power consumption (watts) of a core when executing PARSEC benchmarks on our lab servers at highest frequency P-state	162

F.11 Power consumption (watts) of a core when executing PARSEC benchmarks on our lab servers at lowest frequency P-state	162
--	-----

LIST OF FIGURES

1.1	(a) The top three supercomputers from the Green500 List [2], and (b) the top three supercomputers from the Top500 List [3].	2
1.2	HPC facility cooled by CRAC system.	4
2.1	Comparison of constrained optimization techniques for MO-EC with Tabu Search, GALS, and GA (25 node system, 458 total cores, and 10,000 tasks). The system deadline was set to 15,500 seconds, the energy budget was set to 58 MJ, and the energy-robustness constraint was set to 90%.	33
2.2	Varying the energy budget for MO-EC: (a) makespan-robustness, and (b) energy-robustness (25 node system, 458 total cores, and 10,000 tasks). The system deadline was set to 15,500 seconds, the energy-robustness constraint was set to 90%. The heuristics were terminated after six hours of execution time.	36
2.3	Varying the system deadline for EO-MC: (a) energy-robustness, and (b) makespan-robustness (25 node system, 458 total cores, and 10,000 tasks). The energy budget was set to 58 MJ and the makespan-robustness constraint was set to 90%. The heuristics were terminated after six hours of execution time.	37
2.4	Comparison of Tabu Search, GA, and GALS heuristics over 72 hours of execution time for MO-EC using the large simulation size (250 nodes, 4,580 total cores, and 100,000 tasks). The system deadline was set to 13,500 seconds, the energy budget was set to 580 MJ, and energy-robustness constraint was set to 90%.	38
2.5	Progress of Tabu Search, GALS, and GA for MO-EC over 72 hours of heuristic execution time for the (a) small simulation size (25 nodes, 458 total cores, 10,000 tasks), and (b) large simulation size (250 nodes, 4,580 total cores, and 100,000	

tasks). For the small simulation size, the system deadline was set to 15,500 seconds, the energy budget was set to 58 MJ, and the energy-robustness constraint was set to 90%. For the large simulation size, the system deadline was set to 13,500 seconds, the energy budget was set to 580 MJ, and energy-robustness constraint was set to 90%.....	39
2.6 Comparison of Tabu Search, GALS, and GA across heterogeneous environments with varying TMA for MO-EC using the small simulation size (25 nodes, 458 total cores, and 10,000 tasks). The system deadline was set to 15,500 seconds, the energy budget was set to 58 MJ, and the energy-robustness constraint was set to 90%.....	40
3.1 HPC facility in hot aisle/cold aisle configuration.....	52
3.2 (a) Small HPC platform configuration, and (b) large HPC platform configuration.	64
3.3 (a) Reward rate comparison between different workload environments on the 1,080 nodes system, and (b) power consumption comparison in relation to power budget constraint (red-line). Solid bars represent the GA, NLP, or greedy technique when using <i>RR</i> as the objective (co-location aware), and the hashed bars represent those techniques when using <i>NERR</i> as the objective (co-location unaware).....	66
3.4 (a) Reward rate comparison between isolated and non-isolated cold aisle configurations on the 1,080 nodes system, and (b) power consumption comparison in relation to power budget constraint (red-line). Hashed portion of bars show power consumed by the CRAC unit, and solid portion of bars show the power consumed by the compute nodes.	67

3.5	Sensitivity analysis of our greedy, GA, and NLP resource management techniques for both isolated and non-isolated HPC facility configurations using the 1,080 node system on the thermal (red-line) constraint.....	71
3.6	Sensitivity analysis of our greedy, GA, and NLP resource management techniques for both isolated and non-isolated HPC facility configurations using the 1,080 node system on the power constraint.....	72
3.7	Comparison of reward rate on the (a) small (1,080 node) platform, and (b) large (4,320 node) platform. NLP excluded in (b) because it was unable to finish within two weeks.....	74
4.1	Data center in hot aisle/cold aisle configuration [4].....	83
4.2	System costs across twenty four epoch period for each heuristic, four locations....	94
4.3	System costs across twenty four epoch period for different workload types, for a group of four locations. FDL-D-CL shown as solid line, and GALD-CL shown as dashed line.....	95
5.1	Data center aisle configuration.....	101
5.2	Block diagram of the resource management framework. Note that the specific data center shown is for illustrative purposes only, and our framework can be applied to any data center configuration.....	107
5.3	Workload arrival patterns with (a) constant, (b) bursty, and (c) sinusoidal patterns.....	119
5.4	Compares reward earned by techniques for the (a) constant, (c) bursty, and (e) sinusoidal workload arrival patterns. Similarly, (b), (d), and (f) display energy	


	consumed by those workload arrival patterns. All results in this figure are shown for the four runtime mapping techniques (consolidation, LBBN, LBBR, and greedy), and three thermal management strategies (overcooling, throttling, templates).....	122
5.5	Using the <i>bursty</i> workload arrival pattern, we compare total power consumption of the three thermal management techniques when using the <i>greedy</i> runtime mapping heuristic over the course of a day. The energy budget was set to 12,000 MJ.....	126
B.1	Comparison of short-hop termination criteria for MO-EC using the small simulation size (25 nodes, 458 total cores, and 10,000 tasks) over six hours of heuristic execution time. The system deadline was set to 15,500 seconds, the energy budget was set to 58 MJ, and energy-robustness constraint was set to 90%. The red numbers indicate number of long-hops performed.	154

CHAPTER 1

INTRODUCTION AND OVERVIEW


Increasing the processing speed of high-performance computing (HPC) systems offers enormous benefits to many fields, such as providing the means to rapidly comprehend complex biological processes through protein-folding, decreasing the amount of time necessary to predict weather patterns and earthquakes through computationally-intensive simulations, and providing better or faster optimization of manufacturing processes. But enhancing the performance of such HPC systems is becoming increasingly difficult as the power and energy consumption of these systems increases with the performance. Energy consumption has been identified as a key factor limiting growth of HPC systems [5] as large-scale computing systems require an increasing amount of electricity to power a large number of compute cores and also power the cooling infrastructure required to maintain safe operation. The increased energy consumption leads to increased total costs of ownership to operate such facilities.

One method that HPC systems incorporate to increase the energy efficiency of their systems is the use of heterogeneous hardware. By incorporating general-purpose graphics processing units (GPGPUs) and having a mix of energy efficient and high-performance processing units, heterogeneous HPC systems are able to provide a trade-off between low power or high-performance operation, and thus able to obtain more performance per watt when high-performance is not required. In fact, the Green500 list (see Figure 1.1 (a)) [2] that ranks the most energy efficient HPC systems has heterogeneous systems filling out the top ten energy efficient machines. In addition, the push to exascale [6] is largely prohibited by electricity consumption. For example, the highest-performing supercomputer according to the Top500 list (see Figure 1.1 (b)) is the Tianhe-2 system that at 33.86 petaFLOPS has



name	speed (petaFLOPS)	country	power (kW)	efficiency (GFLOPS/W)
Shoubu	0.353	Japan	50.32	7.03
TSUBAME-KFC	0.272	Japan	51.13	5.33
ASUS ECS4000	0.206	Germany	57.15	5.27

(a)



name	speed (petaFLOPS)	country	power (kW)	efficiency (GFLOPS/W)
Tianhe-2	33.86	China	17,808	1.90
Titan	17.59	USA	8,209	2.14
Sequoia	17.17	USA	7,890	2.18

(b)

FIGURE 1.1. (a) The top three supercomputers from the Green500 List [2], and (b) the top three supercomputers from the Top500 List [3].

a peak power consumption of 17,808 kW, which would cost approximately \$17 million per year in electricity using the average cost of electricity in commercial sectors in the U.S. [3, 7]. Extrapolating this system to exascale results in an energy cost of \$500 million per year, or approximately \$1.37 million per day for one HPC system. The high cost of energy for HPC systems today (and projected to be higher in the future), combined with the prohibitive cost of energy for an exascale system, makes the energy-aware management of HPC systems of paramount importance. The goal of the research presented in this thesis is to design intelligent and robust resource management techniques for heterogeneous HPC systems that are performance, energy, power, and thermal-aware. Resource management in the context of high-performance computing refers to the process of assigning and scheduling workloads to resources (e.g., compute nodes). This research contributes novel ideas to the field of heterogeneous HPC systems to help fulfill the needs of future systems.

Chapter 2 considers a heterogeneous HPC system in which the resource manager is given a bag-of-tasks to allocate to processing cores in a robust and energy-efficient manner. We assume cores have dynamic frequency and voltage scaling (DVFS) enabled, allowing the cores

to operate in discrete performance states (P-states). Employing DVFS provides a trade-off between task execution times and power consumption. Task execution times can vary from available estimates, and are thus modeled stochastically as random variables. In addition to being energy efficient, we want our resource management techniques to be robust against these variations. We develop probabilistic measures for performance and energy consumption, which we denote as *makespan-robustness* and *energy-robustness*. Makespan-robustness is defined as the probability of meeting a makespan deadline, and energy-robustness is defined as the probability of meeting an energy budget. We design, analyze, and compare four new heuristics that address two problems: (a) maximizing the makespan-robustness subject to an energy-robustness constraint, and (b) maximizing the energy-robustness subject to a makespan-robustness constraint. We evaluate our techniques through simulated experiments on two different-sized HPC platforms that vary in number of machines and tasks, and perform sensitivity analysis of our techniques against the degree of heterogeneity in the system. We propose new variations of a Tabu Search heuristic and a genetic algorithm and results demonstrate the significance of the novel local search operators that are used in our proposed techniques.

The cooling systems required to operate data centers at safe levels account for a significant portion of the total power consumed by such facilities. Chapter 3 takes a holistic approach to power-aware resource management by considering the thermal relationships between compute nodes and computer room air conditioning (CRAC) units. A depiction of an HPC facility cooled by a CRAC unit is shown in Figure 1.2. We study a rate-based environment where task execution rates are assigned to compute cores within the HPC facility. We incorporate the effects of co-location interference caused by cores competing for shared memory into our model. We design novel heuristics to maximize the reward earned

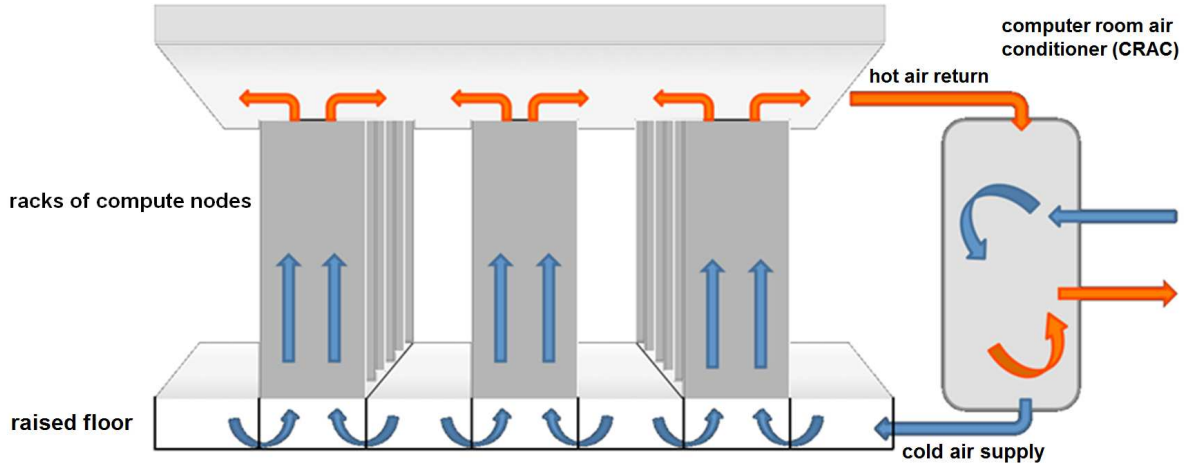


FIGURE 1.2. HPC facility cooled by CRAC system.

for completing tasks by their deadline, while also ensuring the system remains within imposed power and thermal constraints. These heuristics include a non-linear programming (NLP) technique that considers co-location interference, a greedy technique that assigns task types to their most power-efficient node type and P-state, and a genetic algorithm enhanced with local search. We perform in-depth analyses of our techniques when (a) they consider co-location interference versus when they are naïve to these effects, (b) different HPC facility system sizes are considered, and (c) the cold-aisles in the HPC facility are isolated versus non-isolated. We also perform sensitivity analyses of our techniques under a range of values for the power and thermal constraints, as well as workload environments.

The proliferation of geographically distributed data centers has motivated researchers to study energy cost minimization at the geo-distributed level. Chapter 4 examines the geographical load distribution problem using a rate-based environment, and proposes three novel heuristics that minimize energy cost at the geodistributed level while ensuring all tasks complete (i.e., the service rate of the system exceeds the arrival rate of tasks). The three primary benefits of intelligently distributing the workload among data centers are: (a) lower

latency for clients due to shorter communication distances, (b) workloads can be shifted to locations in different time zones to concentrate workload in regions with the lowest electricity prices at that time to exploit time-of-use pricing, and (c) an opportunistic distribution of the workload of data centers during periods of peak demand can allow individual nodes to run in slower but possibly more energy-efficient P-states or exploit on-site renewable energy sources, further reducing electricity costs. We use detailed models of power, temperature, and co-location interference at each data center to provide more accurate information to the geo-distributed resource manager. We demonstrate that our best heuristic for the geo-distributed resource manager can, on average, achieve a cost reduction of 37% compared to the state-of-the-art prior work.

In Chapter 5, we study runtime thermal and energy aware resource management instead of rate-based (offline), as was studied in Chapters 3 and 4. Incorporating intelligent decisions into proactive thermal-aware resource management requires a thermal model to predict the thermal implications of allocating tasks to different cores around the facility. Smart decisions regarding *where* to place incoming tasks can offer the benefit of operating the HPC facility at a hotter CRAC temperature (and therefore consuming less cooling energy) by avoiding hotspots. However, using a thermal model to predict temperatures can be a time-consuming process that requires complicated air flow patterns to be calculated for every mapping decision. Therefore, we propose a novel offline-assisted runtime resource management framework in which offline analysis is used to predict the thermal implications of mapping a given workload, and provide the runtime manager useful information regarding CRAC temperature settings, a set of cores to which the runtime resource manager is allowed to map tasks, and which floor vents to open or close to better direct cold air. The goal of our

proposed technique is to maximize the reward of completing tasks by their individual deadlines throughout the day while adhering to a daily energy budget and temperature threshold constraints.

Chapter 6 summarizes the research presented in this thesis. A discussion of possible future directions for the studies presented are in Chapter 7.

CHAPTER 2

ENERGY-AWARE ROBUST RESOURCE ALLOCATION*

2.1. INTRODUCTION

A recent study [10] estimates that the electricity used by data centers has increased by 56% worldwide between the years 2005 and 2010. With the electricity demands increasing and the energy costs of operating a data center surpassing 20% of the total costs [11], it has become common practice to either operate within an electricity budget or to reduce electricity use while maintaining service guarantees. The need for energy-efficient data centers is becoming more apparent as both power consumption and operating costs continue to rise. Energy-aware resource management techniques that can improve energy efficiency are therefore becoming increasingly important.

A commonly used technique to manage the energy efficiency in computing systems is to employ dynamic voltage and frequency scaling (*DVFS*) in server processors [12]. DVFS allows the cores in a processor to operate in discrete performance states (*P-states*), with lower-numbered P-states consuming more power but reducing the execution time of tasks. Because server processors use a large portion of the energy in a data center, we can employ DVFS to provide a trade-off between execution time and energy consumption.

*The full list of co-authors is at [8]. The other co-authors of this work are: Sudeep Pasricha, Anthony A. Maciejewski, H. J. Siegel, Jonathan Apodaca, Dalton Young, Luis Briceno, Jay Smith, Shirish Bahirat, Bhavesh Khemka, Adrian Ramirez, and Yong Zou. A preliminary version of portions of this work appeared in [9]. This research was supported by the NSF under grant numbers CNS-0615170, CNS-0905399, and CCF-1302693, and by the Colorado State University George T. Abell Endowment. This research used the CSU ISTeC Cray System supported by NSF Grant CNS-0923386.

Many large-scale computing facilities (e.g., data centers) incorporate heterogeneous resources that utilize a mix of different machines to execute workloads with diverse computational requirements. The execution times of tasks on heterogeneous machines are typically inconsistent such that if machine A is faster than machine B for a given task, machine A may not be faster for all tasks [13]. By assigning tasks to machines in an intelligent manner, it is possible to leverage machine heterogeneities to reduce task execution times and energy consumption.

The act of assigning tasks to machines is commonly referred to as resource allocation. Resource allocation decisions often rely on estimated values for task execution times whose actual values vary and may differ from available estimates (e.g., due to cache misses or data dependent execution times). These uncertainties in task execution times may cause a completion time (makespan) deadline or energy budget to be violated, therefore we want resource allocation techniques to be robust against these variations.

This research addresses the problem of *statically* allocating a workload of independent tasks to a heterogeneous computing cluster. Static mapping is used in several environments [13, 14], such as planning an efficient schedule for some set of jobs to be run at some time in the future. In the resource management literature, it is common to assume that information that characterizes the execution times of frequently executed tasks can be collected, e.g., [15–18]. We work closely with Oak Ridge National Labs, and in their environment, as well as others, similar types of tasks are executed frequently allowing for the collection of historical information about the execution times of tasks on machines. In this study, we assume knowledge of the means and variances of the execution times of each task on each machine, and can use this information to build probability distributions that approximate historical information. We want our resource allocations to be robust against the variations in the

task execution times, where we define *energy-robustness* as the probability that the energy budget is not violated, and *makespan-robustness* as the probability a makespan deadline is not violated. These probabilities are calculated using the means and variances of the task execution times.

By intelligently allocating resources and configuring DVFS, we utilize the heterogeneities in execution time and power to address two problems: (1) optimizing (maximizing) the makespan-robustness with a constraint on energy-robustness; and (2) optimizing (maximizing) the energy-robustness with a constraint on makespan-robustness. We refer to (1) as *MO-EC* (makespan-robustness optimization under an energy-robustness constraint) and (2) as *EO-MC* (energy-robustness optimization under a makespan-robustness constraint). This study focuses on the design and analysis of makespan- and energy-robust resource allocation heuristics for a heterogeneous computing cluster to address both the energy-constrained and deadline-constrained problems. We also analyze the impact of four different methods of constrained optimization used within the heuristics, and we demonstrate the flexibility and performance of the heuristics when the constraints are easy or difficult to meet. In summary, we make the following contributions:

- The design and analysis of resource management techniques for both optimizing makespan-robustness with an energy-robustness constraint (MO-EC) and optimizing energy-robustness with a makespan-robustness constraint (EO-MC).
- An enhanced power model that uses real system specifications for CPU voltage and frequency of DVFS P-states and overhead power of additional server components. Also a workload model where tasks have varying degrees of compute and memory intensity.

- An analysis on the effectiveness of our techniques on two different sized platforms that vary in both number of machines and tasks.
- A sensitivity analysis of our techniques against the level of heterogeneity.

The rest of this chapter is organized as follows. Section 2.2 discusses related work. The system model and workload are defined in Section 2.3. Section 2.4 describes the stochastic measures for makespan and energy consumption. The energy-aware resource allocation heuristics are proposed in Section 2.5, and the methods of constrained optimization used in conjunction with the heuristics are discussed in Section 2.6. Section 2.7 discusses the results, and finally we give the conclusions in Section 2.8.

2.2. RELATED WORK

Energy-aware resource allocation in high performance computing (HPC) systems is an important research area as high power consumption and associated energy costs are difficult obstacles. Therefore, numerous recent works have focused on energy-aware resource allocation techniques that exploit energy saving techniques such as DVFS to reduce the energy consumption of computing systems. To the best of our knowledge, our work is the first to address energy-aware resource allocation of a bag-of-tasks with uncertain task execution times to a heterogeneous computing system with goals of considering robustness of both makespan and energy consumption.

Energy-aware scheduling on heterogeneous platforms that considers deterministic task execution times has been previously studied (e.g., [19–21]). Li et al. [19] consider energy-aware scheduling of a collection of independent tasks on a heterogeneous platform that is DVFS-enabled. The primary contribution is the design of a resource allocation algorithm that minimizes energy consumed while ensuring the collection of tasks completes by a common deadline. Energy-aware scheduling of a bag-of-tasks application to a heterogeneous

computing system is considered in [20]. The goal in that paper is to allocate the bag of same-size tasks to the heterogeneous system to minimize energy consumed under a throughput constraint. Energy-aware scheduling on milliclusters is studied in [21]. Milliclusters are a collection of numerous low-power processing elements (e.g., those found in mobile devices) that are organized into a large cluster for scientific computing purposes. Three heuristics are designed to minimize energy consumption to complete a collection of tasks while adhering to each task’s individual deadline. Our work is novel and different from those listed because it considers uncertain task execution times (modeled as random variables) and the design of relevant performance measures to capture the robustness of both makespan and energy consumption against these uncertain task execution times.

Robust resource allocation of tasks with uncertain execution times has been studied, e.g., [22–25]. These works do not consider energy consumption, which is the focus of our work. Energy-aware resource allocation of tasks with uncertain execution times was studied in [26]. The goal for the resource allocation techniques in [26] is to finish as many tasks as possible by their individual deadlines without violating an energy constraint. The definition of robustness is the expected number of tasks that will complete by their individual deadlines. Our work considers complex resource management techniques designed for a static scheduling problem, probabilistic measures for both energy consumption and performance, and a more accurate power model that includes the overhead power of compute nodes and static power consumption of cores.

2.3. SYSTEM MODEL

2.3.1. COMPUTE NODES. The cluster we model consists of N heterogeneous compute nodes, and each node i contains \mathbf{n}_i homogeneous cores. Cores are the most basic processing element in this study, with each core processing one task at a time, e.g., as done in a Cray [27].

Cores are also DVFS-enabled to use P-states that allow a core to change operating voltage and frequency to provide a trade-off between the execution time and power consumption of the processor. Each core of compute node i has \mathbf{PS}_i P-states available. We assume each core in the system can operate in an individual P-state, and our resource allocation techniques are designed such that P-states do not switch during task execution. Lower-numbered P-states consume more power, but provide faster execution times, e.g., P-state 0 provides the shortest execution times but also consumes the most power.

2.3.2. POWER MODEL. The power consumption of a node includes the dynamic power of the cores, the static power consumption of the cores, and the base overhead power of the node (e.g., for disks, memory, network interface cards) [28]. We assume that when a core is finished processing its assigned workload, the core is able to deactivate and the power consumption (both dynamic and static) becomes negligible. Similarly, when all cores of a compute node are finished with their assigned workloads, the entire node is able to deactivate and consume no overhead power. For our static resource allocation problem with independent tasks, nodes do not have idle time because nodes are active when processing tasks and then deactivate when finished with their assigned workload, the ACPI S6 state [12].

For core j within node i operating in P-state π , we use the well-known equation for dynamic power ($P_{ij\pi}^{dyn}$) using the load capacitance (C_i), the supply voltage ($Vdd_{j\pi}$), and the frequency ($f_{j\pi}$) as [29]

$$(1) \quad P_{ij\pi}^{dyn} \propto C_i \cdot Vdd_{j\pi}^2 \cdot f_{j\pi}.$$

We consider the static power consumption of cores and the overhead power used by additional components (e.g., memory, disks, add-on cards). We assume the static power of

a core and the overhead power of a node are constant and independent of the voltage and frequency of the P-state that the core is currently in.

2.3.3. WORKLOAD. The workload consists of a collection of \mathbf{T} independent tasks to be completed before a given system deadline δ and within a given energy budget of Δ . Such a collection of independent tasks is known as a *bag-of-tasks*. We assume the tasks to be executed are known *a priori* such that we can perform a static (i.e., off-line) mapping. The actual values of task execution times vary (e.g., due to cache misses or data dependent execution times), and we model the execution times stochastically to account for these variations. That is, we are provided a mean and a variance that describe an execution time probability density function (*pdf*) for each task executing on each compute node in each P-state. In an actual system, the means and variances of these Gaussian execution time distributions can be approximated using historical, experimental, or analytical techniques [30–32]; in Section 2.7 we discuss our values for evaluation. The use of Gaussian distributions allows us to sum task execution times using a closed-form equation rather than having to perform convolution, however our proposed resource management techniques are applicable for task execution times that are described by any distribution.

It has been shown [33] that the arithmetic intensity (the number of operations performed per word of memory transferred) of a workload can greatly affect how the execution time of a task scales with the frequency of the CPU. The execution times of tasks with high arithmetic intensity generally scale proportionally with frequency, e.g., when the frequency is halved the execution time of the task doubles. Because the overhead power of a compute node remains constant regardless of P-state, the greater execution times resulting from low-power P-states can result in greater energy consumption than when executing the task as fast as possible in P-state 0. The most energy-efficient P-state depends on the ratio of compute energy saved

by operating in a low-power P-state to the amount of overhead energy consumed over the longer execution time that results from the decrease in frequency. With memory intensive workloads (i.e., workloads with low arithmetic intensity), a reduction in frequency has less impact on the execution times of tasks, because the processor spends a large portion of cycles waiting for data from memory. Memory intensive tasks offer greater potential for energy savings using DVFS, as power can be greatly reduced with little effect on execution time. We assume each task is one of a set of *task types* that is representative of the arithmetic intensity of the task. Our method for the scaling of execution time by task type and frequency is detailed in Appendix A.

2.4. STOCHASTIC MEASURES

2.4.1. OVERVIEW. The traditional performance measure for bag-of-tasks scheduling problems is *makespan*, the time required for all tasks within the bag to finish execution. In this study, we also are concerned with the energy consumption required to execute the bag-of-tasks, however, typically both makespan and energy are measures that rely on deterministic values for time. Because real environments have uncertainty, resource allocation decisions in such environments should be based on this stochastic information and be robust against the variations in the task execution times.

We define a “robust” resource allocation as one that can mitigate the impact of uncertainties on both performance and energy objectives. To claim robustness for a system, the following three questions must be answered [34]: (1) *What behavior makes the system robust?* In our system, a resource allocation is considered *makespan-robust* if the entire workload completes within the system deadline δ , and *energy-robust* if the workload can be completed within an energy budget of Δ . (2) *What uncertainties is the system robust against?* We want our system to be robust against the uncertain task execution times. (3)

How is the robustness of the system quantified? We quantify the makespan-robustness of a resource allocation as the probability that the entire workload is completed by δ , and the energy-robustness as the probability that the workload uses no more than Δ energy.

2.4.2. MAKESPAN-ROBUSTNESS. The execution time distribution for each task on each node is modeled as a Gaussian distribution with a given mean and variance. For a given resource allocation, let the set \mathbf{T}_{ij} denote tasks in T that have been assigned to core j in compute node i and let $t_{ij}^x \in \mathbf{T}_{ij}$ where $1 \leq x \leq |\mathbf{T}_{ij}|$. Let $\mathbf{PS}(t_{ij}^x)$ denote the assigned P-state for task t_{ij}^x . We denote the mean execution time associated with task t_{ij}^x executed in P-state π as $\mu(t_{ij}^x, \pi)$ and the associated variance as $V(t_{ij}^x, \pi)$.

The calculation of the completion time of core j when using a stochastic model for task execution times is performed by taking the convolution of the random variables (representing the execution times of tasks) for all tasks assigned to core j . The convolution of two independent normally distributed random variables α and β produces a normally distributed random variable with its mean being the sum of the means of α and β and the variance being the sum of the variances of α and β .

The expected finishing time of core j in compute node i , denoted F_{ij} , is the sum of the mean execution times of all tasks assigned that core and is given as

$$(2) \quad F_{ij} = \sum_{\forall t_{ij}^x \in \mathbf{T}_{ij}} \mu(t_{ij}^x, \mathbf{PS}(t_{ij}^x)).$$

The variance of the completion time distribution of core j , denoted σ_{ij}^2 , is the sum of the variances of the execution times of all tasks assigned to that core and is given as

$$(3) \quad \sigma_{ij}^2 = \sum_{\forall t_{ij}^x \in \mathbf{T}_{ij}} V(t_{ij}^x, \mathbf{PS}(t_{ij}^x)).$$

Thus, the completion time distribution of all tasks assigned to core j in compute node i can be expressed as the distribution $\mathcal{N}(F_{ij}, \sigma_{ij}^2)$. Given deadline δ , we can compute the probability that $\mathcal{N}(F_{ij}, \sigma_{ij}^2)$ is less than δ by converting $\mathcal{N}(F_{ij}, \sigma_{ij}^2)$ to its associated cumulative density function (*cdf*) and finding the probability associated with that core finishing before time δ (i.e., $\mathbb{P}(\mathcal{N}(F_{ij}, \sigma_{ij}^2) \leq \delta)$). We define the overall *makespan-robustness* of a resource allocation, denoted Ψ , as the minimum probability across all cores. That is, each core has a probability of at least Ψ that it will complete its assigned workload by deadline δ . We calculate Ψ as

$$(4) \quad \Psi = \min_{\forall i \in \mathcal{N}} (\min_{\forall j \in n_i} \mathbb{P}(\mathcal{N}(F_{ij}, \sigma_{ij}^2) \leq \delta))$$

We denote the makespan-robustness constraint as Γ , so for EO-MC we require resource allocations to meet this constraint (i.e., $\Psi \geq \Gamma$). When using a distribution other than the normal distribution for task execution times, the completion time for all tasks on a given core can be calculated by taking the convolution of all of the task execution time random variables assigned to that core. The makespan-robustness can then be calculated by converting the result into its associated cdf and then finding the probability associated with that core finishing before the deadline.

2.4.3. ENERGY-ROBUSTNESS. The energy required to process the workload is determined by summing the energy used by all tasks in the workload. In our model, the overhead power of nodes (\mathbf{O}_i), the dynamic power of cores ($\mathbf{P}_{ij\pi}^{dyn}$), and the static power of cores ($\mathbf{P}_{ij\pi}^{stat}$) all contribute to the total energy consumed. These power values are multiplied by the mean execution times of tasks to calculate the expected energy to process the workload, or by the variances of the execution times of tasks to calculate the variance in energy consumed to process the workload. Let \mathbf{T}_{ij}^π denote the subset of tasks assigned to core j of

compute node i processed in P-state π , i.e., $T_{ij}^\pi = \{\forall t_{ij}^x \in T_{ij} \mid PS(t_{ij}^x) = \pi\}$. The energy consumed is calculated as the product of execution time (a random variable) and average power (a deterministic value). The multiplication of a random variable with a scalar value has the effect of multiplying the expected value by that value and the variance by the square of that value [35]. The expected dynamic energy spent by core j in compute node i at P-state π , denoted $Mean_{ij\pi}^{dyn}$, is the sum of the mean values of dynamic energy consumption for all tasks assigned to that core and is given as

$$(5) \quad Mean_{ij\pi}^{dyn} = \sum_{t_{ij}^x \in T_{ij}^\pi} P_{ij\pi}^{dyn} \cdot \mu(t_{ij}^x, \pi).$$

Likewise, the variance of the dynamic energy spent by core j in compute node i in P-state π , denoted $Var_{ij\pi}^{dyn}$, is the sum of the variance values for dynamic energy consumption for all tasks assigned to that core and is given as

$$(6) \quad Var_{ij\pi}^{dyn} = \sum_{t_{ij}^x \in T_{ij}^\pi} (P_{ij\pi}^{dyn})^2 \cdot V(t_{ij}^x, \pi).$$

Similarly, the expected static energy consumed by core j on node i , denoted $Mean_{ij\pi}^{stat}$, is the sum of the mean values of static energy consumption for all tasks assigned to that core and is given as

$$(7) \quad Mean_{ij\pi}^{stat} = \sum_{t_{ij}^x \in T_{ij}^\pi} P_i^{stat} \cdot \mu(t_{ij}^x, \pi).$$

The variance of static energy, denoted $Var_{ij\pi}^{stat}$, is the sum of the variance values of static energy consumption for all tasks assigned to that core and is given as

$$(8) \quad Var_{ij\pi}^{stat} = \sum_{t_{ij}^x \in T_{ij}^\pi} (P_i^{stat})^2 \cdot V(t_{ij}^x, \pi).$$

Recall that a node remains active and consumes overhead power (O_i) until all cores within the node are finished with their workload. Let F_i be the maximum expected completion time among cores in node i , and σ_i^2 be the associated variance. The energy required to process the workload includes the overhead power, the dynamic power consumed by cores, and the static energy consumed by cores. We calculate the expected energy required to process the entire workload across all compute nodes, denoted ζ , as

$$(9) \quad \zeta = \sum_{i=1}^N \left(F_i \cdot O_i + \sum_{j=1}^{n_i} \sum_{\forall \pi \in PS_i} (Mean_{ij\pi}^{dyn} + Mean_{ij\pi}^{stat}) \right)$$

The variance of the energy required to process the entire workload, denoted γ , is

$$(10) \quad \gamma = \sum_{i=1}^N \left(\sigma_{ij}^2 \cdot (O_i)^2 + \sum_{j=1}^{n_i} \sum_{\forall \pi \in PS_i} (Var_{ij\pi}^{dyn} + Var_{ij\pi}^{stat}) \right)$$

The distribution for the total energy consumed to process the workload can be expressed as $\mathcal{N}(\zeta, \gamma)$. Given an energy budget of Δ , we can compute the probability that $\mathcal{N}(\zeta, \gamma)$ is less than Δ by converting $\mathcal{N}(\zeta, \gamma)$ to its associated cdf and finding the probability that the energy required to process the workload is less than Δ (i.e., $\mathbb{P}(\mathcal{N}(\zeta, \gamma) \leq \Delta)$). We denote this probability as ϕ , i.e., ϕ is the *energy-robustness* of a resource allocation. The energy-robustness constraint is denoted η , so for MO-EC we require resource allocations to meet this constraint (i.e., $\phi \geq \eta$).

2.5. HEURISTICS

2.5.1. OVERVIEW. The goal of this study is to design and analyze resource allocation heuristics with two different goals, MO-EC and EO-MC. In this section, we present three greedy heuristics and three non-greedy heuristics that have been adapted for our environment. The Minimum Expected Energy and Min-Min Completion Time (*Min-Min CT*) heuristics provided poor results, but were found to be useful as seeds for our non-greedy heuristics. We define a solution generated by a resource allocation technique as a complete mapping of tasks to both cores and P-states. Though we use Gaussian distributions for task execution times in our simulation study, our heuristics can use the mean values of any distribution to perform resource allocation.

2.5.2. MINIMUM EXPECTED ENERGY. The Minimum Expected Energy (*Min-Energy*) heuristic greedily assigns each task to the maximum makespan-robustness core in the node and P-state combination that minimizes the *expected* energy consumption of the task.

2.5.3. MIN-MIN COMPLETION TIME. Min-Min Completion Time (*Min-Min CT*) is a two-phase greedy heuristic, based on concepts in [13, 36–38]. We consider two variations of the heuristic, Min-Min P_{max} and Min-Min P_{min} , that differ in how P-state assignments are selected.

Min-Min P_{min} : All tasks are initially “unmapped” (placed in the *unmapped batch*). Each unmapped task is then paired to the core that yields the minimum expected completion time (*MECT*) when each core is considered to be executing in the *lowest-numbered* P-state (P-state 0). In the second phase, the task-core combination that yields the overall MECT is selected for assignment in P-state 0, and the task is removed from the unmapped batch. The ready times of all cores are updated and the heuristic begins another iteration. This

process continues until all tasks are mapped (i.e., the unmapped batch is empty).

Min-Min P_{max} : Same as Min-Min P_{min} except using the highest-numbered P-state.

2.5.4. MIN-MIN BALANCE. Min-Min Balance starts from an initial solution (either Min-Min P_{min} or Min-Min P_{max} and tries to improve the solution using greedy modifications. The *minimum makespan-robustness core*, denoted $core_{minM}$, refers to the core that has the least probability of finishing its assigned workload by the deadline, that is, the core that determines the makespan-robustness measure of the solution. The *maximum makespan-robustness core*, denoted $core_{maxM}$, refers to the core that has the greatest probability of finishing its workload by the deadline. We now show how we design the Min-Min Balance heuristic for MO-EC and EO-MC.

MO-EC: We start by generating an initial mapping using Min-Min P_{max} . The solution is then modified to increase makespan-robustness by reassigning tasks and P-states, keeping moves that improve the solution (i.e., greater makespan-robustness value without violating the energy-robustness constraint). The first step reassigns an arbitrary task from $core_{minM}$ to $core_{maxM}$ in the lowest-numbered P-state that does not violate the energy-robustness constraint. Then $core_{minM}$ and $core_{maxM}$ are recalculated, and this step is repeated until any task transferred from $core_{minM}$ does not result in an improved solution. The second step changes the P-state of an arbitrary task on $core_{minM}$ to a lower-numbered (i.e., better performing) P-state unless the energy-robustness constraint will be violated. If the constraint is not violated, $core_{minM}$ is recalculated and the process is repeated until decreasing the assigned P-state of any task on $core_{minM}$ violates the energy-robustness constraint.

EO-MC: We start by generating an initial mapping using Min-Min P_{min} . We use two steps that modify the allocation to improve energy-robustness, keeping moves that improve

the solution (i.e., better energy-robustness without violating the makespan-robustness constraint). The first step reassigns an arbitrary task from $core_{minM}$ to $core_{maxM}$ in the P-state that most improves energy-robustness without violating the makespan-robustness constraint. Then $core_{minM}$ and $core_{maxM}$ are recalculated, and this step is repeated until any task transferred from $core_{minM}$ does not result in an improved solution. The second step increases the value of the assigned P-state of an arbitrary task on $core_{maxM}$ by one unless the makespan-robustness constraint will be violated. If the constraint is not violated, $core_{maxM}$ is recalculated and the process is repeated until increasing the assigned P-state of any task on $core_{maxM}$ violates the constraint.

2.5.5. TABU SEARCH**.

2.5.5.1. *Overview.* The distinguishing feature of Tabu Search is its exploitation of memory through the use of a *Tabu List* [39]. We use a Tabu List to store regions of the search space that have been searched and should not be searched again. Our implementation of Tabu Search, based on concepts in [13], combines intelligent local search (“short hops”) with global search (“long-hops”) in an attempt to find a globally optimal solution.

Local search is performed using three short-hop operators: (1) *task swap*, (2) *task re-assignment*, and (3) *P-state reassignment*. One short-hop consists of one iteration of all three operators. Long-hops are performed when local search terminates, with the purpose of jumping to a new neighborhood in the search space, while avoiding areas already searched. After each long-hop, short-hops are again performed to locally search the region near the long-hop solution. The Tabu List stores unmodified long-hops (i.e., starting solutions) that indicate neighborhoods that have been searched before, and may not be searched again. A new solution generated by a long-hop must differ from any solution in the Tabu List by 25%

**Though we refer to this heuristic as Tabu Search in this thesis and our published version of this chapter [8], it could be more accurately described as an iterative local search.

Algorithm 1 Pseudo-code for our Tabu Search heuristic

1. **while** termination criteria not met **do**
 2. generate new long-hop, avoiding Tabu areas
 3. **while** solution is improving **do**
 4. *task swap*
 5. *task reassignment*
 6. *P-state reassignment*
 7. **end while**
 8. **end while**
-

of the task-to-core and P-state assignments, otherwise a new long-hop solution is generated.

Pseudo-code for the Tabu Search heuristic is given in Algorithm 1.

We now discuss how long-hops are performed and the purpose of the mean rank matrix before detailing the three short-hop operators (*task swap*, *task reassignment*, and *P-state reassignment*).

2.5.5.2. *Long-hops.* The purpose of a long-hop is to jump to new areas of the solution space to begin a new local search (i.e., short-hops), while avoiding areas of the search space that have already been searched through the use of a Tabu List. The initial solution (first long-hop) is generated using the appropriate Min-Min Balance allocation (for MO-EC or EO-MC) to help ensure that the constraints are met. Subsequent long-hop solutions are generated by first unmapping 25% of arbitrary tasks then reassigning them using Min-Min Balance, as before.

2.5.5.3. *Mean Rank Matrix.* We introduce the concept of a mean rank matrix that contains the rank of each heterogeneous node for each task, based on mean execution times. That is, for a given task, the nodes are ranked by how fast the nodes can execute the task (e.g., if node i can execute task t faster than node j , node i is given a better rank for task t). Let the rank of task t on node i be $rank(\mathbf{t}, \mathbf{i})$, where

$1 \leq rank(\mathbf{t}, \mathbf{i}) \leq N$. We define the best-ranked (fastest) node for a task as the *rank 1 node*.

When comparing the rank of any two tasks A and B on node i , task A is ranked lower

(better) than task B if $\text{rank}(\mathbf{A}, \mathbf{i})$ is less than $\text{rank}(\mathbf{B}, \mathbf{i})$. The mean rank matrix is used in some of the short-hop operators.

2.5.5.4. *Short-hops Overview.* The short-hop operators are used to perform greedy local search on a solution generated by a long-hop. *Task swap* swaps two tasks between two different cores, *task reassignment* transfers a task from one core to another, and *P-state reassignment* changes the P-state of a task. The *task reassignment* and *P-state reassignment* operators modify the assignments of specific tasks and cores, whereas *task swap* incorporates some randomness by selecting arbitrary cores to swap tasks. We found that incorporating some randomness with greedy intelligence provided the best results. We tried several variations of the three short-hop operators but only present our best-performing methods for the sake of brevity. The decisions made by the three short-hop operators change depending on whether it is desired to solve MO-EC or EO-MC. We first detail the operators when designed for MO-EC, then explain changes when designed for EO-MC.

2.5.5.5. *Short-hop Operators for MO-EC.* **Task Swap** The goal of the *task swap* operator is to swap tasks that are assigned to poorly (high) ranked nodes to better (low) ranked nodes, a move that can potentially improve both makespan and energy-robustness. We divide the task swap operator into four steps. (1) We first choose an arbitrary core j and create a task list consisting of all tasks assigned to core j on compute node i , recalling that the notation for such a task list is \mathbf{T}_{ij} . (2) \mathbf{T}_{ij} is sorted in descending order by the rank of each task for node i . (3) We select the first task in the list, denoted \mathbf{task}_A , and find the rank 1 node for the task, denoted \mathbf{node}_{best} . Within \mathbf{node}_{best} , an arbitrary core z is chosen. The task from core z that has the lowest rank for node i , denoted \mathbf{task}_B , is selected for swap. (4) The core assignments for \mathbf{task}_A and \mathbf{task}_B are swapped, and the best P-state combination (according to the method of constrained optimization used) is found to run the

cores in when executing the tasks. If the solution improves, the swap is kept and *task swap* ends. Otherwise, the swap is not kept, and *task swap* repeats using the next task in the list \mathbf{T}_{ij} until the solution improves or all tasks in \mathbf{T}_{ij} have been considered.

Task Reassignment The goal of *task reassignment* is to improve makespan-robustness by transferring tasks from the core with the worst makespan-robustness to another core. Task reassignment consists of three steps. (1) Find the minimum makespan-robustness core (core j on compute node i) and create a task list consisting of all tasks assigned to that core (\mathbf{T}_{ij}). (2) \mathbf{T}_{ij} is sorted in descending order by the rank of each task for node i . (3) We select the first task in the list (\mathbf{task}_A), and find the rank 1 node for the task (\mathbf{node}_{best}). Within \mathbf{node}_{best} , the highest makespan-robustness core is selected as the target core (core z), and \mathbf{task}_A is assigned to core z in the best P-state (according to the constrained optimization method). If the solution improves, the new assignment is kept and *task reassignment* ends. Otherwise, the new assignment is not kept, and *task reassignment* repeats using the next task in the list \mathbf{T}_{ij} until the solution improves or all tasks in \mathbf{T}_{ij} have been considered.

P-state Reassignment The goal of *P-state reassignment* is to change P-state assignments of tasks to greedily optimize the performance metric if the constraint is met, or to greedily meet the constraint if the constraint is violated. If the energy-robustness constraint has been met (i.e., $\phi \geq \eta$), the minimum makespan-robustness core (core j on compute node i) is chosen, and a task list is generated consisting of all tasks assigned to that core (\mathbf{T}_{ij}). A task is chosen arbitrarily from the list (\mathbf{task}_A), and the P-state of the task is decreased by 1 if not already currently assigned to execute in P0.

If the energy constraint has not been met (i.e., $\phi < \eta$), the maximum makespan-robustness core (core j on compute node i) is chosen, and a task list is generated consisting of all tasks assigned to that core (\mathbf{T}_{ij}). A task is chosen arbitrarily from the task list (\mathbf{task}_A),

and the P-state of the task is changed to the one that gives the highest system-wide energy-robustness.

For both cases, if the solution improves, the new P-state is kept and *P-state reassignment* ends. Otherwise, the new P-state is not kept, and *P-state reassignment* repeats using the next task in the list \mathbf{T}_{ij} until the solution improves or all tasks in \mathbf{T}_{ij} have been considered.

2.5.5.6. *Short-hop Operators for EO-MC.* **Task Swap** We make the following two changes to *task swap* to optimize for EO-MC instead of MO-EC. First, step 2 is changed to sort \mathbf{T}_{ij} in descending order of expected energy consumption instead of rank. Second, Step 3 is changed to find the minimum energy node for \mathbf{task}_A and selects \mathbf{task}_B from core z that consumes the least expected energy for node i .

Task Reassignment We change step 2 of *task reassignment* from MO-EC to sort \mathbf{T}_{ij} in descending order by expected energy consumption instead of rank, and in step 3 the minimum energy node is found as the destination for the transfer of \mathbf{task}_A rather than the rank 1 node.

P-state Reassignment If the makespan-robustness constraint has been met (i.e., $\Psi \geq \Gamma$), the maximum makespan-robustness core (core j on compute node i) is chosen, and a task list is generated consisting of all tasks assigned to that core (\mathbf{T}_{ij}). A task is chosen arbitrarily from the list (\mathbf{task}_A), and the P-state of the task is assigned to the P-state that gives the best energy-robustness.

If the makespan-robustness constraint has not been met (i.e., $\Psi < \Gamma$), the minimum robustness core (core j on compute node i) is chosen, and a task list is generated consisting of all tasks assigned to that core (\mathbf{T}_{ij}). A task is chosen arbitrarily from the list (\mathbf{task}_A), and the P-state of the task is decreased by 1 if not already currently assigned to execute in P0.

2.5.6. GENETIC ALGORITHM.

2.5.6.1. *Overview.* Genetic algorithms have been shown to be effective in resource allocation and job shop scheduling problems (e.g., [13, 40]). The Genitor [41] GA implemented in this study operates on a population of 200 chromosomes (determined empirically). Each chromosome is used to represent a solution (i.e., a complete resource allocation). A chromosome consists of a collection of $|\mathbf{T}|$ genes, where each gene represents a task assignment to a core and P-state. The initial population is generated using four solutions generated heuristically using Min-Energy, Min-Min \mathbf{P}_{max} and \mathbf{P}_{min} , and Min-Min Balance heuristics (using the appropriate Min-Min Balance method for MO-EC or EO-MC), and 196 randomly generated solutions.

After the initial population generation, all chromosomes in the population are evaluated and ranked (based on the method of constrained optimization used, detailed in Section 2.6). *Crossover* and *mutation* operators are used to generate offspring chromosomes by altering existing solutions. The GA enforces the population size by eliminating the lowest-ranked chromosomes such that the population remains fixed at its original size. We now discuss how crossover and mutation are used to generate new offspring.

2.5.6.2. *Crossover and Mutation.* Two crossover operators are used to swap task assignments or P-states between chromosomes: i) *task-assignment crossover* and ii) *P-state crossover*. Both crossover operators start by selecting two parent chromosomes using a linear bias [41] and two crossover points x and y are generated such that $x < y \leq |\mathbf{T}|$. In *task-assignment crossover*, all of the task-to-core assignments in genes ranging from x to y of the first chromosome are swapped with all of the task-to-core assignments in genes ranging from x to y of the second chromosome. Because cores on different nodes may have different numbers of P-states available, if a task changes nodes we assign the task in the P-state that

is closest to the clock frequency of its previous assignment. After *task-assignment crossover*, *P-state crossover* is performed. *P-state crossover* also considers the offspring generated by task-assignment crossover when selecting parent chromosomes (i.e., an intermediate population of 202 chromosomes). In *P-state crossover*, all of the P-state assignments in genes ranging from x to y of the first chromosome are swapped with all of the P-states in genes ranging from x to y of the second chromosome. Assume an offspring is created from genes 1 to $x-1$ and $y+1$ to $|\mathbf{T}|$ of parent A , and genes x to y of parent B . Gene i ($x \leq i \leq y$) of the offspring is assigned to the P-state of parent B that most-closely matches the frequency of the P-state of gene i in parent A . The crossover operators generate two new offspring each (four total).

Two mutation operators are used to alter task-to-core assignments and P-states: i) *task-assignment mutation*, and ii) *P-state mutation*. *Task-assignment mutation* is probabilistically performed on both of the offspring generated by *task-assignment crossover* and *P-state mutation* is probabilistically performed on both of the offspring generated by *P-state crossover*. For both mutations, offspring chromosomes have a probability \mathbf{p}_m of being mutated (empirically set to 0.1). If a chromosome is selected for mutation, each gene in that chromosome has a probability \mathbf{p}_{mg} of being mutated (empirically set to 0.001). In *task-assignment mutation*, if a gene is mutated the task corresponding to that gene is assigned to a random core (in a P-state selected as in *task-assignment crossover*). In *P-state mutation*, if a gene is mutated the task corresponding to that gene is assigned to a random P-state. After crossover and mutation, offspring chromosomes are added to the population, evaluated (as specified in Section 2.6), and the least-fit chromosomes are discarded to bring the population back to its original size. This completes one generation of the GA. This process is repeated for a

predetermined time limit (see Section 2.7), and the most-fit chromosome is returned as the solution.

2.5.6.3. *Differences between MO-EC and EO-MC.* The genetic algorithm is versatile because its intelligence lying in how solutions are evaluated and ranked. Therefore the only differences between MO-EC and EO-MC are the different Min-Min Balance seeds used and how the chromosomes are ranked (detailed in Section 2.6).

2.5.7. GENETIC ALGORITHM WITH LOCAL SEARCH. Our genetic algorithm with local search (GALS) combines the population-based global search from the GA with the local search techniques from our Tabu Search heuristic. After both crossover and mutation operations are finished (as performed in the GA), a local search is applied to the offspring chromosomes using the three local search operators from our Tabu Search heuristic under the condition that an offspring chromosome is *at least* as “good” as the worst chromosome in the population, so as to not waste time trying to improve a poor solution. The number of iterations of local search on each offspring chromosome was experimentally found to provide the best results at 200 iterations.

The differences between MO-EC and EO-MC for GALS are the same as for the GA, and the short-hop operators from Tabu Search (detailed in Sections 2.5.5.5 and 2.5.5.6).

2.6. CONSTRAINED OPTIMIZATION

2.6.1. OVERVIEW. The previous section described the heuristics we propose to use for our resource allocation problem. Incorporating constraints into heuristics that are typically designed to optimize for an unconstrained objective (e.g., Tabu Search and GA) is a difficult problem and a research topic in itself. In this section we present several constraint-handling methods adapted from the literature [42–45] that we use in combination with some of the

proposed heuristics. These methods help us determine solutions that are “better” than other alternatives over the search space examined by the heuristics. In this section, we present the “static penalty function,” “dynamic penalty function,” and “superiority of feasible solutions” techniques. Details of the less-effective “limiting the search space” technique can be found in Appendix C.

2.6.2. STATIC PENALTY FUNCTION. The static penalty function technique [42, 44, 45] reduces the objective function value of infeasible solutions based on the solution’s distance from feasibility, but still allows infeasible solutions to be considered when searching for an optimal feasible solution. The distance from feasibility of a solution for MO-EC is

$$(11) \quad \mathbf{d}_\phi = \boldsymbol{\eta} - \boldsymbol{\phi}.$$

For EO-MC, the distance from feasibility is

$$(12) \quad \mathbf{d}_\Psi = \boldsymbol{\Gamma} - \boldsymbol{\Psi}.$$

We denote \mathbf{c} as a constant used to control how strongly a constraint will be enforced. Our penalized objective function for MO-EC is

$$(13) \quad \psi_\Psi = \begin{cases} \boldsymbol{\Psi} - \mathbf{c} \cdot \mathbf{d}_\phi & \text{if } \mathbf{d}_\phi > \mathbf{0} \\ \boldsymbol{\Psi} & \text{if } \mathbf{d}_\phi \leq \mathbf{0} \end{cases}.$$

Our penalized objective function for EO-MC is

$$(14) \quad \psi_\phi = \begin{cases} \boldsymbol{\phi} - \mathbf{c} \cdot \mathbf{d}_\Psi & \text{if } \mathbf{d}_\Psi > \mathbf{0} \\ \boldsymbol{\phi} & \text{if } \mathbf{d}_\Psi \leq \mathbf{0} \end{cases}.$$

When \mathbf{d}_ϕ or \mathbf{d}_Ψ are greater than zero, it indicates that the constraint has not been met. We then penalize the objective functions (ψ_ϕ or ψ_Ψ) by subtracting a weighted value of the distance from feasibility. A high value of the coefficient \mathbf{c} (i.e., high penalty for an infeasible solution) can produce low quality solutions by restricting exploration of the infeasible region. However, \mathbf{c} must be large enough that a feasible solution is found. In our experiments, the best results were obtained when setting \mathbf{c} to 2.

When \mathbf{d}_ϕ or \mathbf{d}_Ψ are less than or equal to zero it indicates the constraint has been met. Solutions are not rewarded when \mathbf{d}_ϕ or \mathbf{d}_Ψ are less than zero, as all that matters is the constraint is not violated. Heuristics incorporating the static penalty function return the best solution encountered that meets the constraint.

2.6.3. DYNAMIC PENALTY FUNCTION. The static penalty function has a primary deficiency in that solutions obtained greatly depend on the penalty weight \mathbf{c} , and a “good” value for \mathbf{c} will vary depending on the heuristic used and even from iteration to iteration within a heuristic. A dynamic penalty function [44, 45] uses knowledge of the current search state to guide the search along the boundary of feasibility where the optimal solution is likely to occur. For evolutionary algorithms (e.g., GA and GALS), this is done by adjusting the penalty weight to guide the search in such a way that the population has an equal number of feasible and infeasible solutions. We set the penalty weight (\mathbf{c}) to 2, and at the end of each generation of the GA or GALS, the penalty weight is increased by a small amount (0.01) if less than half of the population are feasible solutions or decreased by a small amount (0.01) if at least half of the population are feasible solutions. For Tabu Search, the penalty weight is increased by 0.01 at the end of an iteration if the solution is infeasible or decreased by 0.01 if feasible. Heuristics incorporating the dynamic penalty function return the best solution encountered that meets the constraint.

2.6.4. SUPERIORITY OF FEASIBLE SOLUTIONS. By adopting the rule that any feasible solution is better than any infeasible solution [43], it is possible to avoid experimentally tuning penalty parameters. Our heuristics use this method in the following way: (1) any feasible solution is better than any infeasible solution, (2) when two infeasible solutions are compared, the one with the smallest distance from feasibility (see Equations 11 and 12) is considered better, and (3) when two feasible solutions are compared, the one with the better objective function value is considered better.

2.7. RESULTS

We consider two different simulation sizes in our simulation study. The small simulation size consists of 25 compute nodes (N), based on 25 different servers listed in the SPECpower_ssj2008 results [46], and 10,000 tasks. The large simulation size consists of 250 compute nodes and 100,000 tasks. In Appendix A we provide data collected from SPECpower_ssj2008, give details on how we use this data for our system parameters, provide information on how we obtain our voltage/frequency values for P-states, and workload generation details.

We conducted simulations to find a balance of short-hops and long-hops in Tabu Search (see Appendix B), compare the different heuristics using our constrained optimization techniques (Fig. 2.1), demonstrate the effectiveness of each heuristic at handling various degrees of difficulty to meet the deadline and energy budget (Figs. 2.2 and 2.3), analyze the trends on the large simulation size compared to the small simulation size (Figs. 2.4 and 2.5), and perform a sensitivity analysis of our heuristics across environments of varying heterogeneity (Fig. 2.6). Results in this section show the mean and 95% confidence interval error bars of 96 trials, with the means and variances for task execution times varying between trials. These trials simulate numerous diverse heterogeneous workload/system environments. For

the sake of brevity, we do not include results for the Min-Energy or Min-Min P_{min} and P_{max} heuristics as they performed poorly by themselves but were found useful as seeds in the GA, GALS, and Tabu Search heuristics. Unless otherwise stated, for experiments considering the small simulation size, the system deadline (δ) was set to 15,500 seconds, the energy budget (Δ) was set to 58 megajoules (MJ). For the large simulation size, the system deadline was set to 13,500 seconds, the energy budget was set to 580 MJ. In both cases, the energy/makespan-robustness constraints (η and Γ) were set to 90%.

Figure 2.1 compares the different methods of constrained optimization for Tabu Search, GA, and GALS for the MO-EC problem using the small simulation size. Heuristics are terminated after six hours of heuristic execution time. The heuristics show similar trends for the methods of constrained optimization. All heuristics and methods of constrained optimization produced solutions that met the energy-robustness constraint. The dynamic and static penalty functions can sometimes prefer infeasible solutions with a large objective value over feasible solutions with a smaller objective value, based on the relative value of Ψ and the weighted distance from feasibility. This can help the heuristics obtain a better objective value by allowing exploration into the infeasible region and guiding the search towards a better feasible solution in the end. The “superiority of feasible solutions” and “limiting the search space” techniques always prefer feasible solutions over infeasible ones, hindering the ability of the heuristics to accept high makespan-robustness solutions that may barely violate the energy-robustness constraint and eventually guide the heuristics to better feasible solutions, which led to worse results than the penalty function techniques.

The static penalty function must have a high enough penalty weight so that a feasible solution is found, but small enough such that the heuristics sometimes allow infeasible solutions to be accepted. We experimented with setting c equal to 0.5, 1, 2, and 5, and found the

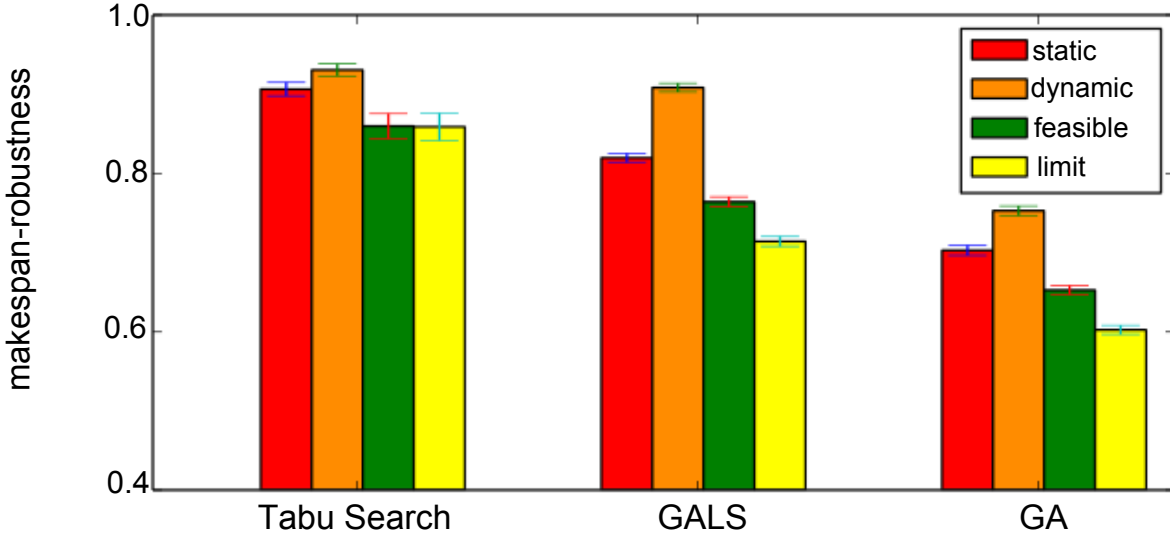


FIGURE 2.1. Comparison of constrained optimization techniques for MO-EC with Tabu Search, GALS, and GA (25 node system, 458 total cores, and 10,000 tasks). The system deadline was set to 15,500 seconds, the energy budget was set to 58 MJ, and the energy-robustness constraint was set to 90%.

best results when setting c equal to 2, which allowed the heuristics to accept some infeasible solutions but to mostly prefer feasible ones. Compared to the static penalty function, the dynamic penalty function is able to fine tune the penalty weight over the course of the search by adapting the penalty weight after each iteration of Tabu Search or generation of GALS and GA, resulting in better solutions than the static penalty function.

The “limiting the search space” technique is the same as the “superiority of feasible solutions” for Tabu Search, as Tabu Search starts with a Min-Min Balance solution that is feasible for the deadline and energy budget considered. However, for GALS and GA, the “limiting the search space” technique performs the worst as the initial population is generated with only feasible solutions, causing the chromosomes to lack diversity and hindering

the benefits of the crossover operator. The results for EO-MC show the same trends as Figure 2.1 across constrained optimization techniques. Because the dynamic penalty function performed best, we use this as our method of constrained optimization for other experiments.

Figures 2.2 and 2.3 show the results of the heuristics at handling different values for the energy budget (Δ) and deadline (δ) for the small simulation size. Figure 2.2 shows the results of the heuristics when maximizing makespan-robustness (Figure 2.2(a)) with an energy-robustness constraint (Figure 2.2(b)) when varying the energy budget (i.e., varying the difficulty of meeting the energy constraint). Figure 2.3 shows the results of the heuristics when maximizing energy-robustness (Figure 2.3(a)) with a makespan-robustness constraint (Figure 2.3(b)) when varying the deadline. The energy-robustness and makespan-robustness constraints (η and Γ) were set to 90%.

For the MO-EC problem, Figure 2.2 shows how well the heuristics perform at exploiting the trade-off between energy-robustness and makespan-robustness to sacrifice the probability of meeting the deadline (makespan-robustness) to meet the energy constraint. For this experiment, each of the 96 trials were executed with the deadline set to 15,500 seconds and the energy budget set to a value between 50 MJ and 64 MJ in 0.5 MJ increments. Each trial of the Tabu Search, GALS, and GA heuristics was terminated after six hours. Within this time, Figure 2.2(a) shows that Tabu Search is able to obtain the best solutions at all energy budgets. Tabu Search and GALS are able to outperform GA in the allotted time as the intelligent local search operators used in Tabu Search and GALS are able to quickly identify moves that improve the solution, whereas the GA must rely on random genetic search which can go through several generations before finding better solutions. Tabu Search focuses on performing many short-hops on one solution and escaping local optima through long-hops,

which outperforms the GALS that performs fewer short-hops (i.e., iterations of local search) on many different solutions over the course of the search and relies on random crossover and mutation to escape local optima.

We can observe that Tabu Search is able to achieve non-zero makespan-robustness and meet the energy-robustness constraint at energy budgets as small as 51 MJ, due to the local search operators intelligently assigning tasks to execute in low-power P-states on high-ranked nodes. In Figure 2.2(b), at energy budgets up to 53 MJ for GALS and 55 MJ for GA, GALS and GA return solutions of 1.0 energy-robustness (feasible) but 0.0 makespan-robustness. At these tight energy budgets, the Min-Min and Min-Min Balance seeds and randomly generated solutions of GALS and GA have very poor energy-robustness and are therefore highly penalized, so the population quickly converges to solutions similar to the Min-Energy solution that has 1.0 energy-robustness and poor makespan-robustness. Also, when the energy budget is set to greater than approximately 60 MJ, the energy-robustness exceeds our set constraint of 0.9 for the GA and Min-Min Balance heuristics, indicating that when given a large energy budget, GA and Min-Min Balance are able to achieve feasible solutions but unable to use all available energy to increase makespan-robustness as desired. This is because the GA and Min-Min Balance heuristics do not incorporate the *P-state reassignment* operator that greedily assigns tasks to run in faster P-states until all available energy is used (e.g., until energy-robustness equals the constraint of 0.9). At higher energy budget values (over 60 MJ), GALS and Tabu Search have similar performance because the energy budget becomes easy enough that the energy-robustness constraint can be achieved when most tasks are running in the highest-power P-state (P0) (through the *P-state reassignment* operator for MO-EC), thus both heuristics can attain high makespan-robustness values while meeting the energy-robustness constraint.

Figure 2.3 compares results of heuristics for EO-MC when varying the deadline (i.e., making the makespan-robustness constraint more or less difficult to attain). For this experiment, the energy budget is fixed at 58 MJ and system deadline is varied between 13,500 seconds and 17,000 seconds. Again, we can observe that Tabu Search and GALS outperform the GA, indicating the significance of the local search operators for the smaller simulation

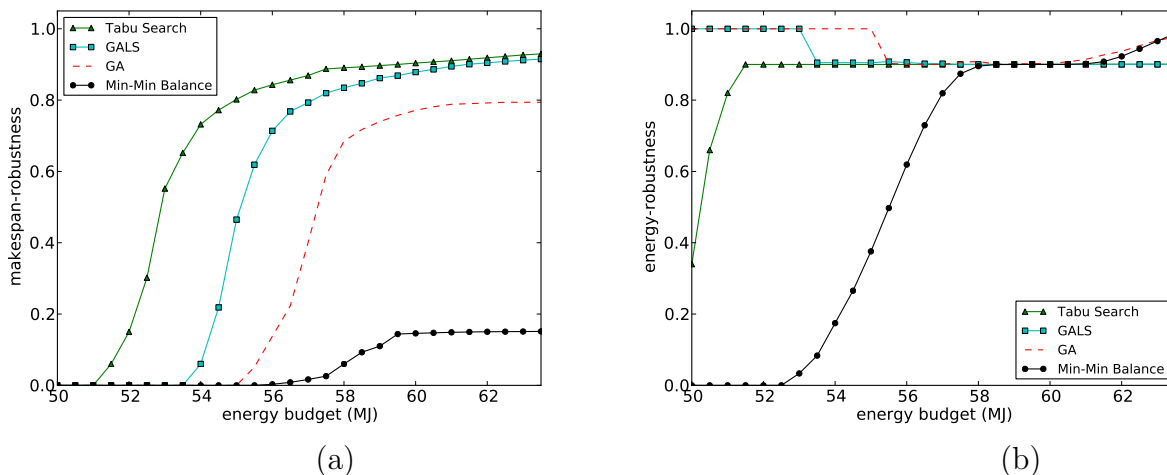


FIGURE 2.2. Varying the energy budget for MO-EC: (a) makespan-robustness, and (b) energy-robustness (25 node system, 458 total cores, and 10,000 tasks). The system deadline was set to 15,500 seconds, the energy-robustness constraint was set to 90%. The heuristics were terminated after six hours of execution time.

size. Tabu Search outperforms GALS at most deadline values, as the benefit associated with improving one solution through using many short-hops and escaping local optima with long-hops exceeds the benefit associated with performing fewer local search iterations on numerous solutions and relying on random crossover and mutation to escape local optima, as in GALS. At higher deadline values (over 16,200s), GALS and Tabu Search have similar performance because the deadline becomes easy enough that the makespan-robustness constraint can be achieved when most tasks are running in the lowest-power P-states (through

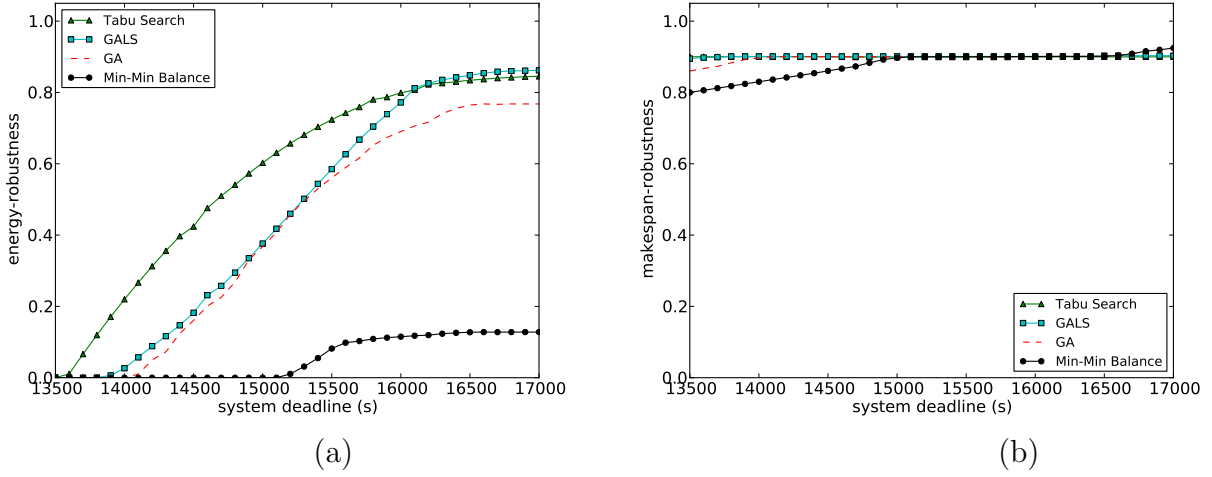


FIGURE 2.3. Varying the system deadline for EO-MC: (a) energy-robustness, and (b) makespan-robustness (25 node system, 458 total cores, and 10,000 tasks). The energy budget was set to 58 MJ and the makespan-robustness constraint was set to 90%. The heuristics were terminated after six hours of execution time.

the *P-state reassignment* operator for EO-MC), thus both heuristics can attain high energy-robustness values while meeting the makespan-robustness constraint.

The 25 node (458 total cores) platform is relatively small for a modern HPC system, so we also experimented with our heuristics on a larger simulated platform consisting of 250 nodes (4,580 total cores) and 100,000 tasks. Figure 2.4 compares results of Tabu Search and GALS over 72 hours of heuristic execution time for MO-EC. With the smaller simulation size of only 25 nodes and 10,000 tasks, Tabu Search often achieved the best results (see Figs. 2.1, 2.2, 2.3). However, Figure 2.4 shows that for 12 hours of heuristic execution time for the larger simulation size, GA outperforms both Tabu Search and GALS, and GALS outperforms Tabu Search. The local search operators used by Tabu Search and GALS are not as effective on the large simulation size, and there are two primary reasons: (1) the local search operators used by Tabu Search and GALS take considerably longer to execute when having to examine considerably more cores and tasks to intelligently swap and reassign tasks

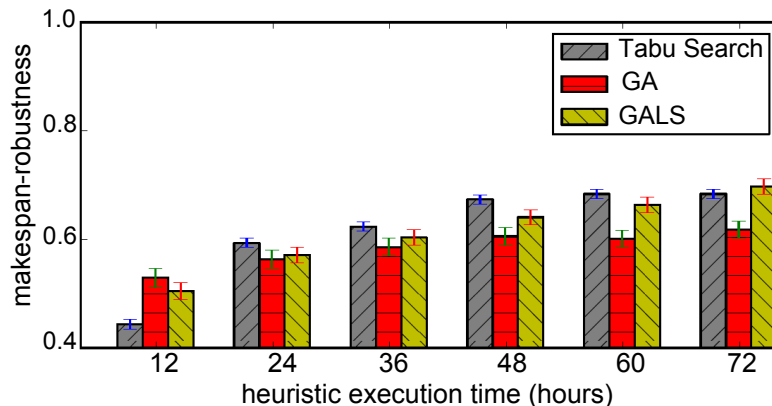


FIGURE 2.4. Comparison of Tabu Search, GA, and GALS heuristics over 72 hours of execution time for MO-EC using the large simulation size (250 nodes, 4,580 total cores, and 100,000 tasks). The system deadline was set to 13,500 seconds, the energy budget was set to 580 MJ, and energy-robustness constraint was set to 90%.

and P-states, and (2) local search operators only change one task and/or P-state assignment each, which can result in small improvements per iteration when considering 100,000 tasks instead of only 10,000 tasks. Genetic search (crossover and mutation) can change numerous task and P-state assignments per generation, which can lead to large improvements early in the heuristic. Over time, however, the randomness of the genetic search becomes less effective and improvements become incremental, leading the intelligent choices made by the Tabu Search and GALS to outperform GA. To help illustrate these observations, Figure 2.5 compares the progress of one trial of Tabu Search, GALS, and GA over 72 hours of heuristic execution time using the small simulation size (Figure 2.5(a)) and large simulation size (Figure 2.5(b)).

On the small simulation size (Figure 2.5(a)), Tabu Search is able to perform approximately 350,000 total iterations of local search over the 72 hours of heuristic execution time, however on the large simulation size (Figure 2.5(b)) Tabu Search is only able to perform

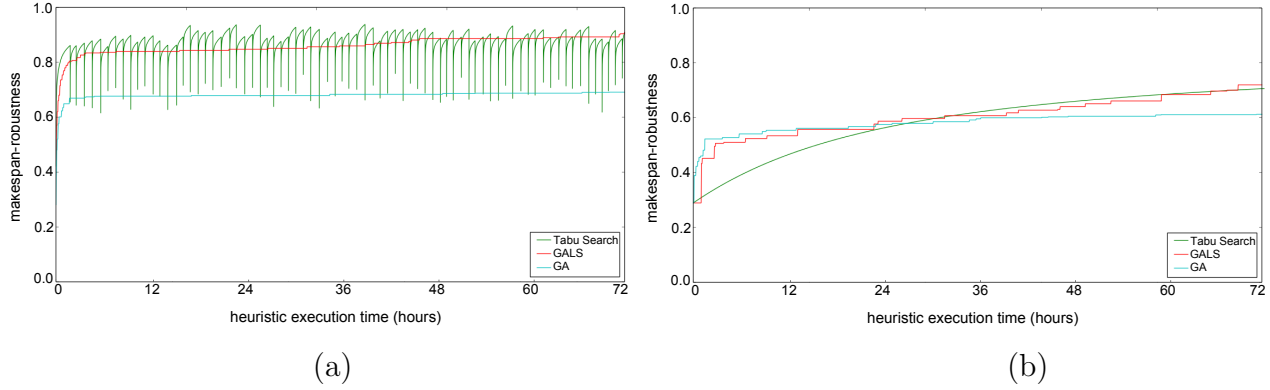


FIGURE 2.5. Progress of Tabu Search, GALS, and GA for MO-EC over 72 hours of heuristic execution time for the (a) small simulation size (25 nodes, 458 total cores, 10,000 tasks), and (b) large simulation size (250 nodes, 4,580 total cores, and 100,000 tasks). For the small simulation size, the system deadline was set to 15,500 seconds, the energy budget was set to 58 MJ, and the energy-robustness constraint was set to 90%. For the large simulation size, the system deadline was set to 13,500 seconds, the energy budget was set to 580 MJ, and energy-robustness constraint was set to 90%.

about 30,000 total iterations of local search, due to the increased time it takes for the short-hop operators to identify intelligent assignments. Figure 2.5(a) shows Tabu Search being very effective on the small simulation size compared to GALS and GA, as the intelligent short-hop operators are fast and can focus on improving one solution until a long-hop is performed (the sharp decreases in makespan-robustness). The genetic search of GALS and GA leads to large improvements early, but the genetic search becomes less effective as the population becomes less diverse over time. GALS is able to perform few local search iterations on many different solutions, leading to a gradual improvement over time. Figure 2.5(b) shows that with the large simulation size, Tabu Search does not perform any long-hops over 72 hours of heuristic execution time, meaning that the local search has yet to meet the termination criteria. We can see that the genetic search of GALS and GA is able to perform large jumps to better solutions early compared to the gradual improvement of Tabu Search due to crossover being able to change numerous assignments per generation through random

recombination of chromosomes. The genetic search becomes less effective as the population converges, and Tabu Search obtains similar results to that of GALS after approximately 24 hours.

Figure 2.6 compares the makespan-robustness results for MO-EC using our Tabu Search, GALS, and GA heuristics when the computing environment has high and low heterogeneity to evaluate the effectiveness of our heuristics across different heterogeneous environments. One measure for characterizing the heterogeneity of a computing environment is task machine affinity (TMA) [47]. TMA captures the degree to which some tasks are better suited to run on specific machines. In an environment with low TMA, typically a node that is faster for one task is typically faster for all tasks. In contrast, an environment with high TMA contains nodes that are better-suited for some tasks, but other machines are better-suited for different tasks. We use methods detailed in [48] to modify the mean task execution time values from the CoV method (Appendix A) to create high and low TMA environments.

Figure 2.6 shows all heuristics performing better on environments with high TMA than low TMA. In the environment with low TMA, all tasks execute the fastest on cores

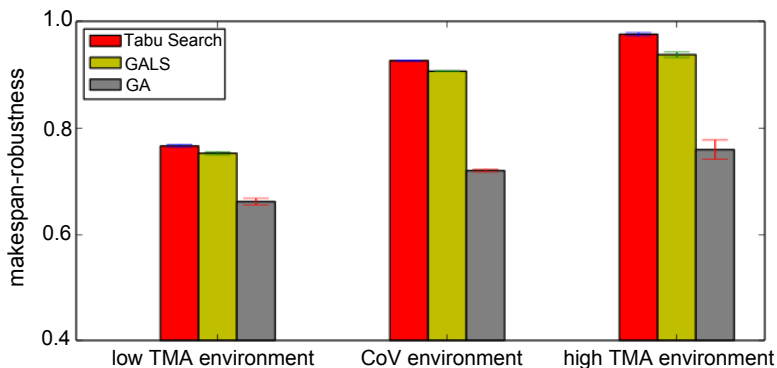


FIGURE 2.6. Comparison of Tabu Search, GALS, and GA across heterogeneous environments with varying TMA for MO-EC using the small simulation size (25 nodes, 458 total cores, and 10,000 tasks). The system deadline was set to 15,500 seconds, the energy budget was set to 58 MJ, and the energy-robustness constraint was set to 90%.

in the fastest node, leaving many tasks executing on subpar nodes when the workload is load balanced. In the high TMA environment, different tasks execute fastest on different nodes, leaving many tasks assigned to their fastest nodes when the workload is load balanced. We can also observe that in the high TMA environment, the makespan-robustness of Tabu Search and GALS exceed the performance of the GA by far more than in the low TMA environment. This is because the *task swap* and *task reassignment* local search operators employed by Tabu Search and GALS move tasks onto nodes that are better according to the mean rank matrix. In a high TMA environment, this makes the *task swap* and *task reassignment* operators very effective because placing tasks on their best-ranked nodes often leads to natural load balancing, giving high makespan-robustness. In the heuristic execution time given, Tabu Search and GALS greatly outperform the random combinations produced by GA. In the low TMA environment, placing tasks on better-ranked nodes does not load balance, leading to Tabu Search and GALS giving performance closer to that of the GA.

In summary, we found that the dynamic penalty function served as the most-effective constrained optimization technique (Fig. 2.1). Also, our Tabu Search heuristic gave the best results among the heuristics for the small simulation size (Figs. 2.2 and 2.3) and the large simulation size when given at least 24 hours to execute (Figs. 2.4 and 2.5). Tabu Search was also able to provide the best results in environments with low and high heterogeneity (Fig. 2.6).

2.8. CONCLUSIONS

In this chapter, we design energy-aware resource allocation techniques to address two challenges that appear in today’s data centers: (a) trying to optimize the makespan when subject to an energy budget constraint, and (b) trying to optimize energy consumption when subject to a makespan deadline. This problem becomes more complex when the execution

times are modeled stochastically rather than deterministically. We develop probabilistic measures for both makespan and energy consumption, which we call makespan-robustness and energy-robustness. Makespan-robustness is the probability of meeting a makespan deadline, and energy-robustness is the probability of meeting an energy budget.

We approach this problem through the design of energy-aware resource allocation techniques incorporated with methods of constrained optimization from the literature. We compared the methods of constrained optimization using our Tabu Search, GALS, and GA heuristics. For a small simulation size, the intelligent search techniques of Tabu Search in combination with the dynamic penalty function outperformed the other resource allocation methods within the computation time given to execute the heuristics. For the large simulation size, however, the computation time overhead of our intelligent local search operators caused GA and GALS to outperform Tabu Search unless Tabu Search is given at least 24 hours to execute. The comparison of heuristics and constrained optimization techniques revealed great potential for our Tabu Search and GALS heuristics when combined with the dynamic penalty function for managing compute resources in an energy-aware manner for both deadline-constrained and energy-constrained systems. Possible directions for future studies in this area are presented in Chapter 6.

CHAPTER 3

RATE-BASED THERMAL, POWER, AND CO-LOCATION AWARE RESOURCE MANAGEMENT FOR HETEROGENEOUS HIGH PERFORMANCE COMPUTING SYSTEMS[†]

3.1. INTRODUCTION

Faster execution time has long been the highest priority design objective for high-performance computing (HPC) systems. The great success achieved thus far in optimizing performance of these systems has come at the cost of increased power consumption, leading to increased ownership costs from powering and cooling these systems. Furthermore, the increased demand for performance from these systems has led to chip designers increasing the number of cores in their multicore processors, and led HPC system administrators to include these high-performance multicore processors in their facilities. The increased number of processing chips in HPC also has increased the power consumption of computing and cooling. The increased number of cores per chip has led to greater interference between cores competing for shared resources. The push to exascale will require even more on-chip parallelism, thereby reducing performance and requiring larger systems to meet needs, thus exacerbating power consumption.

Motivated by the need to reduce electricity costs and encourage green computing, the Green500 list aims to switch the paradigm of performance as a primary design objective

[†]This work was performed jointly with student Eric Jonardi, and is currently under review. The other co-authors of this work are: Sudeep Pasricha, Anthony A. Maciejewski, Gregory A. Koenig, Patrick J. Burns, and H. J. Siegel. A preliminary version of portions of this work appeared in [4]. This research was supported by NSF grants CNS-0905399, CCF-1302693, and CCF-1252500. This research used the CSU ISTE_C Cray System supported by NSF Grant CNS-0923386. We thank Hewlett Packard for donating servers for this work.

to metrics such as energy efficiency and performance-per-watt as higher priority design objectives. The L-CSC supercomputer currently tops the Green500 list with a performance-to-power ratio of 5.27 GFLOPS/Watt [49]. A linear extrapolation of the L-CSC system to exascale results in a power consumption of 190 MW, or approximately \$60 million per year in electricity in the United States. The Defense Advanced Research Projects Agency (DARPA) has set the target for an exaflop capable system to consume no more than 20 MW [6], nearly an order of magnitude of power consumption lower than today's most power-efficient supercomputer.

Designing new power-aware and thermal-aware resource management techniques is one method to reduce power consumption of an HPC system and alleviate the amount of cooling to further reduce power consumption and total cost of ownership. Examples of power-aware and thermal-aware resource management include exploiting different power and performance characteristics of heterogeneity across compute nodes, configuring dynamic voltage and frequency scaling (DVFS) in cores, and setting computer room air conditioning (CRAC) thermostats to higher temperatures during operation. These decisions must ensure the temperatures of the compute nodes do not exceed the red-line threshold, defined as the maximum allowable equipment intake temperature [50].

The prevalence of multicore processors in modern HPC systems has given rise to heavy contention for shared resources among the cores, such as at the last-level cache and DRAM. This has led to performance degradation when running applications on cores that share these resources, with a significant impact on the execution time of those applications [51]. The performance degradation can range from negligible to severe, and the amount of degradation is correlated with such attributes as: how many applications are co-located (i.e., running on cores within the same processor), the memory intensity of the co-located applications

(defined as the ratio of last-level cache misses to the total number of instructions executed), and clock frequency of the target core. With knowledge of these workload characteristics, it is possible to predict the execution time degradation of applications under co-location interference effects [52]. Typically, performance degradation is more severe when memory intensive applications are co-located, because they try to access shared memory simultaneously more often than compute intensive applications. Therefore, to mitigate interference, it is intuitive to co-locate memory intensive applications with compute intensive applications.

We consider an HPC system with workload arrival rates that can be predicted over a decision interval (epoch) [53, 54], where the task arrival rates, temperatures at compute nodes and CRAC units, and the power consumption of the computing system and CRAC units remain virtually invariant over that interval of time. That is, a day split into epochs, and over the course of an epoch the workload arrival rates can be reasonably approximated as constant, e.g., the Argonne National Lab Intrepid log that shows mostly-constant arrival rates over large intervals of time [55]. The performance of the HPC system 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 a rate-based resource management framework, maximizing reward is equivalent to maximizing the *reward rate*. The goal of our resource management techniques is to maximize reward rate. Our techniques mitigate the impact of co-location interference by maximizing a reward rate objective function that considers co-location interference. The problem we solve is to maximize the reward rate earned by the system while obeying red-line temperature thresholds and a power constraint on the whole facility (both compute and cooling power). By taking a holistic approach to the control of such a facility, we maximize the reward rate earned while ensuring that the compute nodes do not exceed their red-line temperatures and the total power consumed by

the compute nodes and CRAC units do not exceed a given power constraint. We assume an oversubscribed environment where it would be common to execute the workload in the fastest P-state, however due to the power budget constraint, our resource management techniques intelligently select P-states to earn more reward within that power budget. To solve this optimization problem, we design a new greedy heuristic, a genetic algorithm (GA) based approach, and improve a non-linear programming (NLP) approach from our previous work [56] to consider the effect of co-locating tasks on multicore processors. We perform extensive analyses of these techniques on different workload environments, HPC facility cold-aisle isolation configurations, and HPC system sizes (large and small).

In summary, we make the following novel contributions:

- Derivation of a new detailed model of a heterogeneous HPC system that considers the power consumption and performance of the compute nodes and cooling system, thermal constraints, DVFS, and co-location interference.
- Design of resource management techniques based on a greedy heuristic, a GA with local search, and an adaptation of a previously proposed NLP approach with consideration of co-location while ensuring that power consumption and thermal constraints are obeyed.
- In-depth analyses of resource management techniques under several important physical HPC facility configurations with different system sizes and cold-aisle isolation capabilities.
- Sensitivity analysis of our resource management techniques under a range of values for the power and thermal constraints across different cold-aisle isolation configurations and workload environments.

The rest of the chapter is organized as follows. We discuss related work in Section 3.2. In Section 3.3, we explain our models for compute nodes, CRAC units, and workload, as well as how we consider co-location interference. Section 3.4 describes our proposed resource allocation techniques. Our evaluation setup and results are in Sections 3.5 and 3.6. In Section 3.7, we conclude and discuss ideas for future work.

3.2. RELATED WORK

3.2.1. OVERVIEW. HPC systems and data centers use a large amount of power, resulting in high electricity costs and a large carbon footprint. In an attempt to encourage green data center design by reducing the power consumption in data centers, the authors of [57] have identified four factors that can drive green data center design: (1) improved physical design to reduce heat recirculation and hot-spots, (2) using highly energy-efficient computing and cooling systems, (3) using sustainable energy sources such as solar power, and (4) using intelligent computing resource management strategies. In this study, we choose to focus on the last point by designing and analyzing green resource management techniques. Temperature, power, and performance of the compute nodes are all tightly coupled, and thus we take a holistic approach to HPC resource management that considers heterogeneous compute nodes, spatial temperature knowledge, DVFS performance states (P-states), CRAC outlet temperatures, and interference caused by co-location on multicore processors.

3.2.2. THERMAL-AWARE SCHEDULING. While larger data centers such as Google’s have small power usage effectiveness (PUE) and therefore only 6% of the power consumption is due to cooling and power distribution [58], a survey conducted in 2014 found an average data center PUE of 1.7 that implies approximately 40% of data center power consumption is due to cooling and power distribution [59]. Also, our collaboration with Oak Ridge National Labs

(ORNL) has demonstrated that HPC systems (e.g., Titan) have significant cooling power costs. A holistic approach to management of both the cooling and computing components of an HPC facility is vital in reducing electricity costs. The power and temperature relationship between the compute nodes and cooling infrastructure is highly correlated. That is, reducing the power of compute nodes also reduces the temperatures of those nodes, therefore easing the load on the cooling infrastructure and reducing cooling costs. However, thermal-aware and power-aware computing decisions are often conflicting. Power-aware computing techniques often attempt to concentrate the workload to a few active nodes, allowing those nodes without a workload to switch into an idle state or be deactivated. Intuitively, this leads to high heat dissipation in a small area, creating thermal hotspots and increasing the local cooling power for just a small number of the nodes. Thermal-aware scheduling techniques reduce cooling power by minimizing hotspots and total heat generated [60].

Both minimizing hotspots and total heat generated is examined in [61], where virtual machines are dynamically migrated based on temperatures and compute node loads to try and satisfy the conflicting objectives of reducing cooling costs by spreading the workload, and reducing compute node power costs by consolidating the workload. The fundamental idea of their migration algorithm is to detect overheating compute nodes (through onboard sensors), and migrate virtual machines from an overheated compute node to the coolest one in the facility. By placing a temperature cap for detecting overheating, and then migrating loads from hot nodes to the coolest ones, hotspots are managed and heat is spread more evenly throughout the room. But neither DVFS nor heat recirculation is considered.

The research in [62] takes an interesting approach in its thermal-aware workload management by considering multiple facilities, each located in different geographical locations. Each facility has intermittent and unreliable renewable energy sources attached (e.g., wind

or solar), making it sensible to migrate “time insensitive” batch workloads to facilities with high renewable energy generation at a given point in time. The computing facilities also are equipped with a thermal energy storage (TES) system. The intuition behind the two kinds of thermal energy storage systems in today’s computing facilities are to either: (1) excessively cool the HPC facility when an abundance of renewable energy is available so it remains at a reasonable temperature when no renewable energy is available, or (2) attach a dedicated thermal storage unit that uses renewable energy to chill liquid for use later. That paper considers the latter, and models the thermal storage unit similarly to a battery with a capacity, charge, and discharge rate. The problem is formulated as an economics problem to minimize the operating cost (energy and “bandwidth” cost for migration) to complete the workload. Unlike [62], our research considers heterogeneity of compute nodes, DVFS, and CRAC unit control in our resource management techniques.

There have been many recent works that consider thermal-aware resource management (e.g., [63–69]). Some do not consider heterogeneity in their work ([62, 69, 61]) or recirculation of heat in the room ([61, 62, 66, 68]). Others do not consider DVFS control ([62, 64–68, 61]) or CRAC unit control ([62, 67, 68, 61]), leaving the thermal-aware allocation choices limited to only turning nodes on/off or basing the choices on CPU utilization. Lastly, some thermal-aware works only examine thermal and power metrics, while not considering performance metrics of any sort ([61]). Today’s systems typically require abiding by performance constraints to ensure either deadlines or SLAs are not violated.

3.2.3. CO-LOCATION INTERFERENCE. Multicore processors are prominent in today’s HPC systems, and the number of cores per processor is increasing. In such processors, many (if not all) cores share resources such as the last-level cache, causing contention and performance degradation for applications executing on those cores because of the thrashing

in shared caches or competition for main memory bandwidth. The degradation experienced is dependent on the characteristics of the co-located applications, and ranges from negligible to severe [51]. The problem of co-location interference between applications on multicore processors is fairly new, thus most recent works are still related to measuring or modeling the phenomena, with very few on the actual design of resource management techniques that use such predictive models to mitigate interference effects. The prominence of multicore processors in modern HPC systems, and the large likelihood that the number of cores per processor will continue increasing, has spurred researchers to investigate predictive models and design techniques to mitigate the effects of co-location interference. The research in [70] proposes a general methodology for measuring application interference in multicore processors, and tests it in a real data center. In [71], an empirical approach to predict performance degradation is proposed. Focusing on shared cache and memory bandwidth as the two shared resources that affect performance the most, functions that predict an application’s pressure on those subsystems are found. Both linear models and neural network models are created using empirical data in [52]. A set of benchmarks from both the PARSEC benchmark suite and NAS benchmark suite are co-located on cores of multicore processors in numerous configurations to collect data for correlated features such as last-level cache misses, memory intensities, and execution times. These models allow predictions to be made for performance degradation when multiple applications are co-located on a multicore processor.

To mitigate co-location interference among virtual machines allocated to the same multicore processor, the algorithm in [72] aims to leverage information from applications the system has seen in the past to classify and then schedule applications without violating QoS guarantees. In [73], it is argued that consolidation of several applications onto a compute node (e.g., through virtualization) increases utilization but results in degrading application

performance significantly. Therefore, a technique that increases node utilization while still maintaining SLAs is proposed and tested. A weighting scheme is used to determine the pairings for critical (strict SLA agreements) and non-critical workloads onto compute nodes.

Our study does not focus on modeling co-location interference effects as [51, 70, 71, 52] do. We focus on the design of resource management techniques that use predictive models to avoid interference effects, such as [72, 73]. However, our work also considers power and temperature as constraints in the decision matrix of our resource management techniques.

3.3. SYSTEM MODEL

3.3.1. OVERVIEW. Our model of an HPC system and workload builds on the model proposed in [56]. We assume the facility 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. If the cold aisles are isolated using containment technology (e.g., using plastic curtains or commercial Plexiglass systems), the recirculation of heat among compute nodes in an HPC system may or may not be important to consider. Some facilities have isolated aisles (e.g., the research computing facility at Hewlett Packard in Fort Collins, CO) and some have non-isolated aisles (e.g., the HPC computing facility at Colorado State University). In our work, we compare both and examine the benefits of isolation.

3.3.2. COMPUTE NODES. The *number of compute nodes* in the HPC system is \mathbf{NN} , and each compute node \mathbf{j} belongs to a compute *node type* $\mathbf{NT}(\mathbf{j})$. We assume a heterogeneous system with the *number of compute node-types* equal to \mathbf{NNT} , and compute nodes

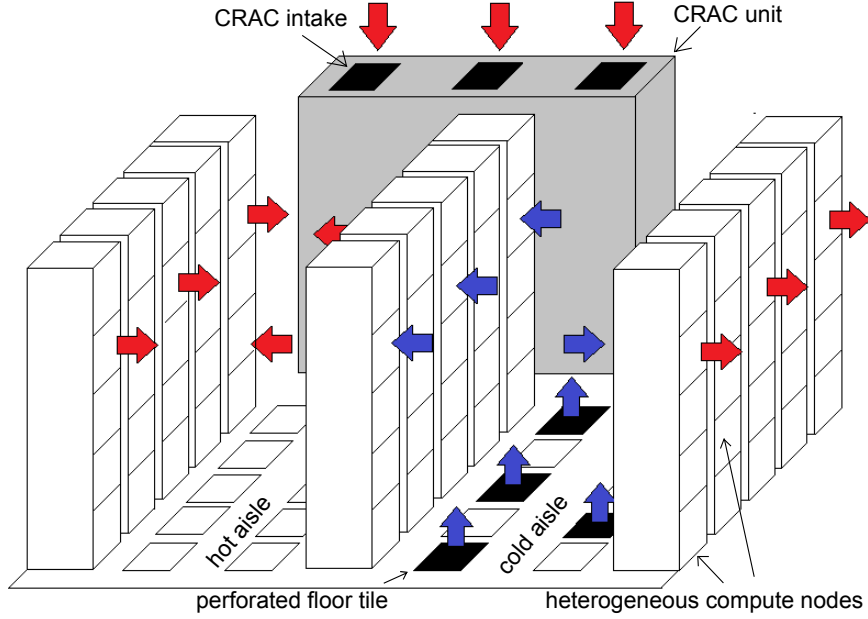


FIGURE 3.1. HPC facility in hot aisle/cold aisle configuration.

that belong to a specific compute node-type 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 be independently assigned performance states (P-states) that provide a tradeoff between power and performance [12]. P-states are discrete voltage and clock frequency pairings, with lower-numbered P-states consuming more power but increasing the execution speed of tasks. We model a case where a core is deactivated by adding one additional P-state to the available P-states of a core, where the deactivated state is the highest-numbered P-state. The total *number of cores* in the HPC system is NC , and $CT(\mathbf{k})$ is the *type of the compute node to which core \mathbf{k} belongs*. If core \mathbf{k} is in node \mathbf{j} , then $CT(\mathbf{k}) = NT(\mathbf{j})$.

3.3.3. WORKLOAD. We assume that we have a set of T known *task types*. The *arrival rate of tasks of type \mathbf{i}* is given by $\lambda_{\mathbf{i}}$. A *reward $\mathbf{r}_{\mathbf{i}}$* is obtained for completing a task of type \mathbf{i} by its individual *deadline $\mathbf{d}_{\mathbf{i}}$* , relative to its arrival time.

The system we consider is heterogeneous, i.e., the power and performance characteristics of the compute node-types are different. Therefore, tasks of the same 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 \mathbf{i} on a core of type \mathbf{j} in P-state \mathbf{k} , $ECS(\mathbf{i}, \mathbf{j}, \mathbf{k})$ (measured in number of tasks per second). In many HPC environments, such as those in national labs (e.g., NCAR, ORNL) and industry (e.g., DigitalGlobe), the tasks executed are from a known set of applications and can therefore be pre-characterized using historical, experimental, or analytical techniques [30–32]; in Section 3.5 we discuss the values used for our evaluation using benchmark data collected from server class machines.

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

$$(15) \quad ER(\mathbf{i}, \mathbf{k}) = DF(\mathbf{i}, \mathbf{k}) \cdot ECS(\mathbf{i}, CT(\mathbf{k}), PS(\mathbf{i}, \mathbf{k})).$$

3.3.4. POWER MODEL. We consider the idle power consumption of a compute node (e.g., from main memory, disks, fans) 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 and the P-state in which the core is executing the task. Let $I(\mathbf{j})$ be the *overhead power consumption* of compute node \mathbf{j} , let $APC(\mathbf{i}, NT(\mathbf{j}), PS(\mathbf{i}, \mathbf{k}))$ be the *average power consumed* by a core \mathbf{k} 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* . We calculate the power consumption of node j , $PN(j)$, as

$$(16) \quad \begin{aligned} PN(j) &= I(j) \\ &+ \sum_{k \in NCN(j)} \sum_{i=1}^T APC(i, NT(j), PS(i, k)) \cdot DF(i, k). \end{aligned}$$

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) of the CRAC unit [74]. Let NCR be the total *number of CRAC units* in the HPC facility, $TC^{in}(i)$ 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 [74]

$$(17) \quad PC(i) = \frac{\rho \cdot C \cdot AFC(i) \cdot (TC^{in}(i) - TC^{out}(i))}{CoP(TC^{out}(i))}.$$

3.3.5. THERMAL MODEL. To calculate the temperatures at compute nodes and CRAC units, we use the concept of thermal influence indices that characterize the causal relationship between heat sources and sinks from [75]. That is, the thermal influence indices are used to help estimate the recirculation of air between compute nodes and CRAC units. We derive the thermal influence indices from computational fluid dynamics (CFD) simulations using the physical layout of the HPC facility we study (see Section 3.5). Let $TN^{in}(j)$ be the *inlet temperature at compute node j* and $TN^{out}(j)$ be the *outlet temperature at compute node j* .

The outlet temperature at compute node \mathbf{j} is a function of the inlet temperature, the power consumed, and the *air flow rate of the node* $\mathbf{AFN}(\mathbf{j})$, calculated as

$$(18) \quad \mathbf{TN}^{out}(\mathbf{j}) = \mathbf{TN}^{in}(\mathbf{j}) + \mathbf{PN}(\mathbf{j}) / (\rho \cdot C \cdot \mathbf{AFN}(\mathbf{j})).$$

Let \mathbf{TC}^{out} and \mathbf{TC}^{in} be the vectors of outlet and inlet temperatures of CRAC units. Also, let \mathbf{A}^{CRAC} be a matrix of thermal influence indices where each element $\mathbf{A}^{CRAC}[\mathbf{i}, \mathbf{y}]$ represents the percentage of heat transferred from CRAC unit \mathbf{i} to CRAC unit or node \mathbf{y} , and let \mathbf{A}^{Node} be a matrix of indices where each element $\mathbf{A}^{Node}[\mathbf{j}, \mathbf{y}]$ represents the percentage of heat transferred from node \mathbf{j} to CRAC unit or node \mathbf{y} . We can then calculate the inlet temperatures of any CRAC unit or node \mathbf{y} as [75]

$$(19) \quad \begin{aligned} \mathbf{T}^{in}(\mathbf{y}) = & \sum_{i=1}^{NCR} \mathbf{A}^{CRAC}[\mathbf{i}, \mathbf{y}] \cdot \mathbf{TC}^{out}(\mathbf{i}) \\ & + \sum_{i=j}^{NN} \mathbf{A}^{Node}[\mathbf{j}, \mathbf{y}] \cdot \mathbf{TN}^{out}(\mathbf{j}) \end{aligned}$$

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

$$(20) \quad \mathbf{T}^{in} \leq \mathbf{T}^{redline}.$$

3.3.6. CO-LOCATION INTERFERENCE. Tasks competing for shared memory in multicore processors can cause severe performance degradation, especially when competing tasks are memory intensive [51]. The memory intensity of a task refers to the ratio of last-level cache misses to the total number of instructions executed [52]. We employ a linear regression model from [52] that combines a set of disparate features (i.e., inputs that are correlated with

application execution time) based on the current applications assigned to a multicore processor to predict the execution time of a target application i on a target core k . A description of the features we use, in addition to the symbols representing each feature, is given in Table 3.1. In a linear model, the output is a linear combination of all features and their calculated

TABLE 3.1. 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
frequency of target core	$C(i, k)$	clock frequency of target core k when running target application i in P-state
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

coefficients, where the coefficients are used to combine the disparate features together into one measure. We classify the task-types into memory intensity classes to calculate these coefficients for the different memory intensity classes using the linear regression model. If we denote u , v , w , x , and y as the coefficients for feature symbols A , B , C , D , and E (from Table 3.1) for a task-type of memory intensity class m on core k , then with z as the constant; the equation for co-located execution time of a task-type i on core k ($CET(i, k)$) is

$$\begin{aligned}
(21) \quad CET(i, k) &= u(m, k) \cdot A(i, k) + v(m, k) \cdot B(i, k) \\
&+ w(m, k) \cdot C(i, k) + x(m, k) \cdot D(k) \\
&+ y(m, k) \cdot E(i, k) + z(m, k)
\end{aligned}$$

The execution rate is the reciprocal of the execution time. Therefore the co-located execution rate for task-type i on core k , $CER(i, k)$, is $1/CET(i, k)$. To assist our resource

management techniques in estimating actual reward rate under the effects of co-location interference, we introduce an objective function called *reward rate*, calculated as

$$(22) \quad \mathbf{RR} = \sum_{i=1}^T \left(r_i \cdot \min \left(\sum_{k=1}^{NC} \mathbf{CER}(i, k), \lambda_i \right) \right).$$

Equation (22) states that the reward rate for a given task-type i is the product of the reward earned and its co-located execution rate. The *min* function in Equation (22) enforces the constraint that if a task is assigned for execution at a faster rate than its arrival rate λ_i , additional reward is not earned.

We also present an objective named *naïve estimated reward rate*, denoted ***NERR***. ***NERR*** uses the execution rate (***ER***) rather than co-located execution rate (***CER***) values, and is therefore naïve to co-located interference effects. ***NERR*** is a false reward rate, however, because co-location interference exists. We use ***NERR*** as an objective in co-location unaware versions of the techniques we propose to demonstrate the value of considering co-location interference effects. ***NERR*** is calculated as

$$(23) \quad \mathbf{NERR} = \sum_{i=1}^T \left(r_i \cdot \min \left(\sum_{k=1}^{NC} \mathbf{ER}(i, k), \lambda_i \right) \right).$$

Reward is only earned when tasks are completed by their deadline. We have no way of guaranteeing deadlines are met in our rate-based 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

$$(24) \quad \frac{1}{\mathbf{ECS}(i, \mathbf{CT}(k), \mathbf{PS}(i, k))} \geq d_i.$$

The goal of this study is to maximize true reward rate when subject to power consumption and thermal constraints, and we believe \mathbf{RR} is a better estimate of reward rate than \mathbf{NERR} because co-location interference exists. The total power consumed by both CRAC units and compute nodes must be less than the power constraint, denoted as ϕ . The thermal constraint is defined in Equation (20). In the next section, we describe several approaches to solve this problem.

3.4. HEURISTICS

3.4.1. NON-LINEAR PROGRAMMING APPROACH. We adapt a power and thermal-aware approach from [56], and improve it to include the effects of co-location by using our \mathbf{RR} objective to estimate the reward rate instead of \mathbf{NERR} . The problem in [56] is formulated as a non-linear program and then solved using approximations, heuristics, and linear programming. The complexity of common interior point algorithms (i.e., the type of algorithms used in many non-linear programming solvers) is $\mathcal{O}(n^{3.5}L)$, where n is the number of variables and L is the input length of the problem [76]. We refer the reader to [56] for the full details of this technique, but give the problem formulation in Appendix E in addition to a brief summary below.

The NLP technique is divided into three steps to solve the formulated mixed integer NLP for maximizing naïve estimated reward rate (using Equation (23)) while obeying the power and thermal constraints. (1) 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. (2) 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. (3) 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. A non-feasible solution can sometimes be returned by the NLP if the power or thermal constraints are set to extreme values. In that case, the local search technique from the GA (Alg. 3) is performed on the resulting solution to set the CRAC outlet temperatures such that the thermal constraint is met (but possibly not the power constraint).

We improve upon this approach in two ways. First, we incorporate the knowledge of the memory intensity of tasks by considering the APC matrix (because 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. The scaling of ECS values for different P-states is based on real data, and therefore is now also a function of memory intensity and clock frequency. Second, we incorporate knowledge of co-location interference in this algorithm by maximizing our **RR** measure, which is a better estimate of reward rate than **NERR**.

3.4.2. **GREEDY HEURISTIC**. We designed a two-phase greedy approach similar in concept to “Min-min” in [13, 37] to assign task-types to cores. For this heuristic, cores are dedicated to a single particular task-type (i.e., 100% of the core’s time is dedicated to tasks of that type). Our greedy heuristic (see Alg. 2) iteratively assigns task-types to cores to find the most efficient mapping, where we define efficiency for a task mapping of type \mathbf{i} on a node of type \mathbf{j} in P-state \mathbf{k} , $EFF(\mathbf{i}, \mathbf{j}, \mathbf{k})$, as

$$(25) \quad EFF(\mathbf{i}, \mathbf{j}, \mathbf{k}) = ECS(\mathbf{i}, \mathbf{j}, \mathbf{k}) / APC(\mathbf{i}, \mathbf{j}, \mathbf{k}).$$

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 $\mathbf{T} \times \mathbf{NNT}$ 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 the heuristic checks if at least one core within the chosen node-type remains unmapped. If so, that core is assigned a 100% desired fraction of time for that task-type (line 6). After making an assignment, that core is removed from consideration (line 7). If the new mapping results in the execution rate of the task-type exceeding its arrival rate (line 8), that core is unmapped (line 9), returned to the pool of available cores (line 10), and that task-type/node-type pair is removed from future consideration (line 11). The heuristic uses the CER value to estimate execution rate (line 8), thus making the decision to provision task-type execution rates as long as the CER value does not exceed the arrival rate. The CRAC outlet temperatures are set to the red-line temperature, and then the outlet temperatures of all CRAC units are iteratively decreased by one degree until the thermal constraints are met (line 12). If no unmapped cores within the selected node-type exist, the current task-type/node-type pair is removed from consideration (line 14). The algorithm repeats using the next pairing until the power constraint is violated, or there are no more task-type/node-type pairings to consider (line 3).

3.4.3. GENETIC ALGORITHM.

3.4.3.1. *Overview.* We also designed a GA based on the Genitor GA [13, 41] to solve our optimization problem. Our GA in this study operates on a population of 200 chromosomes (empirically determined). Each chromosome represents one possible solution, i.e., a complete

Algorithm 2 Pseudo-code for our greedy heuristic

1. get most efficient P-state for each task-type/node-type pair
 2. sort these task-type/node-type pairs by efficiency
 3. **while** power constraint not violated and at least one task-type/node-type pair remains
 4. choose first task-type/node-type pair
 5. **if** unmapped core within selected node-type exists
 6. assign 100% desired fraction of time for selected task-type to a core from selected node-type
 7. remove core from future consideration (mapped)
 8. **if** execution rate of task-type exceeds its arrival rate
 9. unmap core running that task-type
 10. return unmapped core to pool of available cores
 11. remove task-type/node-type pair from future consideration
 12. set CRAC outlet temperatures to hottest temperatures such that thermal constraints are met
 13. **else**
 14. remove task-type/node-type pair from consideration
 15. **end while**
-

resource allocation. Each chromosome consists of a matrix of $T \times NC$ 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. That is, if a core k is spending greater than 100% of the time executing tasks, we normalize the $DF(i, k)$ values on that core. 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%. Normalization is not performed on a core if the value of $SumDF(k)$ is less than 100%.

One chromosome of the initial population is from the greedy heuristic to seed the GA and assist the GA by providing better genetic material than randomly generated chromosomes. After the initial population generation, the chromosomes in the population are evaluated

and ranked by reward rate. 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.

3.4.3.2. *Crossover and Mutation.* Crossover starts by selecting two parent chromosomes using a linear bias [41] and generates two points, \mathbf{x} and \mathbf{y} , such that $\mathbf{x} < \mathbf{y} \leq \mathbf{NC}$. All genes for cores ranging from \mathbf{x} to \mathbf{y} are swapped among parent chromosomes to create two offspring solutions. This operation spans over all task-types for the cores ranging from \mathbf{x} to \mathbf{y} .

Mutation is probabilistically performed on offspring chromosomes to introduce perturbations 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%. Unlike the greedy heuristic that dedicates a core to one task-type (i.e., 100% desired fraction to one task-type), the GA can have cores assigned to execute multiple task-types.

3.4.3.3. *Local Search.* We perform a local search on (possibly mutated) 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. 3. The step described in line 3 is performed by setting the CRAC outlet temperatures to the red-line temperature, and then the outlet temperatures of all CRAC units are iteratively

Algorithm 3 Pseudo-code for our local search technique

1. repeat for $maxIter$ or until power and thermal constraints met
 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 and not all core/task-type combinations in node j are assigned maximum P-state
 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**
-

decreased by one degree until the thermal constraints of the nodes are met. The steps performed in lines 4 to 7 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 hottest node. Second, the CRAC units may be able to run with a higher outlet temperature, reducing the overall power required to maintain red-line temperatures. In our environment, our local search typically ensures that the power and thermal constraints are met, but the resource allocation may have a poor reward rate. If the power and thermal constraints are difficult to meet, there is a limit to how long the search proceeds ($maxIter$) before allowing the GA to continue. We set the value of $maxIter$ to 2,000 in this study. We examine the effects of our resource management techniques under different power and thermal constraint values in Section 3.6.4.

3.5. EVALUATION SETUP

3.5.1. OVERVIEW. In our simulations, we consider two heterogeneous platforms of different sizes. The *small* platform (see Fig. 3.2(a)) consists of one CRAC unit, and 30 compute node cabinets of size 42U with 36 nodes per cabinet (typically some cabinet slots are left empty, e.g., due to power and cooling requirements). This gives a total of 1,080 compute nodes, where each node is one of three node types (see Table 3.2). Error values for the thermal and co-location execution time models can be found in Appendix F, in addition

to values for many of the simulation parameters used in this study. The *large* platform (see Fig. 3.2(b)) consists of two CRAC units, and 120 compute node cabinets with 36 nodes per cabinet (a total of 4,320 compute nodes), where each node is one of the three node types. The power characteristics of our different node-types were obtained from power measurements of three server class machines when executing different PARSEC benchmarks across all P-states (see Table 3.2), and the workload characteristics were also obtained from executing

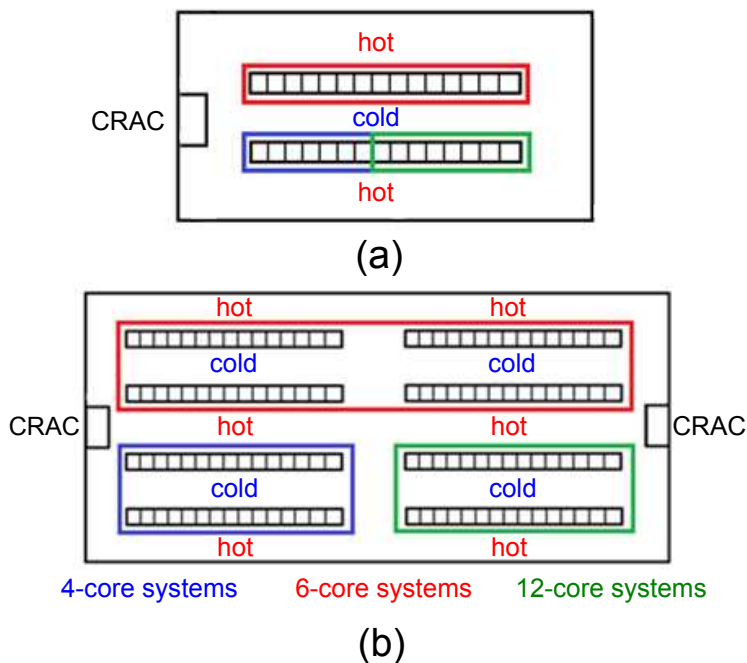


FIGURE 3.2. (a) Small HPC platform configuration, and (b) large HPC platform configuration.

TABLE 3.2. Node-Types Used In Simulations

node-type	Lenovo TS140	HP Z600	HP Z820
processor (Xeon)	E3-1225v3	E5649	E5-2697v2
architecture	22nm	32nm	22nm
number of cores	4	6	12
number of P-states	16	9	16
case fans	2 x 80mm	2 x 92mm, 1 x 80mm	3 x 92mm
air flow rate (m^3/s)	0.0284	0.0519	0.0566

the benchmarks on the node-types listed in Table 3.2. The power constraint was set to 900 kW for the large platform, and 230 kW for the small platform. The red-line temperature was set to 30°C, which is on the high end of ASHRAE’s temperature guidelines [50].

3.5.2. WORKLOAD. The task-types are from the PARSEC benchmark suite, and corresponding ECS matrix entries for each benchmark on each node-type in each P-state are obtained from taking the reciprocals of the execution times of those benchmarks on each of the machines and P-states. We chose to represent the workload in this study based on the PARSEC benchmark suite because of the diverse set of applications it offers, including scientific applications for HPC systems. For example, benchmarks we consider are associated with fluid dynamics, option pricing, body tracking of a person, simulated annealing optimization, and pricing a portfolio of swap options. In all, the following benchmarks were used: *canneal*, *cg*, *ua*, *sp*, *lu*, *fluidanimate*, *blackscholes*, *bodytrack*, *ep*, and *swaptions*. With regards to their memory intensity class, *canneal*, *cg*, and *ua* were classified as “heavy,” *sp*, *lu*, and *fluidanimate* were classified as “medium,” *blackscholes* and *bodytrack* were classified as “light,” and *ep* and *swaptions* were classified as “very-light.”

We split the benchmarks into three workload environments based on their memory intensities to be representative of three different types of real-world computing environments. The *memory intensive* workload consists of *canneal*, *cg*, *ua*, *sp*, and *lu* to represent a database style workload. The *hybrid* workload includes *ua*, *sp*, *lu*, *fluidanimate*, and *blackscholes* to form an all-purpose group of both CPU and memory intensive applications. Finally, the *CPU intensive* environment is formed of *fluidanimate*, *blackscholes*, *bodytrack*, *ep*, and *swaptions* to emulate scientific computing.

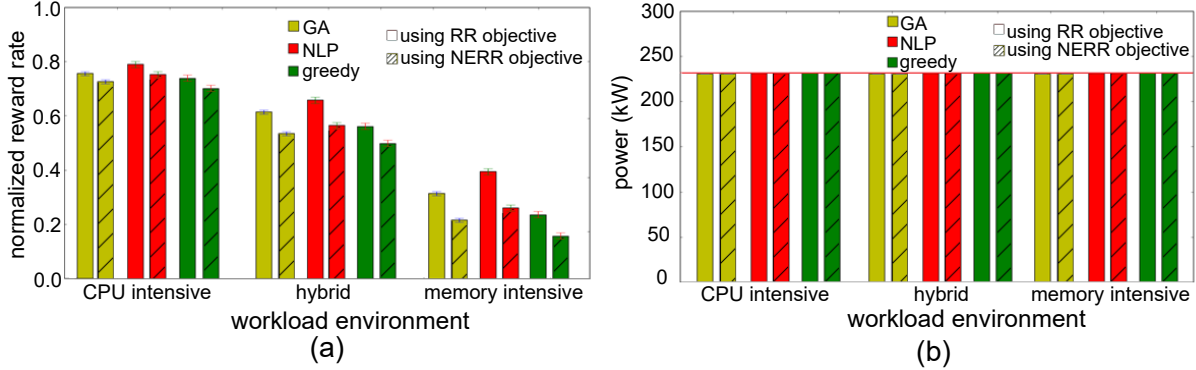


FIGURE 3.3. (a) Reward rate comparison between different workload environments on the 1,080 nodes system, and (b) power consumption comparison in relation to power budget constraint (red-line). Solid bars represent the GA, NLP, or greedy technique when using **RR** as the objective (co-location aware), and the hashed bars represent those techniques when using **NERR** as the objective (co-location unaware).

3.5.3. CRAC UNITS. In this study, we assume that the CRAC units are homogeneous. The CoP for a CRAC unit is a function of its outlet temperature, τ , given by $CoP(\tau) = 0.0068\tau^2 + 0.0008\tau + 0.458$ [74]. The air flow rate of each CRAC unit is set to $26.1 \text{ m}^3/\text{s}$, which is the air flow rate of a CRAC unit with capacity to cool approximately 350 kW [77].

3.6. RESULTS

3.6.1. OVERVIEW. The primary contribution of this research is to provide extensive analyses of our co-location interference and thermal-aware techniques for maximizing reward under a range of physical HPC facility configurations and workload environments. Large-scale computing systems fall into several different categories that execute a diverse set of workloads, making it important to simulate a variety of environments so a system administrator can choose resource management techniques that are most applicable to a specific environment. For example, it is common in supercomputing environments to run scientific applications that are compute intensive, with less co-location interference effects and

therefore less of a need to consider them in resource management. However, in a data processing environment the applications access memory often (memory intensive), and ignoring co-location interference effects can cause severe performance issues. The analyses we present are also useful for physical HPC facility design (compute node placement optimization), where researchers can determine potential hotspots and optimal use of space for node and CRAC placement with an “average” workload.

The bar graphs discussed in the rest of this section represent the averages of 48 trials, with each trial varying in the deadline, reward, and arrival rate values of the task-types. The ECS and power values are obtained from experimental data of the PARSEC benchmarks (see Section 3.5.2) on three heterogeneous server class machines. The error bars are the 95% confidence intervals around the mean of those 48 trials.

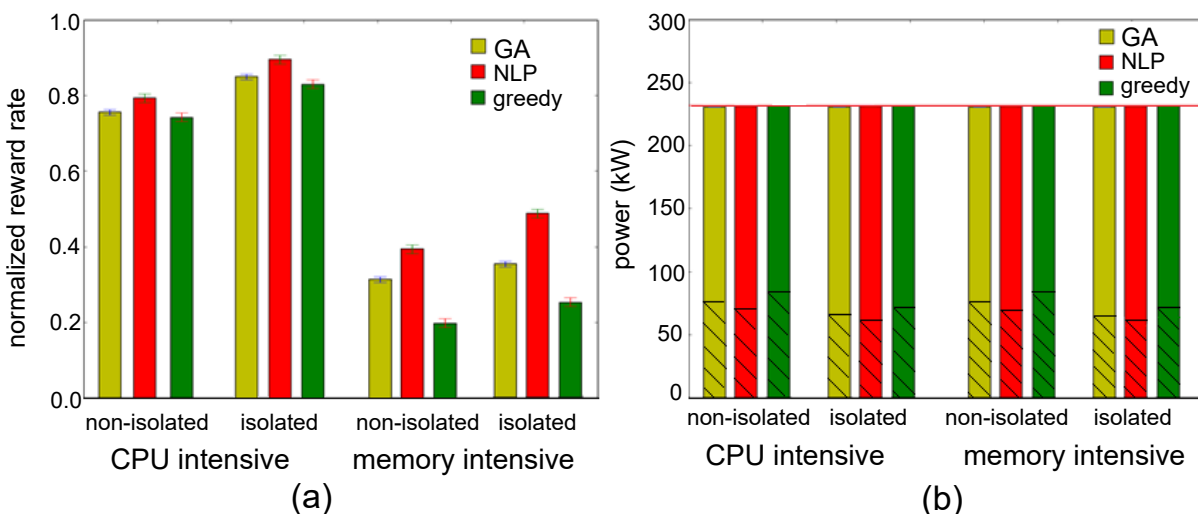


FIGURE 3.4. (a) Reward rate comparison between isolated and non-isolated cold aisle configurations on the 1,080 nodes system, and (b) power consumption comparison in relation to power budget constraint (red-line). Hashed portion of bars show power consumed by the CRAC unit, and solid portion of bars show the power consumed by the compute nodes.

3.6.2. **WORKLOAD EXPERIMENTS.** In our first experiment, we examine the effects that the different workload environments have on the performance of our resource management techniques. Fig. 3.3 shows a comparison of the greedy, GA, and NLP techniques across the three different workload environments: *CPU intensive*, *hybrid*, and *memory intensive*. In this figure, we also compare each of the co-location aware techniques (non-hashed bars) with versions that are co-location unaware (hashed bars). That is, we compare the GA and NLP techniques when using either the naïve estimated reward rate (***NERR***) or reward rate (***RR***) as their objective functions, and greedy uses either the ***ER*** values or ***CER*** values to estimate execution rates (line 8). The NLP technique that uses ***RR*** as its objective (co-location unaware) is similar to [56]. The reward rate is normalized by the total reward rate that the system is capable of obtaining, i.e., when all task-types are allocated to execute at rates equal to their respective arrival rates. To make a fair comparison between the NLP and GA techniques, the GA was terminated at the same time as it took the NLP to finish (approximately four hours). All techniques were able to meet the constraints.

We can see in Fig. 3.3(a) that for all workload types, the NLP technique works best, GA is second best, and greedy last. We can also see, within the same power budget, that all techniques earn significantly less reward as the memory intensity of the workload increases (moving from CPU intensive to hybrid, and hybrid to memory intensive). Intuitively, this makes sense as the co-location interference effects become exacerbated as the memory intensity increases. The NLP technique achieves approximately half of the reward on the memory intensive workload environment than on the CPU intensive environment. Comparing the co-location interference aware techniques with those that are unaware, we can see the benefits of considering co-location interference, and those benefits become more significant as the memory intensity of the workload increases.

Another observation to note is the difference in performance of algorithms within a given workload type. For example, we can see in Fig. 3.3(a) that the dropoff in performance between the NLP and greedy techniques is fairly insignificant for the CPU intensive workload, at least in comparison to the memory intensive workload. Depending on the requirements of the system administrator and the mapping event interval (epoch length), it may be more worthwhile to employ a simple greedy heuristic when the workload environment experiences little co-location interference because it is much faster to make decisions than the GA and NLP. The performance difference between the NLP, GA, and greedy heuristics become more pronounced and extremely significant when executing the memory intensive workload environment, with the greedy performing half as well as the NLP in that case.

3.6.3. ISOLATION EXPERIMENTS. The isolation of the cold-aisles in an HPC facility provides many benefits at a low upfront cost to purchase the isolation system. Isolating cold aisles can be done using several different methods (e.g., hanging plastic curtains, installing custom Plexiglass walls and ceiling), all with the same goal of minimizing the heat recirculation among nodes. Isolating the cold aisles has previously been shown to have the intuitive result of significantly reducing node inlet temperatures when CRACs are set to the same outlet temperatures or higher [78]. In our thermal-aware resource management problem, reducing the inlet temperatures of the compute nodes allows the CRAC units to run at higher temperatures, reducing the cooling power required and allowing more power budget to be allocated to the compute nodes for executing tasks.

To quantify the impact of isolation, we experiment with our resource management techniques after calculating an additional set of the thermal influence indices (i.e., the $\mathbf{A}^{\text{CRAC}}[\mathbf{i}, \mathbf{y}]$ and $\mathbf{A}^{\text{Node}}[\mathbf{j}, \mathbf{y}]$ values). These coefficients were calculated by placing isolation curtains around the cold aisles in the mesh of the computational fluid dynamic (CFD) simulations.

Fig. 3.4 shows a comparison between using the isolated and non-isolated aisles for the CPU intensive and memory intensive workload environments. In Fig. 3.4(a), the normalized reward rate is shown, and in Fig. 3.4(b), the power consumption is shown, with hashed portion of the bars showing the power consumed by the CRAC unit and the solid portion of the bars showing the power consumed by the compute nodes.

In Fig. 3.4(a), we observe that the reward earned is more in an isolated configuration for all techniques, following similar trends as the non-isolated configuration. Fig. 3.4(b) gives the reason why the isolated configuration can earn more reward: under the same power constraint, the techniques when using the isolated configuration are able to use less power for cooling (hashed portion of the bars) and more power for computing (solid portion of the bars). If the reward earned is analogous to some form of revenue (e.g., cloud computing), the initial cost of isolating the aisles could be worth the price over time.

3.6.4. POWER AND THERMAL CONSTRAINT SENSITIVITY ANALYSIS. Our third set of experiments examines the effects of varying the power and thermal constraints on the reward that can be earned in both isolated and non-isolated configurations. The values of both of these constraints have a significant effect on the ratio of power consumed by cooling and computing, and the reward that can be earned by the system. Such studies are valuable to the system administrator or HPC facility owner to answer “what-if” questions, such as “what-if” the system is allowed 20% more power, or “what-if” the maximum allowable (red-line) temperature is reduced to increase reliability?

The results of our temperature and power constraint sensitivity analysis are shown in Figs. 3.5 and 3.6. Figure 3.5 shows the reward rate and associated total power consumption results of our resource management techniques when varying the thermal constraint (red-line temperature) from 24°C to 32°C with the power constraint set to 230 kW. Figure 3.6 shows

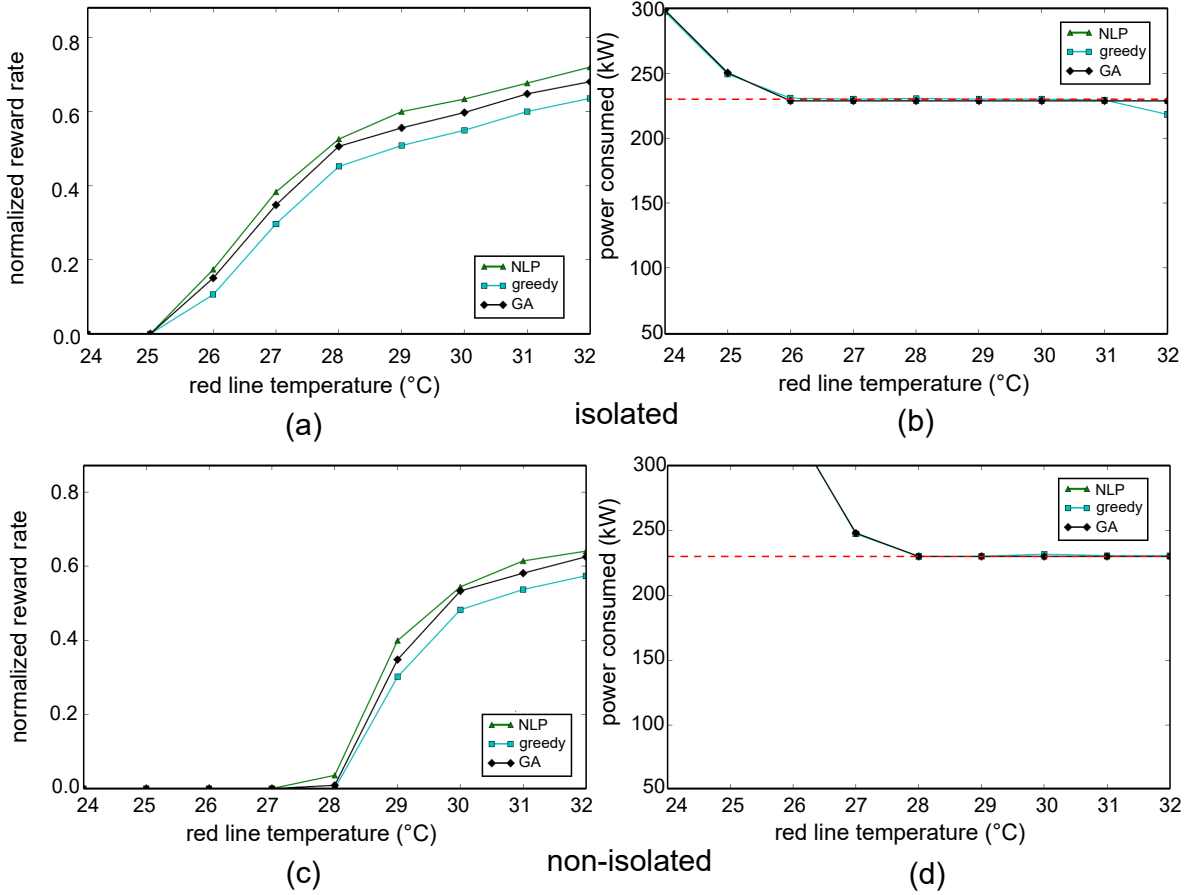


FIGURE 3.5. Sensitivity analysis of our greedy, GA, and NLP resource management techniques for both isolated and non-isolated HPC facility configurations using the 1,080 node system on the thermal (red-line) constraint.

the reward rate and total power consumption of our resource management techniques when varying the power budget constraint from 200 kW to 300 kW in 10 kW increments with the thermal constraint set to 30°C.

Note in Fig. 3.5 that if the red-line temperature becomes difficult to meet (less than 26°C for the isolated configuration and less than 28°C for the non-isolated configuration), none of the techniques are able to meet the power constraint. This is due to the thermal-aware aspects of the techniques, where the CRAC outlet temperatures are set such that the thermal constraints are met. When the CRAC outlet temperatures are set low for the compute node temperatures to be under red-line, the power consumption of the CRAC units becomes very

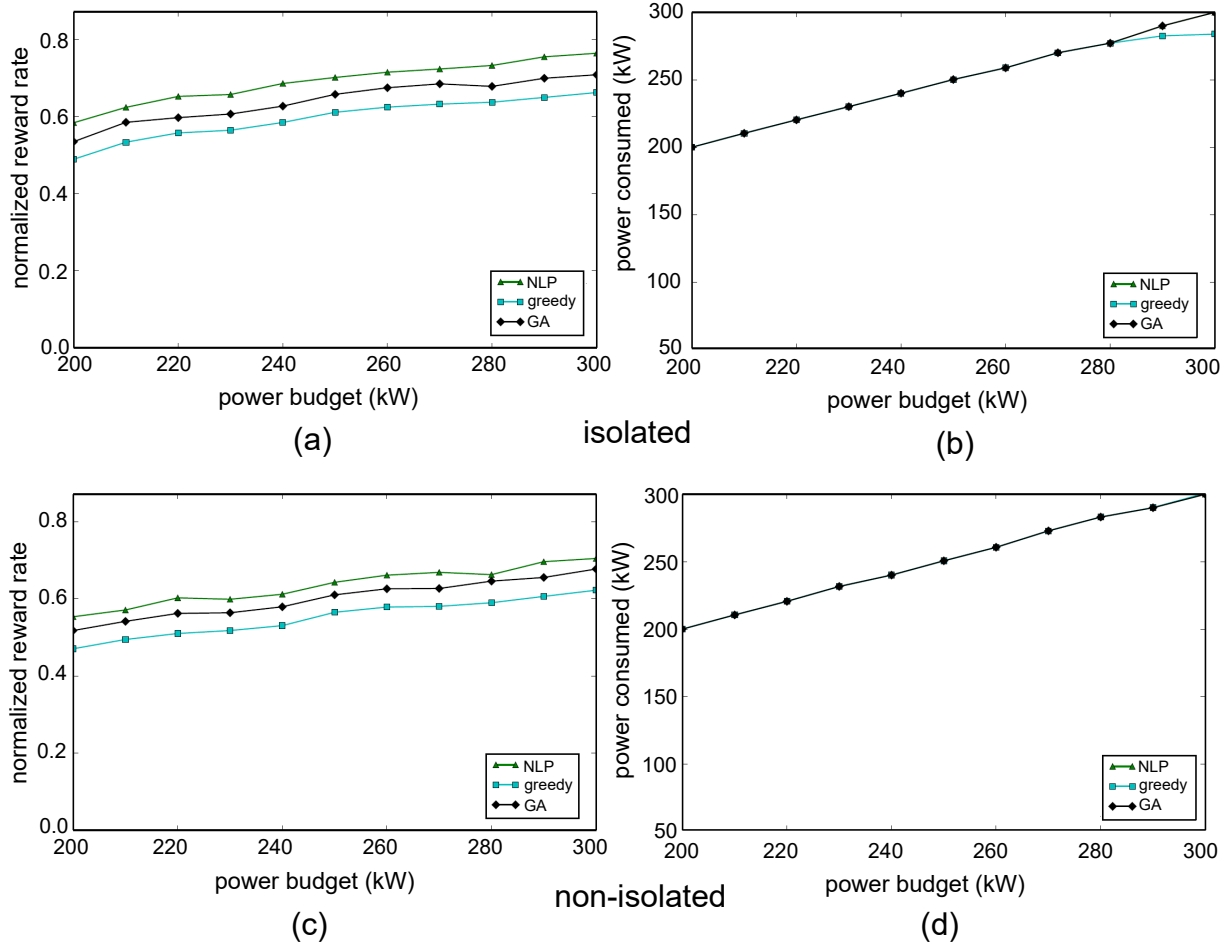


FIGURE 3.6. Sensitivity analysis of our greedy, GA, and NLP resource management techniques for both isolated and non-isolated HPC facility configurations using the 1,080 node system on the power constraint.

high (see Equation (17)), and the system is unable to meet the power budget even with most (if not all) of the compute cores deactivated (either assigned zero desired fraction of time, or running in the highest-numbered P-state). With all cores deactivated, a compute node still consumes idle power ($I(j)$) and would still require cooling. Also note that the amount of reward earned decreases non-linearly with the red-line temperature threshold. This makes sense, because when the CRAC units have to run at a lower temperature to ensure that red-line temperatures are not violated, more of the power budget must be used for cooling. The relationship is non-linear most likely because of the non-linearity of the CoP (see Section

3.5.3). The non-isolated configuration shows the techniques earning less reward, and also violating the power constraint at lower red-line temperature values, but overall similar trends as the isolated configuration. However, in the isolated configuration we see that the greedy technique does not use all available power when the red-line threshold is set to 32°C. This is because assigning all core/task-type combinations to execute in their most power-efficient P-state, in combination with the small amount of cooling power required to maintain a 32°C threshold, does not use all of the allotted power budget. The GA and NLP are able to assign core/task-types to execute in faster P-states to consume that additional power.

Fig. 3.6 shows that all techniques are effective at using all of the power budget and not violating the power budget. The reward earned varies almost linearly with the power budget, and the relationship between power budget and reward rate is not as exaggerated as in Fig. 3.5. Giving the system a greater power budget to operate within does result in more reward rate earned, because nodes can operate in faster P-states and execute more tasks. However, as evidenced by Fig. 3.5, it may be more worthwhile to operate the compute nodes and CRAC units at a slightly higher temperature to realize significant gains in reward rate earned. If reward rate is analogous to a revenue generated for computing, analyses such as these would give important insights into the most profitable operating point.

3.6.5. SCALABILITY ANALYSIS. As HPC facilities become larger, it is important to address the practicality of resource allocation techniques on larger system complexities. Figs. 3.3 and 3.4 showed that NLP was able to achieve the best results for a 1,080 node platform size across three different workload environments and both non-isolated and isolated cold-aisle configurations. However, the primary drawback associated with the NLP technique is its poor scalability. GAs 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. 3.7(b) shows a comparison of greedy and GA on the large (4,320 node) platform. We recorded the performance of GA at many different heuristic execution times to examine the benefits of running it for different time intervals. The NLP is excluded, as it was unable to finish within two weeks.

In summary, for a small platform and problem size, the better results obtained using the NLP approach motivates its use to generate resource allocations that optimize reward rate. However, as the platform size becomes larger, the NLP technique becomes intractable, and GA offers a solution in a reasonable amount of time. In the next subsection, we discuss epoch time interval considerations and how the interval impacts the selection of resource management technique.

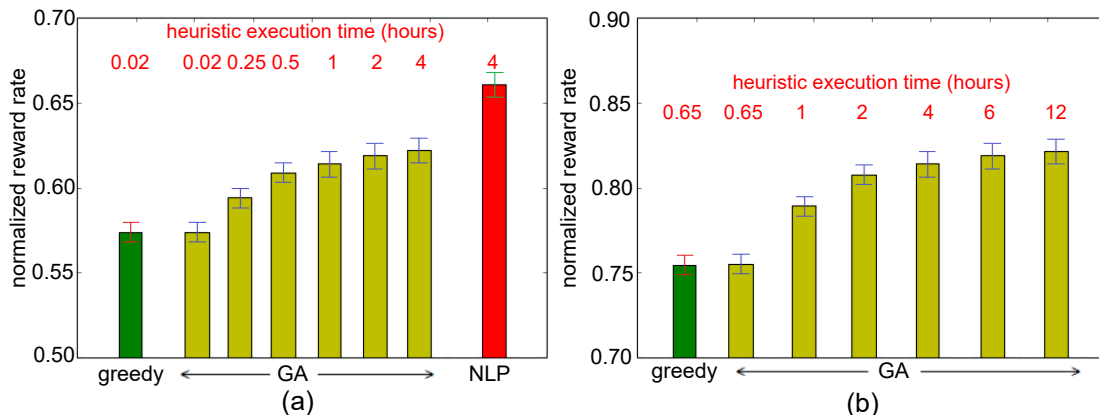


FIGURE 3.7. Comparison of reward rate on the (a) small (1,080 node) platform, and (b) large (4,320 node) platform. NLP excluded in (b) because it was unable to finish within two weeks.

3.6.6. EPOCH SIZE INTERVAL CONSIDERATIONS. The length of the epoch interval has a large impact on the choice of a resource management technique. For an HPC system that has long running and predictable jobs (e.g., supercomputing environments dedicated to climate analysis or molecular biology simulations), the task arrival rates would not change

significantly in short periods of time and thus not frequently require new resource allocations. Setting a longer epoch interval time and performing resource management using the longer running techniques (GA or NLP) lends itself to such environments. In an environment where the types or arrival rates of the workload changes rapidly, resource allocation decisions would have to be fast, motivating the use of the greedy technique. That is, if a resource allocation needs to be found for a short epoch interval time, our greedy heuristic may be the desired approach to take. Over time, however, our GA and NLP heuristics are able to provide better solutions, and the GA can be terminated at any point, e.g., at the start of a new epoch when new execution rates and P-states need to be found.

3.7. CONCLUSIONS

We study the problem of maximizing the reward collected for completing tasks by their deadlines subject to power and thermal constraints for heterogeneous high-performance computing systems. Co-location interference can have a significant impact on the execution speeds of tasks (and thus total reward). We capture these effects with our more accurate performance metric that considers co-location. We analyze several resource management techniques: a greedy technique based on assigning tasks to machines in order of their efficiency (most performance per unit of power), a GA combined with a local search technique that maximizes the reward rate and ensures the power and thermal constraints are met, and an NLP technique building on our prior work to maximize reward rate instead of naïve estimated reward rate.

The primary contributions of this research were to provide in-depth analyses of the problems and solutions associated with co-location and thermal-aware resource management in heterogeneous computing systems. We use a new thermal model that directly considers

CRAC units in its calculation of the thermal influence coefficients, and a new co-location interference model created from a linear regression technique using data from our lab servers. Because the amount of co-location interference can vary greatly between workloads, we classify task-types into three different workload environments of varying memory intensities. We found that our greedy technique can perform almost as well as the more complex GA and NLP techniques when interference between applications is small, but the complex techniques become significantly better at heavy interference.

Isolation of the cold-aisles has a significant impact on the thermal profile of the HPC facility. We quantify the usefulness of isolation by comparing the reward rate our resource management techniques are able to earn in both configurations. We then show a power and thermal constraint sensitivity analysis to answer some “what-if” questions that would help system administrators and facility operators in deciding what power budget or thermal constraints would be ideal. Possible directions for future studies in this area are presented in Chapter 6.

CHAPTER 4

ENERGY COST OPTIMIZATION FOR GEOGRAPHICALLY DISTRIBUTED HETEROGENEOUS DATA CENTERS[‡]

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 [80]). Distributing data centers geographically offers benefits to the clients (e.g., low latency due to shorter communication distances). However, a strong motivating factor for data center operators to geographically distribute their data centers is to reduce operating expenditures by exploiting time-of-use (TOU) electricity pricing [81], and reducing electricity costs is now a focus of data center management.

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 nodes to run in slower but possibly 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 on-site renewable power can provide additional opportunities for geographical load distribution (GLD) techniques to reduce electricity costs.

[‡]This work was performed jointly with student Eric Jonardi, and was presented at the *Energy-efficient Networks of Computers* workshop (E2NC '15) [79]. The other co-authors of this work are: Sudeep Pasricha, Anthony A. Maciejewski, and H. J. Siegel. This research was supported by NSF grants CNS-0905399, CCF-1302693, and CCF-1252500. We thank Hewlett Packard for donating servers for this work.

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. This work applies to environments where there is information about the history of the types of tasks being executed (e.g., DigitalGlobe, Google, DoD). By considering TOU pricing and renewable power models at each data center, we design three new workload management techniques that assume varying degrees of co-location interference knowledge to distribute or migrate the workload to low-cost data centers at regular time intervals, while ensuring all of the workload completes. We compare to the state-of-the-art method [1], and show that our best heuristic can, on average, achieve a cost reduction of 37% comparatively. The contributions of this work are as follows:

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

4.2. RELATED WORK

Workload distribution for geo-distributed data centers has been studied in [82–87]. Knowledge of TOU pricing is typically used to either minimize electricity costs across all geo-distributed data centers (e.g., [82, 88, 84–87]), or to maximize profits when a revenue model

is included for computing (e.g., [83]). A quality of service (QoS) constraint of some form is recognized in most of the aforementioned works, typically as a queuing delay constraint [88, 84, 86]. Others incorporate QoS violations into the cost function, where a monetary penalty is associated with violating queuing delay [83], latency [85], or migration [82] service level agreements (SLAs). The modeling detail varies significantly, some works include dynamic voltage and frequency scaling (DVFS) in decision making [84], some include power consumption of the cooling system in addition to the computing system [85, 86], others consider real-world TOU pricing data [83, 84], and one considers renewable energy sources at each data center location [82]. 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 in GLD, we consider co-location interference as part of our load distribution techniques; 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 [82], 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 different data centers. We differ significantly from [82] by including TOU electricity pricing traces, consideration of DVFS P-state decisions in our management techniques, and integrating interference caused by the co-location of multiple tasks to cores that share resources.

4.3. SYSTEM MODEL

4.3.1. **GEO-DISTRIBUTED LEVEL.** The goal of the geo-distributed resource manager (GDRM) is to minimize the total monetary cost of the system while servicing all requests. We divide time evenly into intervals called *epochs*. As an example, in this work an epoch is an hour of time (T^e), thus a 24-epoch period is a full day.

We assume that the beginning of each epoch is a steady-state scheduling problem where we assign *execution rates* of a set of I 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 meet or exceed their arrival rates. For each epoch τ , we assign a desired data center execution rate $ER_{d,i}^{DC}$ for each task type i to each data center d such that the total execution rate for all task types exceed (or equal) the corresponding arrival rate, AR_i , thus ensuring the workload is completed. That is,

$$(26) \quad \sum_{d=1}^D ER_{d,i}^{DC}(\tau) \geq AR_i(\tau), \quad \forall i \in I.$$

4.3.2. DATA CENTER LEVEL.

4.3.2.1. *Overview.* Each data center d houses NN_d compute nodes that are arranged in hot aisle/cold aisle fashion (Fig. 4.1), and a cooling system comprised of NCR_d computer room air conditioning (CRAC) units. A compute node n is of a heterogeneous compute node type, where node types vary in their execution speeds, power consumption characteristics,

and number of cores, i.e., they are heterogeneous. 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 \mathbf{n} is NCN_n , and NT_k is the compute node type to which core \mathbf{k} belongs.

4.3.2.2. *Core Execution Rates.* At each data center \mathbf{d} , the sum of execution rates of all cores that are assigned to execute task type \mathbf{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 \mathbf{i} on a core of node type \mathbf{n} in P-state \mathbf{p} , $ECS(\mathbf{i}, \mathbf{n}, \mathbf{p})$.

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

$$(27) \quad ER_{i,k}^{core}(\tau) = DF_{i,k}(\tau) \cdot ECS(\mathbf{i}, NT_k, PS_{i,k}(\tau)).$$

At the data center level, we assign $DF_{i,k}(\tau)$ and $PS_{i,k}(\tau)$ such that power is minimized (see Section 4.3.2.3), and the execution rates of all task types on cores in data center \mathbf{d} meets or exceeds the execution rate assigned by the GDRM, ensuring that the arriving workload is fully executed. That is,

$$(28) \quad \sum_{n=1}^{NN_d} \sum_{k=1}^{NCN_n} ER_{i,k}^{core} \geq ER_{i,d}^{DC} \quad \forall \mathbf{i} \in I, \forall \mathbf{d} \in D.$$

4.3.2.3. *Power Model.* The power consumption of a compute node consists of the overhead (“idle”) power consumption and dynamic power consumed by cores executing tasks. We define O_n as the overhead power consumption of compute node \mathbf{n} . Let $APC(\mathbf{i}, NT_k,$

$PS_{i,k}(\tau)$) 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

$$(29) \quad PN_n(\tau) = O_n + \sum_{k=1}^{NCN_n} \sum_{i=1}^I APC(i, NT_k, PS_{i,k}(\tau)) \cdot DF_{i,k}(\tau).$$

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 [74], calculated using Eq. 5 of [4].

4.3.2.4. *Renewable Energy Model.* Solar energy E_d^{solar} and wind energy E_d^{wind} (both kWh) are calculated for each data center d as an average per epoch τ (Eqs. 30 and 31 are from [89]). 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, given by

$$(30) \quad E_d^{solar}(\tau) = \alpha \cdot A_d^{solar} \cdot s_d(\tau) \cdot T^e,$$

$$(31) \quad E_d^{wind}(\tau) = \beta \cdot \frac{1}{2} \cdot A_d^{wind} \cdot \rho_d(\tau) \cdot v_d(\tau)^3 \cdot T^e,$$

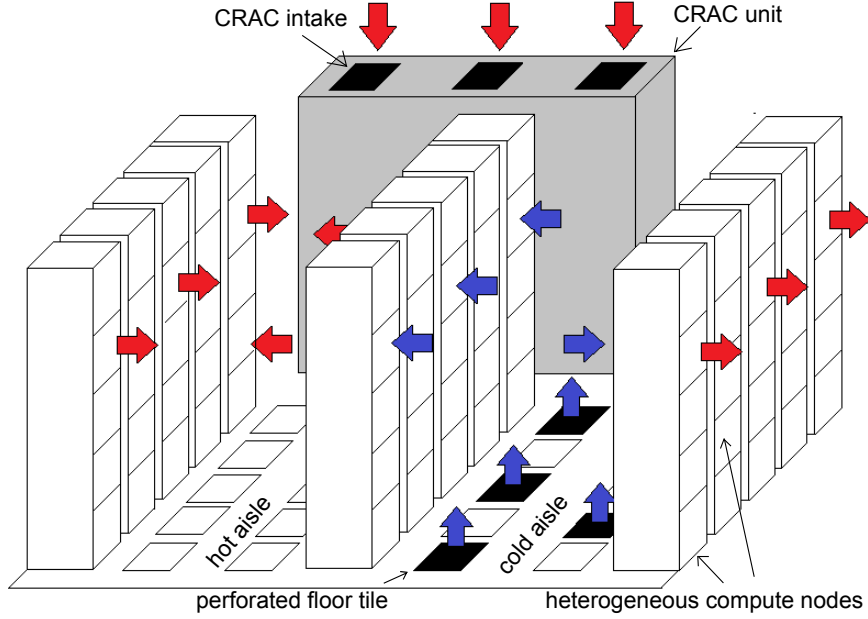


FIGURE 4.1. Data center in hot aisle/cold aisle configuration [4].

$$(32) \quad \mathbf{R}_d(\tau) = \mathbf{E}_d^{solar}(\tau) + \mathbf{E}_d^{wind}(\tau).$$

4.3.2.5. *Thermal Model.* Using the notion of thermal influence indices [75] that were derived using computational fluid dynamics simulations, we can calculate the steady-state temperatures at compute nodes and CRAC units in each data center. Because we assume the same physical layout for each of the data centers (Fig. 4.1), 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 [4]. 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).

4.3.2.6. *System Electricity Cost.* The electricity price at data center \mathbf{d} during epoch τ is defined as $E_d^{price}(\tau)$. Let Eff_d be the approximation of power overhead in data center \mathbf{d} due to the inefficiencies of power supply units and uninterruptable power supplies. The total electricity cost for data center \mathbf{d} during epoch τ , $PC_d(\tau)$, is defined as

$$(33) \quad PC_d(\tau) = E_d^{price}(\tau) \cdot \left[\left(\sum_{c=1}^{NCR_d} PCR_{d,c}(\tau) + \sum_{n=1}^{NN_d} PN_n(\tau) \right) \cdot Eff_d - R_d(\tau) \right].$$

4.3.2.7. *Node Activation/Deactivation Cost.* At each data center, the number of nodes of each node type that are in use changes frequently between epochs. Inactive nodes are placed in a sleep state, but entering and exiting this sleep state takes a non-negligible amount of time. Each node that is active is considered to be active for the entire epoch, which requires that any node transitioning to/from a sleep state do so during the epoch following/prior the current epoch, respectively.

For each data center \mathbf{d} , let $N_{d,j}^{start}(\tau)$ be the number of nodes of type \mathbf{j} that are inactive during epoch τ and active during epoch $\tau + 1$, and let $N_{d,j}^{stop}(\tau)$ be the number of nodes of type \mathbf{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 node type \mathbf{j} , respectively, with the average CPU utilization of node type \mathbf{j} defined as ϕ_j^E . Let the coefficient to approximate CRAC unit power at data center \mathbf{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. Let \mathbf{J}_d be the set of node types in data center \mathbf{d} . Let T^S be the time

required for a node to transition to/from a sleep state. Recall that T^e is the duration of an epoch. The node assignment cost AC_d for data center d during epoch τ is calculated as

$$(34) \quad AC_d(\tau) = \sum_{j \in J_d} E_d^{price}(\tau) \cdot Eff_d \cdot \left(1 + \frac{1}{CUP_d}\right) \cdot \frac{T^S}{T^e} \\ \cdot \left(\phi_j^E P_j^D + P_j^S - P_j^{Sleep}\right) \cdot \left(N_{d,j}^{start}(\tau) + N_{d,j}^{stop}(\tau)\right).$$

4.3.2.8. *Co-Location Interference Model.* Tasks competing for shared memory in multi-core processors can cause severe performance degradation, especially when competing tasks are memory-intensive [51]. The memory-intensity of a task refers to the ratio of last-level cache misses to the total number of instructions executed [52]. We adapt a linear regression model from [52] 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 . These features are $A_{i,k}$, the number of applications co-located on that multicore processor, $B_{i,k}$, the base execution time, $C_{i,k}$, the clock frequency, D_k , the average memory intensity of all applications on that multicore processor, and $E_{i,k}$, the memory intensity of application i on core k .

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 on each of the node types, and calculate the coefficients for each memory-intensity class using the linear regression model. If we denote \mathbf{u} , \mathbf{v} , \mathbf{w} , \mathbf{x} , and \mathbf{y} as the linear model coefficients for feature symbols \mathbf{A} , \mathbf{B} , \mathbf{C} , \mathbf{D} , and \mathbf{E} , respectively, plus the constant term \mathbf{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

$$\begin{aligned}
(35) \quad CET_{i,k}(\tau) &= \mathbf{u}_{m,k} \cdot \mathbf{A}_{i,k} + \mathbf{v}_{m,k} \cdot \mathbf{B}_{i,k} + \mathbf{w}_{m,k} \cdot \mathbf{C}_{i,k} \\
&\quad + \mathbf{x}_{m,k} \cdot \mathbf{D}_k + \mathbf{y}_{m,k} \cdot \mathbf{E}_{i,k} + \mathbf{z}_{m,k} \cdot
\end{aligned}$$

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

$$(36) \quad CER_i(\tau) = \sum_{d=1}^D \sum_{k=1}^{NC_d} CER_{i,k}^{core}(\tau).$$

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

$$(37) \quad \sum_{k=1}^{NC_d} CER_{i,k}^{core}(\tau) \geq ER_{d,i}^{DC}(\tau), \quad \forall i \in I, \forall d \in D.$$

4.4. HEURISTIC DESCRIPTIONS

4.4.1. **PROBLEM STATEMENT.** The GDRM allocates the incoming workload to specific nodes within each data center. The GLD problem is NP-hard [1], and therefore we propose three heuristics for GDRM (FDLD-TAO, FDLD-CL, and GALD-CL), with each having different levels of detail of the system available to it. The system as a whole is under-subscribed, i.e., all tasks must be completed without dropping. The objective of a GDRM is to minimize monetary electricity cost of the geo-distributed system (the sum of Eq. 33 across all data centers) while ensuring the workload is completed (Eqs. 26 and 37).

4.4.2. **FORCE DIRECTED LOAD DISTRIBUTION HEURISTICS.** Force-directed load distribution (FDLD) is a variation of force-directed scheduling [90]. We adapt the FDLD proposed in [1],

designated FDL-D-SO, to our rate-based allocation environment, and propose two new FDL-D heuristics (FDL-D-TAO and FDL-D-CL) that each possess different amounts of co-location interference information to solve this problem.

In FDL-D-SO, to account for co-location interference performance degradation and let the FDL-D technique meet the execution rate constraint at a given data center (Eq. 37), we give the technique simple over-provisioning (FDL-D-SO) 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 FDL-D-TAO technique improves upon FDL-D-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 FDL-D-SO and FDL-D-TAO, the degree of over-provisioning (ϕ^C and ϕ_i^C , respectively) is determined empirically. Lastly, the FDL-D-CL heuristic uses the co-location models given in Sec. 4.3.2.8 to account for co-location effects when calculating task execution rates. All versions of FDL-D consider a system implementation where the computing time of each core in a node is evenly divided among its assigned tasks.

The fundamental operation of all FDL-D variants is described in Algorithm 4. To generate the initial solution, every node in every data center in every epoch is assigned to execute all task types (step 1). Each iteration of the FDL-D removes one instance of one task type from a single node, selecting the task to remove that would result in the lowest total system force, \mathbf{F}^S (steps 3-20). \mathbf{F}^S is the sum of the execution rate forces ($\mathbf{F}^{ER}(\tau)$) and cost forces ($\mathbf{F}^C(\tau)$) across all epochs.

The execution rate force \mathbf{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 node the task is executing on, but the FDL-D 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 node utilization factor ϕ_j^E for each node 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 node of type j in the highest numbered P-state and lowest numbered P-state, respectively. Therefore, the equivalent single core execution rate $R_{j,i}$ of task type i on node type j is

$$(38) \quad R_{j,i} = ER_{j,i}(P_{MAX}) + [ER_{j,i}(P_0) - ER_{j,i}(P_{MAX})] \phi_j^E.$$

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

$$(39) \quad Q_{d,j,i}(\tau) = \sum_{m=1}^{N_{d,j}} \begin{cases} \frac{1}{|W_{d,j,m}(\tau)|} & \text{if } i \in W_{d,j,m}(\tau) \\ 0 & \text{else} \end{cases}$$

Let K_j be the number of cores in a node of type j . The average estimated execution rate $ER_{j,i}^E(\tau)$ of task type i on machine type j during epoch τ , when using either the FDL-D-SO or FDL-D-TAO versions, is given by

$$(40) \quad ER_{j,i}^E(\tau) = \sum_{d=1}^D \sum_{j \in J_d} K_j \cdot R_{j,i} \cdot F \cdot Q_{d,j,i}(\tau)$$

subject to the constraint

$$(41) \quad ER_{j,i}^E(\tau) \geq AR_i(\tau) \quad \forall i \in I.$$

Algorithm 4 Pseudo-code for FDL D heuristics

1. allocate an instance of each task type to every node in every data center in every epoch
 2. **while**
 3. **for** each node with tasks still allocated to it
 4. **for** each task type on the node
 5. temporarily remove task type from node
 6. **if** FDL D-CL
 7. estimate execution rates using Eq. 36 (CER_i)
 8. **else if** FDL D-TAO
 9. estimate execution rates using Eq. 40 and ϕ_i^C
 10. **else if** FDL D-SO
 11. calculate execution rates using Eq. 40 and ϕ^C
 12. estimate power costs using Eq. 43
 13. calculate F^S from F^{ER} and F^C
 14. **if** execution rate constraints are not violated (Eq. 37 for FDL D-CL, Eq. 41 FDL D-SO & FDL D-TAO)
 15. add to set of possible task removal operations
 16. restore task type to node
 17. **if** set of possible task removal operations is empty
 18. **break**
 19. **else**
 20. choose 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 $AC_d(\tau)$, $\forall d \in D$, $\forall \tau \in N^\tau$)
-

To compensate for performance degradation due to co-location effects, node over-provisioning is accomplished by the factor F . F is replaced by either ϕ^C or ϕ_i^C in Eq. 40 when using either FDL D-SO or FLDB-TAO, respectively.

The execution rate force F^{ER} is calculated using

$$(42) \quad F^{ER}(\tau) = \sum_{i \in I} e^{\left(\frac{Z}{AR_i(\tau)} - 1\right)} - 1.$$

When considering the FDL D-CL heuristic, the term Z is replaced by $CER_i(\tau)$, and is replaced by $ER_{j,i}^E(\tau)$ when using either FDL D-SO or FDL D-TAO. Observe that $F^{ER}(\tau)$ decreases to zero as the ratio of Z to $AR_i(\tau)$ decreases to one.

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

$$(43) \quad PC_d^E(\tau) = E_d^{price}(\tau) \cdot \left(\sum_{j \in J^d} \sum_{m=1}^{N_{d,j}} P_{d,j,m}^E \cdot \left(1 + \frac{1}{CUP_d} \right) \cdot Eff_d - R_d(\tau) \right)$$

where

$$(44) \quad P_{d,j,m}^E = \begin{cases} P_j^{Sleep} & \text{if } |W_{d,j,m}| = 0 \\ \phi^j P_j^D + P_j^S & \text{else} \end{cases}.$$

Let $C_d^{actual}(\tau)$ be sum of power ($PC_d^E(\tau)$) and allocation ($AC_d(\tau)$) costs incurred at data center d during epoch τ . Let $C_d^{max}(\tau)$ be the maximum real power cost possible at data center d , calculated using

$$(45) \quad C_d^{max}(\tau) = E_d^{price}(\tau) \cdot \sum_{j \in J^d} N_{d,j} \cdot [\phi^j P_j^D + P_j^S].$$

The cost force F^C can then be calculated with

$$(46) \quad F^C(\tau) = \sum_{d=1}^D e^{\left(\frac{C_d^{actual}(\tau)}{C_d^{max}(\tau)} \right)} - 1.$$

Observe that the value of F^C goes to zero as the ratio of $C_d^{actual}(\tau)$ to $C_d^{max}(\tau)$ decreases to zero.

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

$$(47) \quad F^S = \sum_{\tau=1}^{N^\tau} F^{ER}(\tau) + F^C(\tau).$$

4.4.3. GENETIC ALGORITHM HEURISTIC. We also designed a third heuristic; a genetic algorithm load distribution with full co-location awareness (GALD-CL). The GALD-CL heuristic (Algorithm 5) 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 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 nodes (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 better placement decisions.

The GALD-CL heuristic addresses two potential shortcomings of the FDLN variants. First, the nature of the FDLN variants prevents them from making DVFS decisions. The greedy heuristic in the GALD-CL approach chooses the most efficient P-state for each task type on each node type [4]. Second, the FDLN variants are susceptible to becoming trapped

Algorithm 5 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 $AC_d(\tau)$, $\forall d \in D, \forall \tau \in N^\tau$)
-

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.

4.5. SIMULATION RESULTS

4.5.1. EXPERIMENTAL SETUP. Experiments were conducted for groups of four, eight, and sixteen data centers. The site locations for data centers were selected so that each group would have a fairly even east coast to west coast distribution to better exploit TOU pricing and renewable power. Each data center consists of 4,320 nodes arranged in four aisles, and is heterogeneous within itself, having nodes from either two or three of the node types given in Table 4.1, with most locations having three nodes types and per-node core counts that range from 4-12 cores depending on the mix of node types.

The electricity prices used during experiments were taken directly from Pacific Gas and Electric (PG&E) Schedule E-19 [91]. 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 data was obtained from [92], where each location uses either wind power, solar power, or a combination of the two.

Sleep power for all nodes is calculated as a fixed percentage of static power for each node type, assumed to be 16% based on a recent study of node power states [93]. The average node utilization factor used during FDL D 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 node to or from a sleep state, T^S , was assumed to be five minutes.

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 FDL D heuristics for four, eight, and sixteen locations completed on average in one, four, and thirteen minutes per epoch simulated, respectively.

Each of five task types is representative of a different benchmark from the PARSEC benchmark suite. Task execution times and co-located performance data were obtained from running the benchmark applications on the nodes listed in Table 4.1 [52]. Synthetic task arrival rates were constructed that follow a sinusoidal pattern, peaking during business hours and declining during the evening and until the next morning.

4.5.2. MONETARY 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

TABLE 4.1. Node 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

a 24-hour period (Fig. 4.2). It can be observed that the FDL-D-CL technique, using the co-location models, performs the best of the FDL-D variants for provisioning the minimum number of nodes necessary to meet execution rate requirements. The FDL-D-SO technique performed the worst, severely over-provisioning nodes. The GAL-D-CL heuristic outperformed all other approaches. While not performing as well as well as the GAL-D-CL, the FDL-D variants do have the advantage of reaching a solution more quickly, which may be beneficial in some cases.

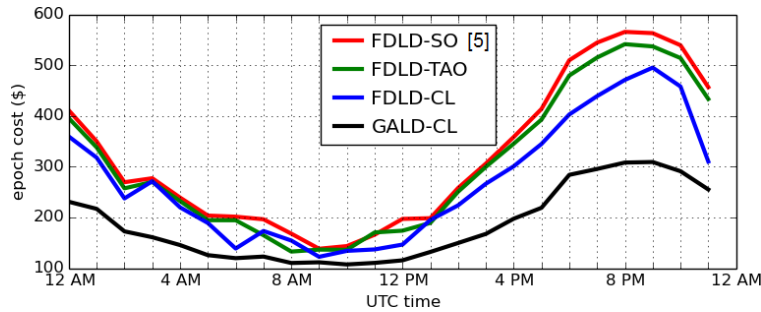


FIGURE 4.2. System costs across twenty four epoch period for each heuristic, four locations.

4.5.3. WORKLOAD TYPE ANALYSIS. The experiment in Section 4.5.2 used a workload that was a mix of memory-intensive and CPU-intensive tasks types. Fig. 4.3 shows experiments for the FDL-D-CL and GAL-D-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 memory-intensive (using data from *canneal*, *cg*, *ua*, *sp*, and *lu* benchmarks), and one where the tasks are highly CPU-intensive (using data from *fluidanimate*, *blackscholes*, *bodytrack*, *ep*, and

swaptions benchmarks). 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.

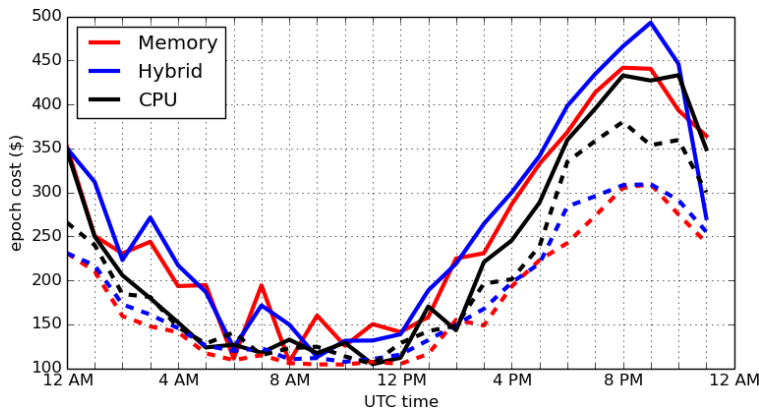


FIGURE 4.3. System costs across twenty four epoch period for different workload types, for a group of four locations. FDDL-CL shown as solid line, and GALD-CL shown as dashed line.

4.5.4. SCALABILITY ANALYSIS. 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 FDDL-SO method is given in Table 4.2. It should be noted that as the number of data centers in the group grows larger, the time for the FDDL 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 FDDL heuristics was very manageable as the number of data centers in the group increased.

TABLE 4.2. Monetary cost reduction compared to FDDL-SO [1]

Heuristic	4 data centers	8 data centers	16 data centers
FDDL-TAO	5.2%	4.6%	5.7%
FDDL-CL	14.2%	15.7%	18.8%
GALD-CL	39.7%	39.2%	36.8%

4.6. CONCLUSION

We proposed three 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 about the co-location interference in the decision process of the heuristics resulted in a lower energy cost by reducing or eliminating node over-provisioning while still meeting all required workload execution rates. Our FDL-D-CL and GALD-CL heuristics resulted on average in 10% and 37%, respectively, lower total cost than the prior work (represented by the FDL-D-SO heuristic) [1]. In systems where the workload profile changes rapidly and therefore requires short epochs (a few minutes), we recommend FDL-D-CL. When the workload profile is not changing rapidly and workload distribution decisions are given more time (an hour), GALD-CL is a more suitable technique. Possible directions for future studies in this area are presented in Chapter 6.

ONLINE RESOURCE MANAGEMENT IN THERMAL AND ENERGY CONSTRAINED HETEROGENEOUS HIGH PERFORMANCE COMPUTING SYSTEMS[§]

5.1. INTRODUCTION

It was reported by the Natural Resources Defense Council that U.S. data centers consumed an estimated 91 billion kilowatt-hours of electricity in 2013, with an estimated \$9 billion annual cost that is expected to grow to \$13 billion by 2020 [94]. In addition, the push to exascale is largely prohibited by electricity consumption. For example, the highest-performing supercomputer according to the Top500 list is the Tianhe-2 system that at 33.86 petaFLOPS has a peak power consumption of 17,808 kW, which would cost approximately \$17 million per year in electricity using the average cost of electricity in commercial sectors in the U.S. [3, 7]. Extrapolating this system to exascale results in an energy cost of \$500 million per year, or approximately \$1.37 million per day for one high-performance computing system (HPC). The high cost of energy for data centers today (and projected to be higher in the future), combined with the prohibitive cost of energy for an exascale system, makes energy-aware management of HPC systems of paramount importance.

The cooling infrastructures in HPC systems consume a significant portion of overall energy, and as such managing resources in a thermal-aware manner is an extremely important

[§]This research is currently under review. The other co-authors of this work are: Sudeep Pasricha, Anthony A. Maciejewski, H. J. Siegel, and Patrick J. Burns. This research was supported by NSF grants CCF-1302693 and CCF-1252500. This research used the CSU ISTE_C Cray System supported by NSF Grant CNS-0923386. We thank Hewlett Packard for donating servers for this work.

aspect of energy-efficient HPC system operation. Because it can be difficult to predict temperatures in an HPC facility due to complicated air flow patterns, it is typical to just provide more cooling than necessary to ensure that compute nodes do not overheat and reliability is maintained even under worst-case scenarios. But such an approach comes with a higher-than-necessary cost of energy consumption by the cooling infrastructure. One simple solution to reduce the cooling energy cost is to increase the temperature of the facility by raising the thermostat temperature of the computer room air conditioning (CRAC) units to a reasonable temperature under normal conditions (i.e., an average workload), and then relying on compute nodes to throttle themselves to lower processing speeds using dynamic voltage and frequency scaling (DVFS) if the workload becomes high and temperature sensors detect overheating. This approach, however, causes unexpected decreases in performance due to throttling that can result in missed task deadlines.

Using thermal models to predict temperatures at different locations within the HPC facility can allow for proactive resource management techniques to understand the thermal implications of allocating tasks to different cores around the facility. Knowing such thermal information offers the benefit of being able to turn up the CRAC units' thermostat temperatures and allocate tasks to cores in such a manner that the facility can be run at a hotter temperature without triggering the throttling mechanisms. The challenge is that temperature prediction requires complicated airflow models to be calculated for every mapping decision or CRAC thermostat setting. This can be a time-consuming process, especially when performing a full computational fluid dynamics (CFD) simulation or when using thermal models that use heat flow estimations based on results from CFD simulations, e.g., [95, 96]. In a dynamic online resource management environment, allocation decisions need to be made quickly to start task execution as soon as possible.

The goal of our online resource management framework proposed in this paper is to assign dynamically arriving tasks to execute on cores such that the collective reward earned from completing those tasks by their deadline is maximized over a period of time (e.g., a day), within an allowable energy budget allotted over that period of time. We assume there is control over (a) task-to-core mappings, (b) DVFS in cores to allow for performance state (P-state) changes, (c) CRAC thermostat settings, and (d) the perforated floor vent openings where cold supply air from the CRAC enters the HPC facility. We propose a novel resource management framework that uses a database of offline generated solutions, called *templates*, to assist an online resource manager in making thermal-aware decisions in real-time. The offline solutions provide: (a) an allowable set of cores that the online resource manager can use for task-to-core mapping decisions, (b) CRAC thermostat settings, and (c) which floor vents to close, partially open, or fully open. During runtime, our resource management framework uses a proposed online greedy mapping heuristic to assign dynamically arriving tasks to cores and set core P-states. The allowable core set provided by the offline solution assists the online mapping heuristic by limiting the number of cores that can have tasks mapped to them (and thus the search space) based on the current state of the data center.

In summary, we make the following novel contributions:

- An offline thermal and power-aware optimization formulation for generating templates that considers holistic control of a heterogeneous HPC system and its cooling system. To the best of our knowledge, this is the first work to consider floor vent opening decisions in HPC facility resource management.
- An online resource management framework that intelligently chooses templates based on the current state of the data center to assist our proposed greedy mapping heuristic in assigning dynamically arriving tasks to cores and P-states. The

templates aid the online resource manager in making fast power and thermal-aware mapping decisions with the goal of maximizing reward earned within an energy budget constraint.

- Analysis and comparison of our offline-assisted online resource management scheme under different workload environments and energy budget values with three online mapping schemes and two thermal management strategies from prior work.

The rest of the paper is organized as follows. In Section 5.2, we explain our models for compute nodes, workload, CRAC units, and thermal models. The problem statement is defined in Section 5.3. Section 5.4 details our proposed offline-assisted online resource management framework, and Section 5.5 describes the techniques with which we compare our framework. Our simulation setup and results are in Sections 5.6 and 5.7. We discuss related work in Section 5.8. In Section 5.9, we conclude and discuss ideas for future work.

5.2. SYSTEM MODEL

5.2.1. OVERVIEW. We study a heterogeneous HPC facility that is configured in a hot aisle/cold aisle manner (Fig. 5.1), such as the data center at Colorado State University or Hewlett Packard’s data center research facility in Fort Collins [97]. As demonstrated in Fig. 5.1, 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 then cool it. We assume the perforated floor tiles can be in the closed, partially open, or fully open positions.

5.2.2. COMPUTE NODES. The computing system is composed of a total of \mathbf{N} *compute nodes*, where compute nodes can be one of \mathbf{NT} heterogeneous *compute node types*. The

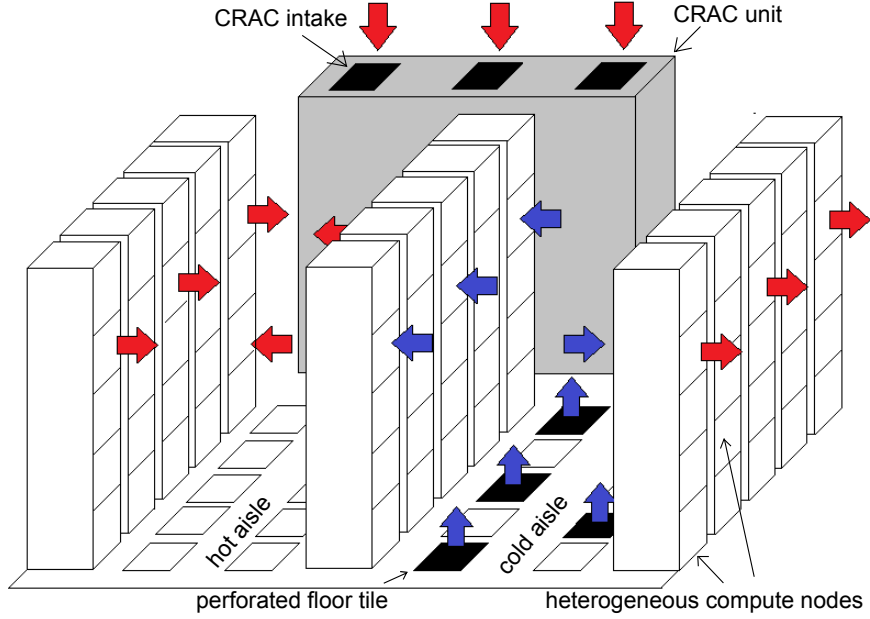


FIGURE 5.1. Data center aisle configuration.

compute node type of compute node j is denoted NT_j , where nodes within the same type are identical, meaning they contain the same number of cores and have the same performance and power characteristics. The cores within a node are homogeneous, and we assume the cores can be individually configured to different P-states that offer a tradeoff between power consumption and performance [12]. We denote NC as the *total number of cores* in the HPC system, NC_j as the *number of cores within compute node j* , and CT_k as the *node type to which core k belongs*.

5.2.3. WORKLOAD. The workload consists of dynamically arriving tasks that can be one of T *task types*. A task type i is defined by its deadline relative to its arrival time (D_i) and the reward earned for completing tasks of that type by the deadline (R_i). The reward earned is representative of a priority level given to different task types, where tasks with high reward may be more important to the system to complete or a user has paid more to get their job a higher priority. We denote the task type of a task i as TT_i .

Because the system is heterogeneous, tasks of the same type can have different execution times when executed on different node types. The execution times also vary across different P-states. We assume that we know the *estimated time to compute (ETC)* of any task type i on a core k of node type j in P-state $PS_{i,k}$, denoted $ETC_{i,j,PS_{i,k}}$. In an actual system, this can be approximated using historical, experimental, or analytical techniques [30–32].

5.2.4. POWER AND ENERGY MODELS.

5.2.4.1. *Compute Node Power.* We consider the idle power consumption of a compute node (e.g., from main memory, disks, fans) in addition to the non-idle power consumption when the CPU cores are active. We assume that CPU cores are able to change P-states over time, 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 and the P-state in which the core is executing the task. Let P_j^{idle} be the *idle power consumption of compute node j* , let $APC_{i,j,PS_{i,k}}$ be the average power consumed by a core k in a node of type j executing tasks of type i in P-state $PS_{i,k}$. We use these variables to calculate total energy consumption in Section 5.2.4.3.

5.2.4.2. *CRAC Unit Power.* 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) of the CRAC unit [74]. Let NCR be the total *number of CRAC units* in the HPC facility, $TC_i^{in}(t)$ be the *inlet temperature of CRAC unit i* at time t , $TC_i^{out}(t)$ be the *outlet temperature of CRAC unit i* at time t , 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* at time t , $P_i^{CRAC}(t)$, is calculated as [95]

$$(48) \quad P_i^{CRAC}(t) = \frac{\rho \cdot C \cdot AFC_i \cdot (TC_i^{in}(t) - TC_i^{out}(t))}{CoP(TC_i^{out}(t))}.$$

5.2.4.3. *Total Energy Consumption.* The energy consumption of a core is the sum of energy consumed for all tasks that are executed on the core over the desired time interval t_p . The *estimated energy consumed* by a core k of type CT_k when executing a task of type i in P-state $PS_{i,k}$, is $EEC_{i,CT_k,PS_{i,k}}$, which is the product of the execution time and average power consumption,

$$(49) \quad EEC_{i,CT_k,PS_{i,k}} = ETC_{i,CT_k,PS_{i,k}} \cdot APC_{i,CT_k,PS_{i,k}}.$$

As is typical in a highly utilized environment, we assume compute nodes cannot be deactivated (i.e., shut down). Therefore, we assume the idle power of a compute node is constant throughout the day and the *idle energy* consumed by a compute node j over a period of t_p , E_j^{idle} , is

$$(50) \quad E_j^{idle} = \int_0^{t_p} P_j^{idle} dt = P_j^{idle} \cdot t_p.$$

The energy consumed by a CRAC unit i is the integral of its power consumption over t_p time. That is, the *CRAC energy consumed* by CRAC unit i , E_i^{CRAC} , is

$$(51) \quad E_i^{CRAC} = \int_0^{t_p} P_i^{CRAC} dt.$$

The *total energy consumed* by the compute and cooling systems, \mathbf{E}^{total} , is the sum of core energy, idle node energy, and cooling energy. If $\mathbf{ET}_k^{t_p}$ represents the set of *executed tasks* on core k over t_p time, then we calculate \mathbf{E}^{total} as

$$(52) \quad \begin{aligned} \mathbf{E}^{total} &= \sum_{k=1}^{NC} \sum_{i \in \mathbf{ET}_k^{t_p}} \mathbf{EEC}_{i,CT_k,PS_{i,k}} \\ &+ \sum_{j=1}^N \mathbf{E}_j^{idle} + \sum_{z=1}^{NCR} \mathbf{E}_z^{CRAC}. \end{aligned}$$

5.2.5. TRANSIENT THERMAL MODEL. We assume the HPC facility has temperature sensors located at the inlets and outlets of the compute nodes. However, for simulation purposes, we use a transient model for data center thermal prediction [95] to act as sensor output values that track temperatures through time.

The transient thermal model is focused around a collection of *temporal contribution curves* ($\mathbf{c}_{i,j}(\mathbf{t})$) and weighting factors $\mathbf{w}_{i,j}$. The $\mathbf{c}_{i,j}(\mathbf{t})$ curves intuitively represent how long it takes for node (or CRAC) \mathbf{j} to experience a temperature change generated by node (or CRAC) \mathbf{i} . For example, if node \mathbf{i} is assigned a heavy workload and generates a large amount of heat, node \mathbf{j} located across the facility may not experience that heat from node \mathbf{i} until minutes later, as defined by the $\mathbf{c}_{i,j}(\mathbf{t})$ curve. The $\mathbf{w}_{i,j}$ factors represent how much of the heat generated by node \mathbf{i} is realized at node \mathbf{j} . Using these values, in addition to *outlet temperatures* (\mathbf{T}_i^{out}) of all nodes in the facility, we can calculate the *inlet temperature* of any node \mathbf{j} at time \mathbf{t} , $\mathbf{T}_j^{in}(\mathbf{t})$, as [95]

$$(53) \quad \mathbf{T}_j^{in}(\mathbf{t}) = \sum_{i=1}^{N+NCR} \mathbf{w}_{i,j} \cdot \int_{-\infty}^0 \mathbf{c}_{i,j}(\tau) \mathbf{T}_i^{out}(\mathbf{t} + \tau) d\tau.$$

The $\mathbf{c}_{i,j}(\mathbf{t})$ curves and $\mathbf{w}_{i,j}$ factors are calculated using CFD simulations following the method in [95].

We can calculate the *instantaneous power of node j* at a time \mathbf{t} , $\mathbf{PN}_j(\mathbf{t})$, by summing the idle power of the node j with the *APC* values of all cores within node j that are executing tasks at time \mathbf{t} . The power consumed by a compute node is dissipated as heat, and the outlet temperature at compute node j at time \mathbf{t} is a function of the inlet temperature, the power consumed, and the *air flow rate of the node \mathbf{AFN}_j* , calculated as [96]

$$(54) \quad \mathbf{T}_j^{out}(\mathbf{t}) = \mathbf{T}_j^{in}(\mathbf{t}) + \frac{\mathbf{PN}_j(\mathbf{t})}{(\rho \cdot C \cdot \mathbf{AFN}_j)}.$$

The outlet temperature of a CRAC unit is assumed to be the value to which the CRAC thermostat is set.

If a thermal violation occurs at the outlet of a compute node (i.e., a node exceeds threshold temperature $\mathbf{T}^{threshold}$), all cores within that node are throttled to their highest-numbered (lowest power) P-state until cores have finished executing their current tasks.

5.2.6. STEADY-STATE THERMAL MODEL. For our offline template generation, we assume an HPC facility in the steady-state. Therefore, we use a steady-state thermal model to determine inlet and outlet temperatures of all compute nodes in the HPC facility. Steady-state temperatures are calculated in a similar manner as transient temperatures, except without the temporal contribution ($\mathbf{c}_{i,j}(\mathbf{t})$) curves. 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{W} be the matrix of $\mathbf{w}_{i,j}$ values that represent the percentage of heat transferred from CRAC unit or node i to CRAC unit or node j . We can then calculate steady-state temperatures by solving the linear combination [96]

$$(55) \quad \mathbf{T}^{in} = \mathbf{W}\mathbf{T}^{out}.$$

5.3. PROBLEM STATEMENT

The objective of our resource manager is to maximize the reward earned over a period of time t_p (e.g., a day) for completing tasks by their individual deadlines. If the set of *tasks that have completed execution by their deadline by time t_p* is TCD_{t_p} , then the goal is to maximize *reward earned by the system* over that period, $R_{system}(t_p)$, calculated as

$$(56) \quad R_{system}(t_p) = \sum_{i \in TCD_{t_p}} R_{TT_i}.$$

The system is constrained by both energy consumption and node outlet temperatures. Over the t_p period of time, the total energy consumption of the facility is not to exceed an energy budget of γ . Reward ceases to be gained when the total energy consumption of the facility over time period t_p exceeds the energy budget, i.e., when $E^{total} \geq \gamma$. This is because in our model, we assume that the facility will discontinue operation until the next period of time. We examine the use of several different energy budget values in the results we present in Section 5.7.

Compute node threshold temperatures are soft constraints that are handled by the throttling mechanism described in Section 5.2.5. If a compute node j is overheating (i.e., $T_j^{out} > T^{threshold}$) then all cores within that node are throttled to their highest numbered P-state until all cores have finished executing their current tasks. We examine how throttling can affect the reward earned by the system (e.g., by causing tasks to miss deadlines) in Section 5.7.

5.4. PROPOSED RESOURCE MANAGER

5.4.1. OVERVIEW. A block diagram of our offline-assisted online thermal and energy aware resource manager is shown in Fig. 5.2. We chose an offline-assisted approach to our

online resource manager to assist in making fast decisions. Also, this type of offline-assisted online management concept has been shown to be effective in the past (e.g., [81]).

Dynamically arriving tasks enter the system, and at a *mapping event* the resource manager has to decide the following: (1) which cores within the HPC system to execute those tasks, and (2) the P-states that those selected cores are configured to when executing the assigned tasks. At a *thermal management event*, the resource manager decides (3) CRAC thermostat settings, and (4) which floor vents to close, partially open, or fully open. When the resource manager uses templates to assist in resource allocation, the template database provides the CRAC thermostat settings, floor vent openings, and an *allowable core vector* (*ACV*), as indicated by the dashed lines connecting the resource manager with the template database in the figure. The mapping event interval time (denoted t_{map}) is set to 60 seconds, and the thermal management event interval time (denoted $t_{thermal}$) is set to 300 seconds in this study.

Template generation is performed offline to create a number of templates in a database that each reflect a steady-state solution for different dynamic states of the HPC facility. We

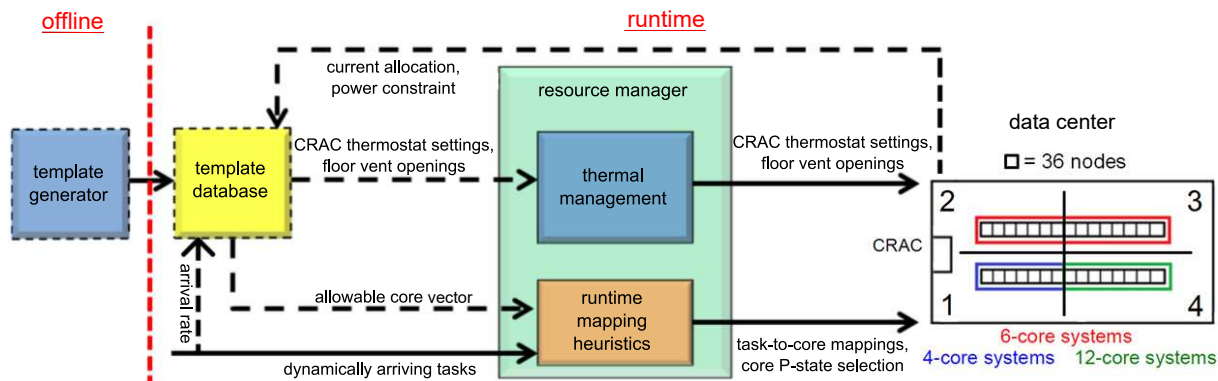


FIGURE 5.2. Block diagram of the resource management framework. Note that the specific data center shown is for illustrative purposes only, and our framework can be applied to any data center configuration.

first present the parameters used to define the state of the HPC facility, and then formulate a steady-state optimization problem to maximize an average service rate of the HPC facility subject to power and thermal constraints. We generate a number of templates that vary in three parameters (inputs): (a) newly arrived workload arrival rate, (b) number of cores already executing tasks in each of the quadrants of the HPC facility, and (c) power constraint. The quadrants of the data center we study are shown in Fig. 5.2. During runtime, our proposed framework chooses a template based on values of those parameters to obtain the values to set the CRAC thermostats, the floor vent opening combinations, and the **ACV**. Because there are an infinite number of possibilities for these parameters (resulting in an infinite number of templates required to capture every possible combination of these parameters), we discretize these parameters into a reasonable set of values and round actual values to the closest discretized value when selecting the appropriate template. The resolutions we discretize these values to are given in Section 5.6.4.

5.4.2. TEMPLATE INPUT PARAMETER DEFINITIONS. *Parameter 1:* The first parameter, *newly arrived workload arrival rate*, is calculated as the sum of all arrival rates of the **T** task types over a considered interval (either the mapping event interval or thermal management interval), weighted by their execution times. We denote this parameter $\lambda_{arriving}^{weighted}(t)$. First, if we denote NCT_j^{total} as the *total number of cores that are of node type \underline{j}* , then the average execution time for a task type **i** across all **NT** node types in the middle P-state for cores within that node type (P_j^{mid}), ETC_i^{avg} , is

$$(57) \quad ETC_i^{avg} = \frac{\sum_{j=1}^{NT} (NCT_j^{total} \cdot ETC_{i,j,P_j^{mid}})}{\sum_{j=1}^{NT} NCT_j^{total}}.$$

The middle P-state is the P-state halfway between the highest-power and lowest-power P-state, and is used in these offline estimations to represent an “average” P-state value. The online mapping heuristics actually select P-states that cores are in when executing various tasks.

The average execution time across all T task types, ETC^{avg} , is then calculated as

$$(58) \quad ETC^{avg} = \frac{\sum_{i=1}^T ETC_i^{avg}}{T}.$$

We normalize these values with the average execution time across all T task types using

$$(59) \quad ETC_i^{normalized} = \frac{ETC_i^{avg}}{ETC^{avg}}.$$

If $\lambda_i(\mathbf{t})$ is the arrival rate for tasks of type i over a given time interval \mathbf{t} , our *newly arrived workload arrival rate* parameter ($\lambda_{arriving}^{weighted}(\mathbf{t})$) is the sum of weighted arrival rates over all T task types. Intuitively, $\lambda_{arriving}^{weighted}(\mathbf{t})$ is an average amount of “work” arriving to the system that is represented as one parameter. We calculate $\lambda_{arriving}^{weighted}$ by averaging the arrival rates of all task types in the system, weighted by their execution times (as a greater execution time implies more work to be done). $\lambda_{arriving}^{weighted}(\mathbf{t})$ is calculated as

$$(60) \quad \lambda_{arriving}^{weighted}(\mathbf{t}) = \sum_{i=1}^T ETC_i^{normalized} \cdot \lambda_i(\mathbf{t}).$$

Parameter 2: The number of cores already executing tasks in quadrant q , $NC_q^{executing}$, is the number of cores that are currently executing tasks assigned to them by the dynamic resource manager in a quadrant q from prior mapping events (for a visual example of quadrants, please see Fig. 5.2). We use the $\lambda_{arriving}^{weighted}(t)$ and $NC_q^{executing}$ parameters as estimations of the current state of the HPC facility when the dynamic scheduler is choosing a template.

Parameter 3: The last parameter, the power constraint (denoted P_{const}), is the allotted power the system can use in the steady-state optimization problem.

Because the number of templates to be generated is exponential with the number of input parameters, we limit the total number of parameters in this work to six ($\lambda_{arriving}^{weighted}(t)$, $NC_1^{executing}$, $NC_2^{executing}$, $NC_3^{executing}$, $NC_4^{executing}$, and P_{const}). However, the same concepts could be used for a larger number of input parameters (e.g., a finer discretization of a facility than quadrants, or parameters that give exactly *which* cores are executing tasks).

5.4.3. OFFLINE TEMPLATE GENERATION. The template generation technique tries to find an allocation that matches the service rate of the system with the estimated weighted arrival rate $\lambda_{arriving}^{weighted}(t)$ summed with an estimation of the work that already exists in the system ($\lambda_{existing}^{weighted}(t)$) at a given mapping event or thermal management event. The *average execution rate* for a core in node j across all T task types, ER_j^{avg} , is

$$(61) \quad ER_j^{avg} = \frac{\sum_{i=1}^T \left(\frac{1}{ETC_{i,NT_j,P_j^{mid}}} \right)}{T}.$$

Therefore, if $NC_j^{allowable}$ represents the number of cores in node j allowed to execute tasks, the service rate of an allocation in this steady-state environment, $\mu_{allocation}$, is

$$(62) \quad \mu_{allocation} = \sum_{j=1}^N NC_j^{allowable} \cdot ER_j^{avg}.$$

Because we formulate this steady-state optimization problem as a non-linear program, we wish to avoid integer decision variables ($NC_j^{allowable}$) that create a complex mixed-integer non-linear program, so we introduce a continuous “utilization” decision variable U_j that represents the utilization of node j , and later round this to an integer-valued number of cores, i.e., $\lceil U_j \cdot NC_j \rceil = NC_j^{allowable}$. Therefore, we calculate the service rate of a given allocation when using a continuous variable, $\mu_{allocation}^{continuous}$, as

$$(63) \quad \mu_{allocation}^{continuous} = \sum_{j=1}^N U_j \cdot NC_j \cdot ER_j^{avg}.$$

We assume the steady-state optimization problem uses an “average” workload, and because power consumption of compute nodes is a function of the task types assigned to cores on that node, we take an average of that power across all task types. That is, we assume the *core power consumption* of a core in node j , P_j^{core} , in our steady-state formulation is

$$(64) \quad P_j^{core} = \frac{\sum_{i=1}^T APC_{i,NT_j,P_j^{mid}}}{T}.$$

Therefore, the estimation for total power consumption of the HPC facility in the steady-state, $P_{total}^{facility}$, is

$$(65) \quad P_{total}^{facility} = \sum_{j=1}^N (P_j^{idle} + U_j \cdot NC_j \cdot P_j^{core}) + \sum_{z=1}^{NCR} P_z^{CRAC}.$$

Because the template generator is not given information regarding exactly *which* cores are executing tasks by the online resource manager, the assumption is made that the existing work within a quadrant (represented by $NC_q^{executing}$) is evenly divided among all nodes within that quadrant. That is, if NC_q^{total} is the total number of cores in quadrant q , then

each node j in quadrant q has a U_j value of at least $U^q = \frac{NC_q^{executing}}{NC_q^{total}}$, i.e., U^q represents the fraction of total cores allocated in quadrant q that were already assigned tasks to execute. If NN^q is the number of nodes in quadrant q , then the estimated amount of work that exists in the system ($\lambda_{existing}^{weighted}(t)$), is calculated as

$$(66) \quad \lambda_{existing}^{weighted}(t) = \sum_{q=1}^4 \sum_{j=1}^{NN^q} U^q \cdot NC_j \cdot ER_j^{avg}.$$

The decision variables for the steady-state optimization problem are the utilizations of compute nodes, U_j , and the CRAC thermostat temperatures, TC_i^{out} . The following equation shows the optimization problem for the steady-state offline optimization problem:

$$(67) \quad \text{maximize } \frac{\mu_{allocation}}{P_{total}^{facility}}$$

subject to

- (1) $\mu_{allocation} \leq \lambda_{arriving}^{weighted}(t) + \lambda_{existing}^{weighted}(t)$
- (2) $U^q \leq U_j \leq 1, \forall j \in q, \forall q$
- (3) $P_{facility}^{total} \leq P_{const}$
- (4) $T_j^{out} \leq T^{threshold}, \forall j$

The objective is to maximize the amount of work performed per unit power, where the decision variable U_j affects both $\mu_{allocation}$ and $P_{total}^{facility}$, and decision variable TC_i^{out} affects $P_{total}^{facility}$ as the thermostat temperature of the CRAC units contributes to the power consumption of the facility. Constraint 1 guarantees that the allocated service rate does not exceed the amount of work to be done (newly arriving and existing). Constraint 2 guarantees that the utilization of each node (representative of the number of cores that are allowed to be active in that node) is greater than that already assigned to that node (represented by

U^q), and less than 1.0 (as a node cannot have more cores allowed to have tasks assigned to it than the total number of cores that exist in that node). Constraint 3 guarantees the power constraint. Constraint 4 guarantees the thermal constraints (a node’s outlet temperature cannot exceed $T^{threshold}$). The optimization for this problem formulation is performed using KNITRO software [98], a non-linear programming solver, to obtain the U_j and TC_i^{out} values. Rounding is then performed (using a ceiling function) to obtain discrete $NC_j^{allowable}$ values from the continuous U_j values.

Floor vent openings for each of the quadrants are not considered a decision variable in the presented optimization problem, as they are integer variables (either closed, partially open, or fully open), and accurately relaxing these to continuous variables is out of the scope of this paper. Thus, when generating the actual templates, we perform the above optimization when considering distinct thermal models for all possible floor vent opening combinations (closed, partially open, or fully open) in all quadrants. That is, we have coefficients for a steady-state thermal model (see Section 5.2.6) calculated for all vent opening combinations, assuming all vents in a given quadrant are opened to the same degree. We perform the template optimization for each of the floor vent opening combinations and record the best one to store in the template database.

The template optimization problem is not formulated to provide the online resource manager with an exact optimal solution for a given workload, but rather provide the online resource manager an energy-efficient estimation of where it should place tasks in the facility (the $NC_j^{allowable}$ values), and what the CRAC thermostat settings (TC_i^{out}) and floor vent configurations should be set to if tasks were allocated based on those $NC_j^{allowable}$ values.

5.4.4. ONLINE MAPPING HEURISTIC. The greedy online mapping heuristic assigns dynamically arriving tasks to cores, and also determines the P-states that cores are configured

to when executing those tasks. The heuristic is invoked at the beginning of every mapping event. At the start of a mapping event, our greedy deadline-aware heuristic (Alg. 6) drops all tasks in the unmapped batch that would not be able to make their deadlines even when executed on their fastest node type in the fastest P-state (line 1). Next, the \mathbf{ACV} is obtained. The \mathbf{ACV} is a vector composed of the $\mathbf{NC}_j^{allowable}$ values, where element \mathbf{j} of the \mathbf{ACV} is the number of cores in node \mathbf{j} that are allowed to have tasks mapped to them. If using templates to assist in resource allocation, the \mathbf{ACV} is obtained from the template in the template database that most-closely represents the parameters given in Section 5.4.2 (line 2). When not using templates (i.e., using a comparison thermal management technique from Section 5.5), the \mathbf{ACV} consists of all cores within the HPC facility (line 3). From the \mathbf{ACV} , a subset of cores called *idle allowable cores* is obtained, which is a set of cores not currently executing tasks in the \mathbf{ACV} (line 4). That is, the idle allowable cores in a node \mathbf{j} is the number of cores within that node that are both allowed to have tasks mapped to them and are currently not executing any tasks. The idle allowable cores in a node \mathbf{j} are found as follows: if a node \mathbf{j} has less cores that are currently executing tasks than what the value of \mathbf{ACV}_j is, then the difference between those values is how many more cores within node \mathbf{j} are idle allowable cores, and are allowed to have tasks mapped to them. For example, if the value of \mathbf{ACV}_j is 2 and there is currently only one core in node \mathbf{j} currently executing a task, one more core in node \mathbf{j} is allowed to execute a task. That core is randomly chosen from the idle cores in node \mathbf{j} and added to the idle allowable core set.

The unmapped batch of tasks is then sorted in descending order by their reward earned if completed by their deadline (line 5). The algorithm then iteratively assigns tasks to the core (from the set of idle allowable cores) and P-state combinations that result in the lowest energy while still meeting the deadline until there are no tasks left to map, or until there are

no idle allowable cores to map tasks (lines 6-10). If n is the number of idle allowable cores at a mapping event, and m is the number of tasks in the unmapped batch at a mapping event, then the complexity of the heuristic is approximately $\mathcal{O}(n^2 + m \log m)$ if $m > n$, or $\mathcal{O}(mn)$ if $n > m$.

Algorithm 6 Pseudo-code for our greedy deadline-aware heuristic

1. drop tasks that cannot meet deadline
 2. **if** using templates **then** obtain **ACV** from template database
 3. **else** set **ACV** to all cores in system
 4. obtain *idle* allowable core set
 5. sort unmapped batch of tasks in descending order by reward
 6. **while** unmapped batch is not empty **and** idle allowable core set is not empty
 7. select first task in unmapped batch
 8. find core/P-state combination from set of idle allowable cores that gives lowest energy and meets deadline
 9. assign task to that core and P-state
 10. remove task from unmapped batch and remove core from idle allowable core set
 11. **end while**
-

5.4.5. SUMMARY OF OUR OFFLINE-ASSISTED ONLINE RESOURCE MANAGEMENT FRAMEWORK. Referring back to Fig. 5.2, our framework assigns dynamically arriving tasks to cores within the HPC facility, selects the core P-states, CRAC thermostat settings, and floor vent openings. Task-to-core assignments and P-state configuration settings are performed online using the greedy online mapping heuristic presented in Section 5.4.4. At a thermal management event, the CRAC thermostat settings and floor vent opening decisions are found by using the appropriate template stored in a database that relates to the current estimation of the state of the HPC facility and workload. At a mapping event, templates can assist dynamic mapping heuristics by providing an allowable core vector (**ACV**) to map tasks to for the current state of the facility. The appropriate template is selected based on three parameters: (a) newly arrived workload, (b) current core allocation in each quadrant, and

(c) a power constraint calculated by dividing the energy budget remaining in our considered period of time t_p (e.g., a day) by the amount of time remaining.

5.5. COMPARISON TECHNIQUES

5.5.1. OVERCOOLING AND THROTTLING THERMAL

MANAGEMENT STRATEGIES. The thermal management techniques presented in this section provide the CRAC thermostat temperatures ($\mathbf{TC}^{out}(\mathbf{i}, \mathbf{t})$). It is common in HPC facilities to simply *overcool* the HPC facility, by setting the CRAC thermostat temperatures to a low enough temperature so that compute nodes do not overheat (i.e., node outlet temperatures do not exceed $\mathbf{T}^{threshold}$) even under high-heat scenarios caused by fully-utilized compute nodes. The *overcool* strategy simply sets the CRAC thermostat to a low constant temperature (22°C). Another method is to run the HPC facility at a higher temperature, and allow throttling to manage overheating nodes (as explained in Section 5.2.5). The *throttling* strategy sets the CRAC thermostat to a relatively high constant temperature (28°C). We assume that when using the *overcool* or *throttling* approaches, the \mathbf{ACV} (cores that are allowed to have tasks actually mapped to them) is composed of all cores in the system. The floor vents are set to the fully-opened configuration for both *overcool* and *throttling* strategies.

5.5.2. ONLINE MAPPING HEURISTICS.

5.5.2.1. *Consolidation*. At the beginning of a mapping event, the *consolidation* technique assigns unmapped tasks in an arbitrary order to a core within the system that is not currently executing any task. Ties (where multiple cores are not executing any tasks) are solved by assigning the task to the first core in the list of all cores in the HPC system (\mathbf{NC}). In this work, cores are ordered in the list starting with all cores in quadrant 1 (from left-to-right by rack in Fig. 5.2, and from top node to bottom node in each rack), then quadrant 2, etc. Cores

are configured to execute tasks in P-state P0, which represents the highest performance and highest power P-state. This technique results in spatial consolidation of tasks, similar in concept to virtual machine consolidation (e.g., [93, 91]).

5.5.2.2. *Load Balancing by Node.* At the beginning of a mapping event, the *load balancing by node* (*LBBN*) technique assigns unmapped tasks in an arbitrary order to the node in the system that is currently executing the least number of tasks. For example, a node with zero cores currently executing tasks is always preferred over a node with one core already executing a task. Similarly, a node with one core currently executing a task is always preferred to a node with two cores currently executing tasks. When a node is chosen for assignment, a random core within that node is assigned to execute the task. Ties (where multiple nodes have the same least number of cores executing tasks) are solved by assigning the task to the first node in the list of all N nodes. When a task is assigned to a core, the core is configured to execute the task in P-state P0, which represents the highest performance and highest power P-state. This technique is very similar to the FIRSTAVAILABLE scheduler from Moab workload manager, where nodes are allocated to jobs in the order they are presented to the resource manager [99].

5.5.2.3. *Load Balancing by Rack.* At the beginning of a mapping event, the *load balancing by rack* (*LBBR*) technique assigns unmapped tasks in an arbitrary order to the rack (as defined as a vertical cabinet containing multiple nodes, e.g., represented by a black square in Fig. 5.2) that is currently executing the least number of tasks. For example, a rack with zero cores currently executing tasks is always preferred over a rack with one core currently executing a task. Similarly, a rack with one core currently executing a task is always preferred to a rack with two cores currently executing tasks. When a rack is chosen for assignment, a random core within a random node within the selected rack is assigned to execute the

task. Ties (where multiple racks have the same least number of cores executing tasks) are solved by assigning the task to the first rack in the list of all racks. When a task is assigned to a core, the core is configured to execute the task in P-state P0, which represents the highest performance and highest power P-state. This technique results in a uniform spatial distribution of tasks throughout the facility, similar in concept to the UniformWorkload [74] and Uniform Task [100] techniques.

5.6. SIMULATION SETUP

5.6.1. OVERVIEW. In our simulations, we consider a heterogeneous platform composed of one CRAC unit, 30 server racks with 36 compute nodes per rack (a total of 1,080 compute nodes), as shown in Fig. 5.2. Each node is one of three node types (see Table 5.1). Power characteristics of our different node-types were obtained from power measurements of three server class machines when executing different PARSEC benchmarks [101] in all P-states, and the workload characteristics were also obtained from executing the benchmarks on the node-types listed in Table 5.1. The threshold temperature for compute nodes was set to 33°C, which is on the high end of ASHRAE’s data center guidelines for allowable temperatures [50].

TABLE 5.1. Node types used in simulations

node type	Lenovo TS140	HP Z600	HP Z820
total # of nodes	270	540	270
# of cores in node	4	6	12
# of P-states	16	9	16
air flow rate (m^3/s)	0.0284	0.0519	0.0566

5.6.2. WORKLOAD. We study three different workload arrival patterns (see Fig. 5.3). Fig. 5.3 (a) shows a constant task arrival rate of λ_{const} , and Fig. 5.3 (b) shows a bursty

arrival pattern, with intervals of a λ_{high} task arrival rate followed by intervals of λ_{low} task arrival rate. Fig. 5.3 (c) shows a sinusoidal arrival pattern, with higher task arrival rates during daytime and smaller during nighttime. For the arrival rate patterns of Fig. 5.3 (a) and (b), tasks arrive to the system according to a Poisson process with rate λ_{const} , λ_{high} , or λ_{low} , with λ_{const} set to 6.0, λ_{high} set to 9.0, and λ_{low} set to 3.0 tasks per second. The task arrivals for the sinusoidal pattern are from the MetaCentrum2 workload trace obtained from the Parallel Workload Archive [92]. The task type of a task is randomly chosen based on a uniform distribution.

The execution time and power characteristics for different task-types are from the PARSEC benchmark suite, and cor-responding ETC and APC matrix entries for each benchmark on each node type in each P-state are obtained by measuring the execution times and average power consumption of those benchmarks on each of the machines and P-states from Table 5.1. The following benchmarks were used: *canneal*, *cg*, *ua*, *sp*, *lu*, *fluidanimate*, *blackscholes*, *bodytrack*, *ep*, and *swaptions*. These benchmarks were chosen to enable a diverse portfolio of consumer and scientific applications on which to evaluate our framework. The deadlines for tasks of a given type were set to the average execution time for that task type across all node types in the P_{mid} P-state (see Equation 57). The reward values earned for completing tasks of a given type by their individual deadlines are linearly proportional to their execution times (i.e., tasks with longer execution times have higher reward values).

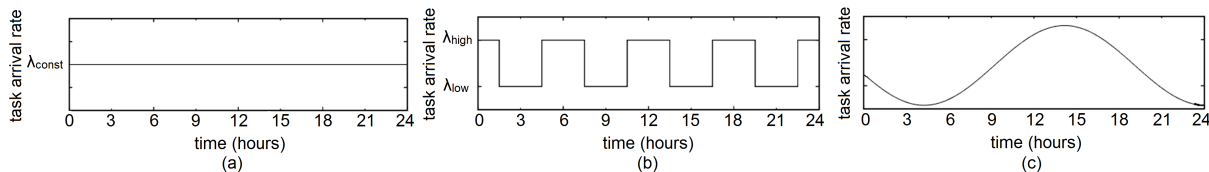


FIGURE 5.3. Workload arrival patterns with (a) constant, (b) bursty, and (c) sinusoidal patterns.

5.6.3. CRAC UNITS. The CoP for a CRAC unit is a function of its outlet temperature, τ , given by $CoP(\tau) = 0.0068\tau^2 + 0.0008\tau + 0.458$ [74]. The air flow rate of each CRAC unit is set to $26.1 \text{ m}^3/\text{s}$, which is the air flow rate of a CRAC unit with capacity to cool approximately 350 kW [77].

5.6.4. TEMPLATE GENERATION. We generated a range of templates for the parameters listed in Section 5.4.2 for our *templates* technique. For the newly arrived workload parameter ($\lambda_{arriving}^{weighted}(\mathbf{t})$), we used the values [2.2, 6.6, 11.0, 15.4, 19.8], as these values evenly divide the range of [0,22], and 22 is approximately the maximum service rate capability for the system in Table 5.1. The number of cores in quadrant 1 currently executing tasks ($NC_1^{executing}$) is discretized using the values [180, 540, 900] that splits the 1,080 total number of cores in that quadrant. Similarly, the number of cores executing tasks in quadrants 2 and 3 ($NC_2^{executing}$ and $NC_3^{executing}$) are discretized using the values [200, 605, 1010, 1415], and the number of cores executing tasks in quadrant 4 ($NC_q^{executing}$) is discretized into [180, 540, 900, 1260, 1620, 1980, 2340, 2700, 3060]. Last, the power constraint parameter (P_{const}) is discretized into [150, 170, 190, 210, 230] kilowatts.

5.7. RESULTS

In this section we present results, discussion, and analysis of our proposed offline-assisted online resource management framework that uses templates and our greedy deadline-aware online mapping heuristic, and compare it with the *consolidation*, *LBBN*, and *LBBR* mapping heuristics that use either *overcooling* or *throttling* thermal management approaches. The bar graphs showing results for bursty and constant arrival rate patterns discussed in the rest of this section represent the averages of 12 trials, with each trial varying in the actual arrival times of tasks (as a Poisson process is random) as well as the task types of arriving tasks

(another random process). The error bars are the 95% confidence intervals around the mean of those 12 trials.

The first results (Fig. 5.4) compare the *consolidation*, *LBBN*, *LBBR*, and *greedy* mapping techniques in combination with the *overcooling*, *throttling* thermal management techniques, as well as the greedy technique with the *templates* thermal management technique that represents our offline-assisted online resource management framework. Fig. 5.4, (a), (c), and (e), show the reward earned by the different techniques for the (a) constant workload arrival pattern, (c) bursty workload arrival pattern, and (e) sinusoidal workload arrival pattern. Similarly, Fig. 5.4 (b), (d), and (f) show the energy consumption of those techniques for the constant, bursty, and sinusoidal arrival rate patterns, respectively. These results are compared across a wide range of different daily energy budget values from 8000 MJ to 16000 MJ that represent a range from a highly constrained system to a less constrained system. For instance, *consolidation* using the *overcooling* thermal management technique is only able to process tasks for approximately half of the day when given an energy budget of only 8000 MJ. The system uses approximately 22000 MJ per day when all cores in the facility are executing the highest-powered task in P-state P0, and the cooling system is set in the *overcooling* management scheme. However, for energy budgets above 16000 MJ, results were observed to be the same as the results for 16000 MJ for the techniques and arrival rates simulated in this study.

We can observe in Fig. 5.4 (a) and (c) that our proposed greedy heuristic in combination with templates provides a significant benefit in reward earned at lower energy budget levels (less than 14000 MJ). At such low energy budget levels, the system is not able to remain activated for the entire day, regardless of the mapping heuristic or thermal management technique chosen, i.e., all available energy in the budget is consumed before the day ends

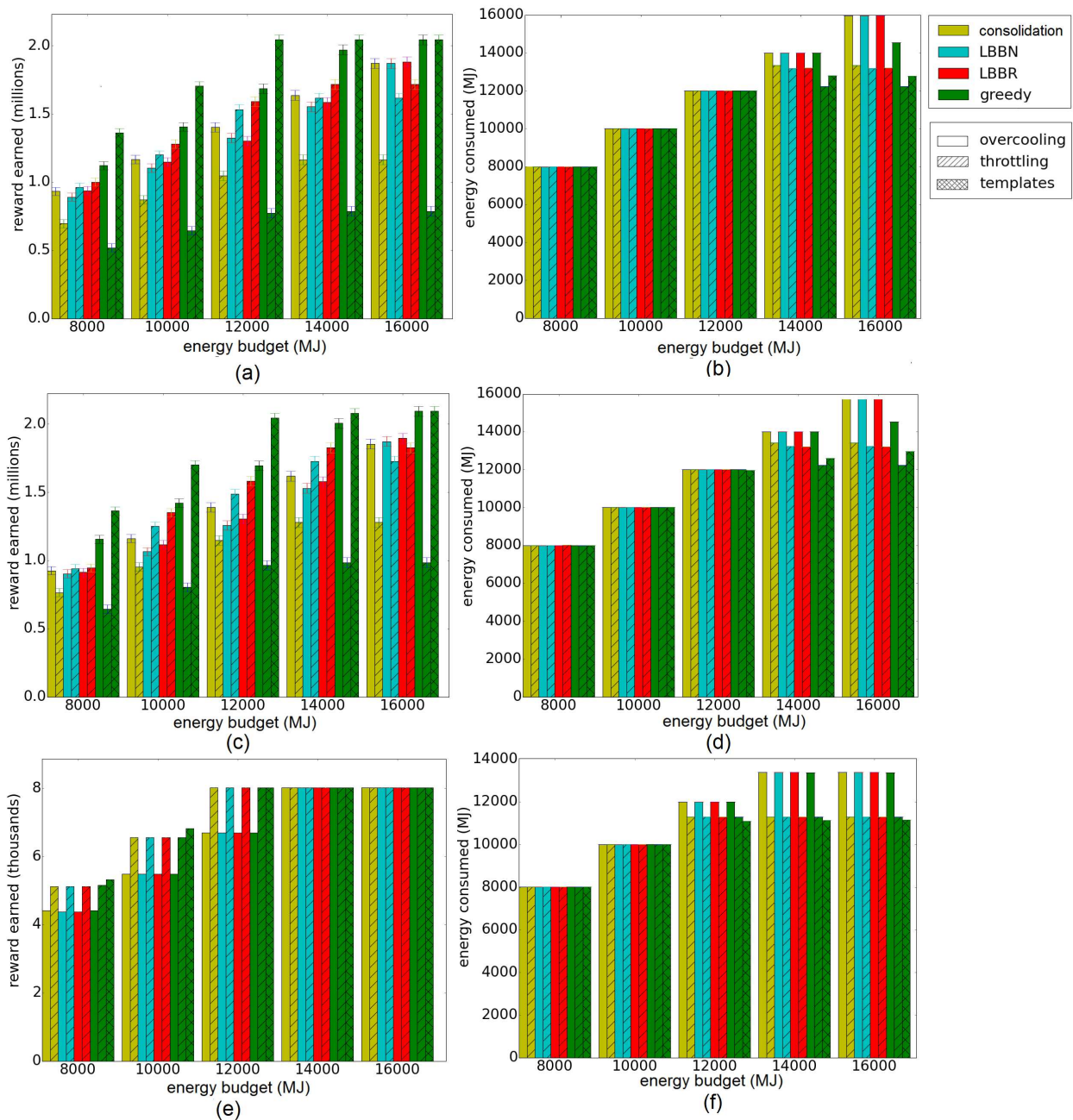


FIGURE 5.4. Compares reward earned by techniques for the (a) constant, (c) bursty, and (e) sinusoidal workload arrival patterns. Similarly, (b), (d), and (f) display energy consumed by those workload arrival patterns. All results in this figure are shown for the four runtime mapping techniques (consolidation, LBBN, LBBR, and greedy), and three thermal management strategies (overcooling, throttling, templates).

and the facility is essentially deactivated, not allowing any more tasks to complete. However, the greedy online mapping heuristic is able to exploit DVFS to save some energy, and when using templates is able to reduce cooling energy consumed (compared to *overcooling*), and therefore is able to complete more tasks by their deadline than the other techniques.

We can also analyze some other interesting trends in Fig. 5.4 (a) and (c) by examining other techniques. First, we can observe that when using the *consolidation* online mapping technique, the *overcooling* thermal management technique significantly outperforms the *throttling* thermal management technique. However, when using *LBBN* and *LBBR*, the *throttling* technique earns significantly more reward than *overcooling* technique. This is because *consolidation* maps most tasks spatially close to each other (i.e., in the same quadrant), whereas *LBBN* and *LBBR* spatially distribute the tasks in the facility. This means that when using the *throttling* thermal management technique (i.e., operating the facility at a hotter temperature and relying on the throttling mechanisms to prevent overheating), the *consolidation* technique packs all tasks in a corner of the facility and causes hotspots in that area, invoking the throttling mechanisms almost constantly and causing a large number of missed deadlines. The *throttling* technique works well for *LBBN* and *LBBR* because tasks are more spatially distributed, therefore causing fewer hotspots (and thus not invoking the throttling mechanisms), while still being able to operate the facility at a hotter temperature and save on cooling energy to use more of the energy budget for computing. The *overcooling* thermal management technique performs better for the *consolidation* technique than *LBBN* or *LBBR* because the nodes that the *consolidation* technique consolidates the workload on (primarily those in quadrant 1 in Fig. 5.2) happen to also, in general, consume less energy than the other nodes. We leave the study of optimal server placement within the facility as future work.

Another interesting observation is that, when given a large amount of energy budget (e.g., 16000 MJ), Fig. 5.4 (a) and (c) show that using the greedy online mapping technique in combination with the *overcooling* thermal management technique still slightly outperforms the comparison heuristics that use *overcooling*, even though those techniques assign all tasks in P-state P0 and the system has a large enough energy budget to finish the whole day. This is because the comparison techniques do not exploit any form of heterogeneity. Thus, even when executing a task in P0, a task may still miss its deadline when executed on a low-performance node if the task had to wait in the unmapped queue for awhile until a mapping event was triggered. One last observation from Fig. 5.4 (a) is that the *throttling* thermal management technique performs very poorly when paired with the *greedy* online mapping technique. This is because the *greedy* mapping technique assigns tasks to execute in the lowest-energy P-state that still hits the deadline. As a result, if any throttling occurs from overheating, tasks will very likely miss their deadlines.

Fig. 5.4 (e) displays the comparison of reward earned by techniques across several energy budget levels when using the sinusoidal arrival rate pattern. It is important to recall that this experiment used the real trace from the MetaCentrum2 system from the Parallel Workload Archive [92], and even though the computing systems are similar in size, the trace has a significantly lower arrival rate for tasks compared to the *constant* and *bursty* arrival rate patterns that were generated. As such, much less reward can be earned (as evidenced by the y-axis of Fig. 5.4 (e) in the magnitude of thousands compared to Fig. 5.4 (a) and (c) that shows reward in millions). Therefore, given enough energy, all tasks can easily complete by their deadlines, as shown by the reward earned by all techniques when given an energy budget of at least 14000 MJ. Also, the *throttling* technique outperforms the *overcooling* technique for all mapping heuristics, and this is because even though the facility is at a higher

temperature for the *throttling* approach (28°C), the small number of tasks being executed using this workload trace does not cause the compute nodes to generate much heat and trigger the throttling mechanisms.

We examine the energy consumption of the facility when using the different techniques for the constant workload arrival rate pattern in Fig. 5.4 (b), the bursty pattern in (d), and the sinusoidal pattern in (f). At energy budgets of only 8,000 MJ and 10,000 MJ, all techniques use all of the available energy budget before the day ends. However we can see in Fig. 5.4 (a) and (c) that our *greedy* online mapping heuristic in combination with the *templates* thermal management technique makes the best use of the limited amount of energy and earns significantly greater reward than the others. When using the *overcooling* thermal management approach, the facility requires nearly 16000 MJ of energy to last the entire day without deactivating due to exceeding the energy budget. We can also observe in Fig. 5.4 (b), (d), and (f) that when using our offline-assisted online mapping framework (i.e., the greedy mapping technique in combination with the templates thermal management approach), energy can be saved compared to the *overcooling* technique when given a larger energy budget (e.g., 16,000 MJ). The *templates* technique uses slightly more energy than *throttling* at higher energy budget levels, but the reward earned using the *throttling* technique is approximately half that of when using the *templates* technique.

We further analyze the different thermal management techniques by comparing the total instantaneous power consumption over time in Fig. 5.5 using the *bursty* arrival rate pattern. For this simulation, we set the energy budget to 12,000 MJ. The first observation is that the power consumption increases and decreases to match the arrival rate of tasks. The reason is two-fold: (a) more tasks are being executed during high arrival rate periods and therefore more compute power is consumed, and (b) the increase in compute power causes the facility

to increase in temperature (including the inlet temperature of the CRAC unit), causing the CRAC units to consume more power as well (see Eqn. 48).

As expected, when using the *overcooling* technique, the power consumption is much higher than the other thermal management techniques. Consequently, the facility exhausts its allotted energy budget before the day ends (at about 71,000 seconds, or 19.5 hours). Comparing the *throttling* and *templates* techniques, we can see the *templates* technique actually consumes slightly less power at most times throughout the day. Thus, when using the *templates* technique, the system is able to continue executing tasks throughout the entire day whereas when using the *throttling* technique, the energy budget is exhausted early, i.e., at approximately 23.5 hours. The reason is that the *templates* technique limits the allowable core vector (*ACV*) to which tasks can be mapped and sets a near-ideal CRAC temperature to sufficiently cool the nodes housing those cores. This does, however, result in slightly more tasks being dropped (line 1 of Alg. 6) from the system during periods of high arrival rates when using the *templates* technique compared to the others as there are more tasks in the unmapped batch than idle allowable cores to assign tasks (line 6 of Alg. 6). However, there is still a significant benefit to being able to set a near-ideal CRAC thermostat temperature for an *ACV* when using the *templates* technique, as cooling energy can be saved compared to *overcooling* (resulting in more energy available for computing), without triggering the throttling mechanisms, as what occurs when using the *throttling* technique. The significance of these benefits was quantified in Fig. 5.4 (b).

5.8. RELATED WORK

Thermal management of HPC systems and data centers is an important and active research topic. The first explorations into studying thermal management in HPC facilities

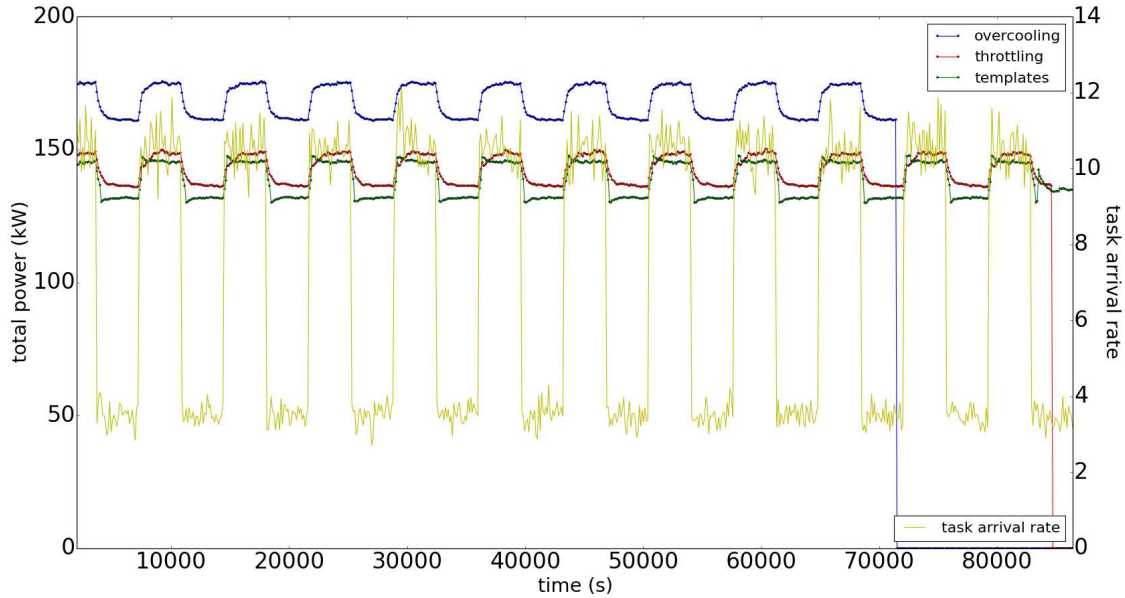


FIGURE 5.5. Using the *bursty* workload arrival pattern, we compare total power consumption of the three thermal management techniques when using the *greedy* runtime mapping heuristic over the course of a day. The energy budget was set to 12,000 MJ.

focused on modeling temperatures and heat. Moore et al. [74] introduced the notion of heat recirculation among compute nodes and proposed a technique that reduces heat recirculation and cooling costs. Due to the inefficiency of using CFD simulations to conduct thermal evaluations, Tang et al. [96] proposed an abstract heat flow model that determines coefficients arranged into a matrix that denote percentage of heat that is transferred from any node in the system to all other nodes in the system. Therefore, by knowing the power consumption, fan speed, density of air, and specific heat of air (and thus heat output) of each of the compute nodes, it was possible to calculate node temperatures in the steady-state without performing a full CFD simulation. Later, a transient thermal model was proposed by Jonas et al. [95] that also gives temporal information, that is, how the temperatures evolve throughout time rather than assuming steady-state. Our work is different from these

prior works as it focuses on resource management rather than modeling, and proposes runtime resource management rather than steady-state as in [74].

Several thermal-aware works have been published that focus on reducing cooling power or energy [102, 103, 100, 61]. In [102], CFD data was studied to find that an effective thermal-aware technique to reduce cooling power is to migrate workload from the node with the highest inlet air temperature. The research of [103] minimized cooling energy by identifying “better cooled” areas of the data center and consolidating more virtual machines (VMs) in those areas and allowing other unused servers to be deactivated. In [100], offline resource management techniques, specifically a genetic algorithm and quadratic programming approach, were proposed to minimize cooling energy. Cooling costs were reduced by minimizing hotspots and total heat generated in [61]. Managing resources to directly minimize cooling power and energy is important, however it is also important to consider performance in HPC and data center environments. Our work studies the problem of maximizing reward earned within a daily energy budget for both computing and cooling systems. As such, we consider the performance of the HPC system as an important objective in our framework in addition to cooling energy. We differentiate further from [61] by considering heterogeneous compute nodes and floor vent opening control.

Performance is also considered as a design objective in some thermal-aware works [56, 62, 68, 1, 104]. The research in [56] considers both performance optimization under a power constraint and power optimization under a performance constraint in a steady-state offline framework. Trade-offs between energy cost savings and workload delay can be exploited using the Stochastic Cost Minimization Algorithm technique proposed in [62]. Application performance, energy consumption, and thermal imbalance were simultaneously optimized

for a heterogeneous HPC system in [68], using fuzzy-based priorities for multi-objective optimization. *CoolBudget* [1] proposed an algorithm to maximize “fair speedup,” a metric that combines instructions-per-second (throughput) with fairness, within a power constraint and temperature thresholds for nodes. The research in [104] studied power management in an internet data center to minimize power of both compute and cooling systems while meeting quality of service (QoS) constraints on latency for servicing web requests. We differentiate from [56] by considering a runtime resource management framework rather than offline steady-state. The research in [62] does not consider heterogeneity, DVFS, heat recirculation, floor vent opening control, or CRAC unit control as we do in our work. DVFS, heat recirculation, floor vent opening control, and CRAC unit control is not considered in [68]. Our work considers floor vent opening control as well as tasks with different priority levels (reward) and deadlines rather than throughput optimization as in [1]. The problem studied in [104] is different than our work, by minimizing power of both computing and cooling systems under latency constraints in an internet data center environment. We also differ from [104] by considering floor vent opening control and a node power model that is dependent on the type of workload being executed.

5.9. CONCLUSIONS

We studied the problem of maximizing the reward collected for completing tasks by their deadlines subject to a daily energy budget and thermal constraints for heterogeneous high-performance computing systems. We proposed an offline-assisted runtime resource management framework to solve this problem, where a number of templates are generated offline and stored in a database to assist the runtime resource manager make thermal-aware decisions based on the incoming workload and state of the HPC facility. We compared our framework with three comparison techniques that use either an *overcooling* approach to thermal

management, or operating the facility at a higher temperature and relying on throttling for thermal management.

The primary contribution of this research was our offline-assisted runtime resource management framework, that included the greedy runtime mapping heuristic and the offline optimization method for generating templates. We also performed analysis and comparison of our framework with three schemes for runtime mapping, and *overcooling* and *throttling* approaches for thermal management. We showed how our framework can greatly increase the reward earned within a daily energy budget compared to the other techniques by intelligently providing enough cooling to avoid triggering the throttling mechanisms, but operating the facility hot enough to also save on cooling energy, allowing more of the energy budget to be spent on computing. Possible directions for future studies in this area are presented in Chapter 6.

CHAPTER 6

SUMMARY OF THESIS

The goal of this thesis was to address challenges (e.g., the prohibitively high cost of energy consumption in large HPC systems) and examine trade-offs in performance, power dissipation, and energy consumption for high-performance computing systems and the facilities in which they operate. Such challenges in these domains inhibit the progression of today's HPC systems to exascale levels, and providing solutions to such problems are of utmost importance to continue providing scientific researchers and consumers of cloud computing services with the compute speed necessary to solve challenging problems and experience highly-efficient computing. A combination of innovative ideas and solutions from numerous disciplines, such as computer architecture, thermodynamics, software tool development, reliability, security, and resource management will be required to overcome the hurdles to building an exascale system. In this thesis, we proposed solutions in the resource management domain that are robust, thermal-aware, power-aware, and energy-aware, and would be of great use to the HPC field as researchers push towards exascale computing.

In Chapter 2 we studied trade-offs between performance and energy in an offline resource management scenario, and proposed heuristic-based resource management solutions for a heterogeneous HPC system that considers performance as the objective and energy consumption as the constraint, and vice versa. An important aspect of this study was the assumption that applications had varying execution times (e.g., due to cache misses or data dependent execution times), a realistic and important consideration. Therefore, execution times of applications were modeled using probability distributions instead of scalar values,

and the resource management solutions we proposed used information from these probability distributions to make intelligent decisions.

We continued to study trade-offs between performance and power consumption in Chapter 3, except in a rate-based environment where power dissipation (instead of energy consumption) was examined due to its rate-based nature (power can be measured in joules per second). Two important additions were considered in this work: (a) co-location interference that causes performance degradation among multiple applications executed simultaneously on cores that share resources, and (b) the impact that the cooling system has on the HPC facility's power consumption. We assumed the resource manager had control of the thermostat temperatures of the air conditioning units to assist in making thermal-aware decisions. The proposed resource management solutions considered the thermal relationship among the compute system and cooling system to optimize performance of the system when limited in the amount of power that can be consumed by the facility.

We extended the rate-based resource management work to a geo-distributed scale in Chapter 4, and proposed methods that minimize monetary energy cost. The resource management techniques that we proposed exploited time-of-use pricing and the assumption that renewable energy sources were located at different data centers to intelligently distribute workload to data centers that currently had inexpensive electricity pricing or an abundance of renewable energy to reduce costs.

Finally, the research in Chapter 5 moved from rate-based to runtime (online) resource management. Making intelligent thermal-aware resource management decisions to reduce the energy consumption of the cooling system can require complex thermal models that take a significant amount of time to compute. This can be prohibitive in online decision making

to ensure that newly-arrived tasks begin execution as soon as possible to avoid missing deadlines. Therefore, we proposed a novel offline-assisted online resource management framework that uses a database of offline-generated solutions to help make thermal-aware decisions.

CHAPTER 7

FUTURE WORK

In Chapter 2, we designed four energy-aware heuristics for assigning bags-of-tasks to a heterogeneous computing system in a robust manner. This study revealed great potential for our Tabu Search and genetic algorithm with local search (GALS) heuristics when combined with a dynamic penalty function for managing compute resources in an energy-aware manner for deadline-constrained and energy-constrained systems. For future work in this area, it would be interesting to consider workloads that consist of mostly compute-intensive or mostly memory-intensive tasks, which may require different techniques in addition to DVFS (such as consolidation) to reduce energy consumption. Introducing tasks of varying memory intensities would not only affect power consumption, but also co-location interference effects. It would be interesting to study these effects in a bag-of-tasks type of environment, and designing models and resource allocation techniques that account for these effects would be a necessity.

Chapter 3 studies maximizing reward earned for completing tasks by their deadlines in a power and thermal constrained environment. We introduce a reward rate performance measure that incorporates co-location interference. We design novel heuristics to maximizing that performance measure, while also ensuring the system remains within imposed power and thermal constraints. These heuristics include a non-linear programming (NLP) technique that considers co-location interference, a greedy technique that assigns task types to their most power-efficient node type and P-state, and a genetic algorithm enhanced with local search. For future work, it would be interesting to account for uncertainties in task execution times that can vary due to changes in the data inputs or network load, and design

co-location and thermal-aware resource management techniques that are robust against such uncertainties to more closely match real-world problems. Also, classifying unknown tasks into task types as they arrive would be an interesting addition to this work.

Chapter 4 proposed three heuristics for workload allocation across geographically distributed data centers. We explored adding different levels of knowledge of the system, particularly co-location interference, to geographical workload distribution algorithms. We also demonstrated that including additional information about the co-location interference in the decision process of the heuristics resulted in a lower energy cost by reducing or eliminating node over-provisioning while still meeting all required workload execution rates. For future work in this area, it would be interesting to study the co-location, thermal, and energy aware geo-distributed load distribution problem in a runtime dynamic environment rather than rate-based over epoch time intervals. Also, considering energy storage (e.g., batteries, thermal energy storage) at individual data centers would be interesting and create difficult workload allocation decisions due to the relationships between time-of-use pricing, renewable energy, and energy storage.

In Chapter 5, we studied the problem of maximizing the reward collected for completing tasks by their deadlines subject to a daily energy budget and thermal constraints for heterogeneous high-performance computing systems. We proposed an offline-assisted runtime resource management framework to solve this problem, where a number of templates are generated offline and stored in a database to assist the runtime resource manager make thermal-aware decisions based on the incoming workload and state of the HPC facility. We have a few directions that would be interesting to pursue relating to this research. First, performing further analyses of our framework by performing a sensitivity study on the discretization resolutions of our templates, examining different mapping event and thermal

management interval times, and performing scalability analyses by also examining a larger HPC facility with more compute nodes and CRAC units. Second, the different placements of the heterogeneous nodes within the facility can impact temperatures and therefore have different cooling requirements. Performing sensitivity analyses of different node placements, and optimizing node placements, are interesting directions for this research. Finally, we wish to examine and model uncertainties in the system, and then design robust resource management techniques that mitigate the impact of those uncertainties.

There are many challenges and research opportunities in the goal of building an exascale computing system. High energy consumption is one challenge that we have examined. However, other aspects such as reliability, resiliency, new programming models, communication technologies, and hardware architectures are all critically important research topics that have challenges that become exacerbated at large scales. New intelligent methods for check pointing applications and recovering from faults are increasingly important with the rapid number of failures that occur in exascale-sized HPC systems. Enhanced tools for parallel programming will be required to enable users to take advantage of a complex exascale system. New advances in both on-chip communications and hardware design will be required to obtain the performance and energy consumption levels needed for exascale computing.

Because cooling systems and thermal issues are of great interest, examining the trade-offs between energy consumption, temperature, and reliability at large scales would be extremely interesting and valuable. Typically, higher temperatures result in less-reliable systems or can even shorten the lifespans of the computer hardware. However, providing enough cooling to maintain low operating temperatures consumes a significant amount of energy and therefore causes high operating expenditures. As HPC systems continue to grow, more power will be

dissipated from compute nodes and reliability will diminish. It would extremely interesting to study such trade-offs and design new technologies that solve such challenges.

BIBLIOGRAPHY

- [1] O. Tuncer, K. Vaidyanathan, K. Gross, and A. K. Coskun, “Coolbudget: Data center power budgeting with workload and cooling asymmetry awareness,” in *32nd Int’l Conf. Computer Design (ICCD ’14)*, 2014, pp. 497–500.
- [2] W. Feng, “The green500 list - november 2015,” Nov. 2015, accessed 29 Dec. 2015. [Online]. Available: <http://www.green500.org/news/green500-list-november-2015>
- [3] E. Strohmaier, J. Dongarra, H. Simon, M. Meuer, and H. Meuer, “The TOP 500 list,” 2015. [Online]. Available: <http://www.top500.org/lists/2015/11/>
- [4] 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)*, 2014, 10 pp.
- [5] D. Filani, J. He, S. Gao, M. Rajappa, A. Kumar, P. Shah, and R. Nagappan, “Dynamic data center power management: Trends, issues, and solutions,” *Intel Technology Journal*, vol. 12, no. 1, pp. 59–67, Feb. 2008.
- [6] P. Kogge *et al.*, “Exascale computing study: Technology challenges in achieving exascale systems,” DARPA, Tech. Rep., Sep. 2008.
- [7] United States Energy Information Administration, “Electric power monthly,” 2015. [Online]. Available: http://www.eia.gov/electricity/monthly/epm_table_grapher.cfm?t=epmt_5_6_a
- [8] M. Oxley, S. Pasricha, A. A. Maciejewski, H. J. Siegel, J. Apodaca, D. Young, L. D. B. no, J. Smith, S. Bahirat, B. Khemka, A. Ramirez, and Y. Zou, “Makespan and energy robust stochastic static resource allocation of a bag-of-tasks to a heterogeneous

- computing system,” *IEEE Trans. Parallel and Distributed Systems*, vol. 26, no. 10, pp. 2791–2805, Oct. 2015.
- [9] M. Oxley, S. Pasricha, H. J. Siegel, and A. A. Maciejewski, “Energy and deadline constrained robust stochastic static resource allocation,” in *Workshop on Power and Energy Aspects of Computation (PEAC '13)*, Sept. 2013, 10 pp.
- [10] J. Koomey, “Growth in data center electricity use 2005 to 2010,” Analytics Press, Tech. Rep., 2011, <http://www.analyticspress.com/datacenters.html>. Accessed 22 Aug. 2012.
- [11] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, “The cost of a cloud: Research problems in data center networks,” *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 68–73, Jan. 2009.
- [12] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation Std., *Advanced Configuration and Power Interface Specification*, 2011, rev. 5.0, <http://www.acpi.info>. Accessed 28 Aug. 2012.
- [13] 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 Computing*, vol. 61, no. 6, pp. 810–837, June 2001.
- [14] G. Ritchie and J. Levine, “A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments,” in *3rd Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG '04)*, Dec. 2004.
- [15] H. J. Choi, D. O. Son, S. G. Kang, J. M. Kim, H.-H. Lee, and C. H. Kim, “An efficient scheduling scheme using estimated execution time for heterogeneous computing systems,” *J. Supercomput.*, vol. 65, no. 2, pp. 886–902, Aug. 2013.

- [16] A. Khokhar, V. Prasanna, M. Shaaban, and C.-L. Wang, “Heterogeneous computing: Challenges and opportunities,” *IEEE Computer*, vol. 26, no. 6, pp. 18–27, June 1993.
- [17] D. Xu, K. Nahrstedt, and D. Wichadakul, “QoS and contention-aware multi-resource reservation,” *Cluster Computing*, vol. 4, no. 2, pp. 95–107, Apr. 2001.
- [18] W. Yuan and K. Nahrstedt, “Energy-efficient soft real-time CPU scheduling for mobile multimedia systems,” *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 149–163, Dec. 2003.
- [19] D. Li and J. Wu, “Energy-aware scheduling for frame-based tasks on heterogeneous multiprocessor platforms,” in *41st Int’l Conf. on Parallel Processing (ICPP ’12)*, Sept. 2012, pp. 430–439.
- [20] J. F. Pineau, Y. Robert, and F. Vivien, “Energy-aware scheduling of bag-of-tasks applications on master-worker platforms,” *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 145–157, Feb. 2011.
- [21] F. Pinel, J. Pecero, S. Khan, and P. Bouvry, “Energy-efficient scheduling on milliclusters with performance constraints,” in *IEEE/ACM Int’l Conf. on Green Computing and Communications (GreenCom ’11)*, Aug. 2011, pp. 44–49.
- [22] L. Briceño, H. J. Siegel, A. A. Maciejewski, M. Oltikar, J. Brateman, J. White, J. Martin, and K. Knapp, “Heuristics for robust resource allocation of satellite weather data processing onto a heterogeneous parallel system,” *IEEE Trans. Parallel and Distributed Systems*, vol. 22, no. 11, pp. 1780–1787, Nov. 2011.
- [23] L. Briceño, J. Smith, H. J. Siegel, A. A. Maciejewski, P. Maxwell, R. Wakefield, A. Al-Qawasmeh, R. C. Chiang, and J. Li, “Robust static resource allocation of DAGs in a heterogeneous multicore system,” *J. Parallel and Distributed Computing*, vol. 73, no. 12, pp. 1705–1717, Dec. 2013.

- [24] F. M. Ciorba, T. Hansen, S. Srivastava, I. Banicescu, A. A. Maciejewski, and H. J. Siegel, “A combined dual-stage framework for robust scheduling of scientific applications in heterogeneous environments with uncertain availability,” in *21st Heterogeneity in Computing Workshop (HCW '12)*, May 2012, pp. 187–200.
- [25] Y.-S. Kee, K. Yocum, A. A. Chien, and H. Casanova, “Robust resource allocation for large-scale distributed shared resource environments,” in *15th Int’l Symp. on High Performance Distributed Computing (HPDC '06)*, June 2006, pp. 341–342.
- [26] B. D. Young, J. Apodaca, L. D. Briceño, J. Smith, S. Pasricha, A. A. Maciejewski, H. J. Siegel, B. Khemka, S. Bahirat, A. Ramirez, and Y. Zou, “Deadline and energy constrained dynamic resource allocation in a heterogeneous computing environment,” *J. Supercomput.*, vol. 63, no. 2, pp. 326–347, Feb. 2013.
- [27] CSU Information Science and Technology Center, 2013, iSTeC Cray High Performance Computing (HPC) System, http://istec.colostate.edu/istec_cray. Accessed 17 Sep. 2013.
- [28] D. Meisner, B. T. Gold, and T. F. Wenisch, “Powernap: Eliminating server idle power,” *SIGPLAN Not.*, vol. 44, no. 3, pp. 205–216, Mar. 2009.
- [29] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach, Fifth Edition*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.
- [30] 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.
- [31] 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 Computing*, vol. 44, no. 1, pp. 33–52, July 1997.
- [32] 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 Computing*, vol. 68, no. 8, pp. 1157–1173, Aug. 2008.
- [33] S.-G. Kim, C. Choi, H. Eom, H. Y. Yeom, and H. Byun, “Energy-centric DVFS controlling method for multi-core platforms,” in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, June 2012, pp. 685–690.
- [34] S. Ali, A. A. Maciejewski, and H. J. Siegel, “Perspectives on robust resource allocation for heterogeneous parallel systems,” 2008, *Handbook of Parallel Computing: Models, Algorithms, and Applications*, S. Rajasekaran and J. Reif, Ed., Chapman & Hall/CRC Press, Boca Raton, FL, pp. 41.1-41.30.
- [35] J. L. Hodges and E. L. Lehmann, *Basic Concepts of Probability and Statistics*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2005.
- [36] O. H. Ibarra and C. E. Kim, “Heuristic algorithms for scheduling independent tasks on nonidentical processors,” *J. Assoc. Comput. Mach.*, vol. 24, no. 2, pp. 280–289, Apr. 1977.
- [37] 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 Computing*, vol. 59, no. 2, pp. 107–131, Nov. 1999.
- [38] W. Sun and T. Sugawara, “Heuristics and evaluations of energy-aware task mapping on heterogeneous multiprocessor platforms,” in *Advanced Parallel and Distributed Computing Models (APDCM ’11)*, May 2011, pp. 599–607.
- [39] F. Glover, “Tabu search, part I,” *ORSA Journal on Computing* 1, 1989.

- [40] A. Doğan and F. Ozgüner, “Genetic algorithm based scheduling of meta-tasks with stochastic execution times in heterogeneous computing systems,” *Cluster Computing*, vol. 7, no. 2, pp. 177–190, Apr. 2004.
- [41] 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.
- [42] D. Powell and M. M. Skolnick, “Using genetic algorithms in engineering design optimization with non-linear constraints,” in *5th Int’l Conf. on Genetic Algorithms*, 1993, pp. 424–431.
- [43] H.-P. Schwefel, *Evolution and Optimum Seeking*. Wiley Interscience, New York, NY, 1995.
- [44] A. E. Smith and D. W. Coit, “Constraint-handling techniques - penalty functions,” in *Handbook of Evolutionary Computation*, T. Back, D. Fogel, and Z. Michalewicz, Eds. Institute of Physics Publishing and Oxford University Press, Bristol, UK, 1997, ch. C5.2.
- [45] O. Yeniay, “Penalty function methods for constrained optimization with genetic algorithms,” *Mathematical and Computational Applications*, vol. 10, no. 1, pp. 45–56, Jan. 2005.
- [46] Standard Performance Evaluation Corporation (SPEC), 2008, SPECpower_ssj2008, http://www.spec.org/power_ssj2008. Accessed 28 Mar. 2013.
- [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 Computing Workshop (HCW ’11)*, May 2011, pp. 33–43.

- [48] B. Khemka, R. Friese, S. Pasricha, A. A. Maciejewski, H. J. Siegel, G. Koenig, S. Powers, M. Hilton, R. Rambharos, and S. Poole, “Utility maximizing dynamic resource management in an oversubscribed energy-constrained heterogeneous computing system,” *Sustainable Computing: Informatics and Systems*, vol. 5, pp. 14–30, 2015.
- [49] W. Feng, “The Green500 list - November 2014,” 2014. [Online]. Available: <http://www.green500.org/lists/green201411>
- [50] A. T. Committee, “2011 Thermal Guidelines for Data Processing Environments - Expanded Data Center Classes and Usage Guidance,” American Society of Heating, Refrigerating, and Air-Conditioning Engineers, Inc., Tech. Rep., 2011.
- [51] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam, “Cuanta: Quantifying effects of shared on-chip resource interference for consolidated virtual machines,” in *Proceedings of the 2Nd ACM Symposium on Cloud Computing (SOCC '11)*, Oct. 2011, pp. 1–14.
- [52] 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 multicore computing,” in *17th Workshop on Advances on Parallel and Distributed Computational Models (APDCM '15)*, 2015, accepted, to appear.
- [53] P. Bodik, R. Griffith, C. Sutton, A. Fox, M. I. Jordan, and D. A. Patterson, “Automatic exploration of datacenter performance regimes,” in *1st Workshop on Automated Control for Datacenters and Clouds*, 2009, pp. 1–6.
- [54] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, “Energy-aware server provisioning and load dispatching for connection-intensive internet services,” in *5th USENIX Symp. Networked Systems Design and Implementation*, 2008, pp. 337–350.

- [55] D. G. Feitelson, "Parallel workload archive," 2009. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/1_anl_int/index.html
- [56] 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. Computers*, accepted 2013, to appear.
- [57] S. K. S. Gupta, A. Banerjee, Z. Abbasi, G. Varsamopoulos, M. Jonas, J. Ferguson, R. Gilbert, and T. Mukherjee, "GDCSim - A Simulator for Green Data Center Design and Analysis," *ACM Trans. Modeling and Comp. Simulation*, vol. 24, no. 1, 2014, 37 pp.
- [58] Google, "Efficiency: How we do it," <http://www.google.com/about/datacenters/efficiency/internal>, Accessed 20 Nov. 2015.
- [59] M. Stansberry, "2014 data center industry survey," 2014. [Online]. Available: <https://journal.uptimeinstitute.com/2014-data-center-industry-survey/>
- [60] M. T. Chaudhry, T. C. Ling, A. Manzoor, S. A. Hussain, and J. Kim, "Thermal-aware scheduling in green data centers," *ACM Comp. Surveys*, vol. 47, no. 3, 2015, 48 pp.
- [61] V. Villebonnet and D. D. Costa, "Thermal-aware cloud middleware to reduce cooling needs," in *23rd Int'l WETICE Conf. (WETICE '14)*, 2014, pp. 115–120.
- [62] Y. Guo, Y. Gong, Y. Fang, P. P. Khargonekar, and X. Geng, "Energy and network aware workload management for sustainable data centers with thermal storage," *IEEE Trans. Parallel and Distributed Systems*, vol. 25, no. 8, pp. 2030–2042, 2014.
- [63] Z. Jiang, W. Huang, I. You, Z. Qian, and S. Lu, "Thermal-aware task placement with dynamic thermal model in an established datacenter," in *8th Int'l Conf. Innovative Mobile and Internet Services in Ubiquitous Comp. (IMIS '14)*, 2014, pp. 1–8.

- [64] F. Kaplan, J. Meng, and A. K. Coskun, “Optimizing communication and cooling costs in data centers via intelligent job allocation,” in *4th Int’l Green Comp. Conf. (IGCC ’13)*, 2013, 10 pp.
- [65] E. K. Lee, H. Viswanathan, and D. Pompili, “VMAP: Proactive thermal-aware virtual machine allocation in HPC cloud datacenters,” in *19th Int’l Conf. on High Performance Comp. (HiPC ’12)*, 2012, 10 pp.
- [66] S. Nesmachnow, C. Perfumo, and I. Goiri, “Controlling datacenter power consumption while maintaining temperature and qos levels,” in *3rd Int’l Conf. Cloud Networking (CloudNet ’14)*, 2014, pp. 242–247.
- [67] 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, 2014.
- [68] H. Sun, P. Stolf, J. M. Pierson, and G. D. Costa, “Multi-objective scheduling for heterogeneous server systems with machine placement,” in *14th Int’l Symp. Cluster, Cloud, and Grid Comp. (CCGRID ’14)*, 2014, pp. 334–343.
- [69] O. Tuncer, K. Vaidyanathan, K. Gross, and A. K. Coskun, “Coolbudget: Data center power budgeting with workload and cooling asymmetry awareness,” in *32nd Int’l Conf. Computer Design (ICCD ’14)*, 2014, pp. 497–500.
- [70] M. Kambadur, T. Mosely, R. Hank, and M. A. Kim, “Measuring interference between live datacenter applications,” in *Int’l Conf. High Performance Comp., Networking, Storage, and Analysis (SC ’12)*, 2012, 12 pp.
- [71] J. Zhao, X. Feng, H. Cui, Y. Yan, J. Xue, and W. Yang, “An empirical model for predicting cross-core performance interference on multicore processors,” in *22nd Int’l*

- Conf. Parallel Architectures and Compilation Techniques (PACT '13)*, 2013, pp. 201–212.
- [72] C. Delimitrou and C. Kozyrakis, “Quality-of-service-aware scheduling in heterogeneous datacenters with paragon,” *IEEE Micro*, vol. 34, no. 3, pp. 17–30, 2014.
- [73] S. Blagodurov, D. Gmach, M. Arlitt, Y. Chen, C. Hyser, and A. Fedorova, “Maximizing server utilization while meeting critical SLAs via weight-based collocation management,” in *2013 Int’l Symp. Integrated Network Management (IM '13)*, 2013, pp. 277–285.
- [74] J. Moore, J. Chase, P. Ranganathan, and R. Sharma, “Making scheduling ”cool“: Temperature-aware workload placement in data centers,” in *Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC '05)*, 2005.
- [75] H. Bhagwat, U. Singh, A. Deodhar, A. Singh, A. Vasani, and A. Sivasubramanian, “Fast and accurate evaluation of cooling in data centers,” *J. Elect. Packaging*, vol. 137, no. 1, 2015, 9 pp.
- [76] A. Nemirovski, “Interior point polynomial time methods in convex programming,” 2004. [Online]. Available: http://www2.isye.gatech.edu/~nemirovs/Lect_IPM.pdf
- [77] BASX Solutions, “CRAC / CRAH Units,” <http://www.basxsolutions.com/--crac-crah-units>, 2015.
- [78] S. Gondipalli, S. Bhojpe, B. Sammakia, M. Iyengar, and R. Schmidt, “Effect of isolating cold aisles on rack inlet temperature,” in *11th Conf. Thermal and Thermomechanical Phenomena in Electronic Systems (ITHERM '08)*, 2008, pp. 1247–1254.
- [79] E. Jonardi, M. A. Oxley, S. Pasricha, H. J. Siegel, and A. A. Maciejewski, “Energy cost optimization for geographically distributed heterogeneous data centers,” in *Workshop on Energy-efficient Networks of Computers (E2NC '15) in the proceedings of the Sixth*

International Green and Sustainable Computing Conference (IGSC '15), Dec. 2015, accepted, to appear.

- [80] “Data center locations,” <http://www.google.com/about/datacenters/inside/locations/index.html>.
- [81] Y. K. Kwok, A. A. Maciejewski, H. J. Siegel, I. Ahmad, and A. Ghafoor, “A semi-static approach to mapping dynamic iterative tasks onto heterogeneous computing systems,” *J. Parallel and Distributed Comp.*, vol. 66, no. 1, pp. 77–98, 2006.
- [82] H. Goudarzi and M. Pedram, “Geographical load balancing for online service applications in distributed datacenters,” in *CLOUD '13*, June 2013, pp. 351–358.
- [83] J. Zhao *et al.*, “Dynamic pricing and profit maximization for the cloud with geo-distributed data centers,” in *INFOCOM '14*, Apr. 2014, pp. 118–126.
- [84] L. Gu *et al.*, “Optimal task placement with QoS constraints in geo-distributed data centers using DVFS,” *IEEE Trans. Comp.*, vol. 64, no. 7, pp. 2049–2059, June 2015.
- [85] H. Xu, C. Feng, and B. Li, “Temperature aware workload management in geo-distributed data centers,” *IEEE Trans. Parallel and Distributed Systems*, vol. 26, no. 6, pp. 1743–1753, May 2015.
- [86] M. Polverini *et al.*, “Thermal-aware scheduling of batch jobs in geographically distributed data centers,” *IEEE Trans. Cloud Comp.*, vol. 2, no. 1, pp. 71–84, Apr. 2014.
- [87] D. Mehta, B. O’Sullivan, and H. Simonis, “Energy cost management for geographically distributed data centres under time-variable demands and energy prices,” in *UCC '13*, Dec. 2013, pp. 26–33.
- [88] L. Gu *et al.*, “Joint optimization of VM placement and request distribution for electricity cost cut in geo-distributed data centers,” in *ICNC '15*, Feb. 2015, pp. 717–721.

- [89] X. Deng *et al.*, “Eco-aware online power management and load scheduling for green cloud datacenters,” *IEEE Sys. Journal*, no. 99, pp. 1–10, 2014.
- [90] 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.
- [91] H. Viswanathan, E. Lee, I. Rodero, D. Pompili, M. Parashar, and M. Gamell, “Energy-aware application-centric VM allocation for HPC workloads,” in *25th Int’l Parallel and Distributed Processing Symp. (IPDPS ’11)*, 2011, pp. 890–897.
- [92] D. G. Feitelson, “Parallel workload archive,” 2015. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/l_metacentrum2/index.html
- [93] H. Lin, X. Qi, S. Yang, and S. Midkiff, “Workload-driven VM consolidation in cloud data centers,” in *29th Int’l Parallel and Distributed Processing Symp. (IPDPS ’15)*, 2015, pp. 207–216.
- [94] J. Whitney and P. Delforge, “Data center efficiency assessment,” Tech. Rep., 2014, <http://www.nrdc.org/energy/files/data-center-efficiency-assessment-IP.pdf>.
- [95] M. Jonas, R. R. Gilbert, J. Ferguson, G. Varsamopoulos, and S. K. S. Gupta, “A transient model for data center thermal prediction,” in *3rd Int’l Green Comp. Conf. (IGCC ’12)*, 2012, 10 pp.
- [96] 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 Conference on Intelligent Sensing and Information Processing (ICISIP ’06)*, Dec. 2006, pp. 203–208.

- [97] Hewlett Packard, “Hp opens new research facility to advance sustainable data center technologies,” 2011. [Online]. Available: http://www8.hp.com/us/en/hp-news/press-release.html?id=914457#.Vli5t_mrSHs
- [98] Artelys, “Artelys knitro,” 2015. [Online]. Available: <http://www.artelys.com/en/optimization-tools/knitro>
- [99] Adaptive Computing, “Node allocation policies (Moab workload manager),” 2015. [Online]. Available: <http://docs.adaptivecomputing.com/mwm/archive/6-0/5.2nodeallocation.php>
- [100] Q. Tang, S. Gupta, and G. Varsamopoulos, “Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach,” *IEEE Trans. Parallel and Distributed Systems*, vol. 19, no. 11, pp. 1458–1472, 2008.
- [101] P. University, “The PARSEC Benchmark Suite,” 2015. [Online]. Available: <http://parsec.cs.princeton.edu/overview.htm>
- [102] D. Demetriou and H. Khalifi, “Thermally aware, energy-based load placement in open-aisle, air-cooled data centers,” *J. Electronic Packaging*, vol. 135, no. 3, 2013, 14 pp.
- [103] E. K. Lee, H. Viswanathan, and D. Pompili, “VMAP: Proactive thermal-aware virtual machine allocation in HPC cloud datacenters,” in *19th Int’l Conf. on High Performance Comp. (HiPC ’12)*, 2012, 10 pp.
- [104] J. Yao, H. Guan, J. Luo, L. Rao, and X. Liu, “Adaptive power management through thermal aware workload balancing in internet data centers,” *IEEE Trans. Parallel and Distributed Systems*, vol. 26, no. 9, pp. 2400–2409, 2015.

- [105] K. Steers, “Hardware tips: Do the math to get your PC all the power it needs,” Feb. 2005, accessed 28 Mar. 2013. [Online]. Available: <http://www.pcworld.com/article/119585/article.html>
- [106] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and R. Bianchini, “Coscale: Coordinating CPU and memory system DVFS in server systems,” in *45th Ann. IEEE/ACM Int’l Symp. on Microarchitecture (MICRO ’12)*, Dec. 2012, pp. 143–154.
- [107] A. Tiwari, M. Laurenzano, J. Peraza, L. Carrington, and A. Snaveley, “Green queue: Customized large-scale clock frequency scaling,” in *2nd Int’l Conf. Cloud and Green Computing (CGC ’12)*, Nov. 2012, pp. 260–267.
- [108] J. Mora, “Understanding bulldozer architecture through linpack benchmark,” in *4th Ann. HPC Advisory Council European Conference*, June 2012, conference Presentation.
- [109] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann, “Power management architecture of the 2nd generation intel core microarchitecture,” in *Hot Chips 23 Symp.*, Aug. 2011, conference Presentation.
- [110] S. Ali, H. J. Siegel, M. Maheswaran, and D. Hensgen, “Representing task and machine heterogeneities for heterogeneous computing systems,” *Tamkang J. Science and Engineering, Special 50th Anniversary Issue*, vol. 3, no. 3, pp. 195–207, Nov. 2000.

APPENDIX A

POWER AND PERFORMANCE DATA COLLECTION AND CALCULATIONS FOR NODE TYPES FROM SPECPOWER

In this appendix that is supplementary to Chapter 2, we show how our system parameters were obtained from SPECpower_ssj2008 as well as details regarding P-state frequencies and voltages and workload generation. The small simulation platform uses 25 different servers, and the large simulation platform consists of ten copies of each of the 25 servers from the small simulation platform for a total of 250 nodes. The parameters we obtain from SPEC for our evaluations is shown in Table A.1. The number of cores per node (n_i) is based on the type of CPU used in each node, giving a total of 458 and 4,580 total cores

TABLE A.1. Data From SPECpower

	CPU Type	Total Cores (n_i)	Idle Power (Watts)	Memory DIMMs	Hard Drives	Add-on Cards
Node 1	Opteron 6278	64	176	32	2 HDD	2
Node 2	Xeon E3-1260L	4	17.8	2	1 SSD	0
Node 3	Opteron 6238	24	89.8	8	1 HDD	1
Node 4	Xeon E5645	12	63.1	4	1 HDD	0
Node 5	Xeon E7-4870	40	423	16	1 SSD	3
Node 6	Xeon E5-2470	8	52.7	2	1 SSD	1
Node 7	Xeon X3470	4	46.5	2	1 SSD	0
Node 8	Xeon E5-2660	16	56.5	4	1 SSD	1
Node 9	Xeon E3-1265L	4	18.9	2	1 SSD	0
Node 10	Xeon E3-1280	4	24.2	2	1 HDD	0
Node 11	Xeon X5675	12	55.6	4	1 SSD	0
Node 12	Opteron 6276	32	68.9	8	1 SSD	1
Node 13	Xeon X5675	12	125	6	1 SSD	0
Node 14	Xeon X3470	4	35.4	2	1 HDD	0
Node 15	Xeon E5-2660	64	331	8	1 SSD	1
Node 16	Xeon E7330	16	418	16	2 HDD	3
Node 17	Xeon X5670	24	101	6	1 SSD	0
Node 18	Xeon E3110	2	75.2	4	2 HDD	2
Node 19	Xeon E5-2440	12	55.1	4	1 SSD	1
Node 20	Xeon E5-2470	16	47.9	6	1 SSD	1
Node 21	Xeon E5-2660	16	59.5	6	1 SSD	2
Node 22	Xeon E3-1265L	4	15.7	2	1 SSD	0
Node 23	Xeon E5-2660	16	54	6	1 HDD	0
Node 24	Xeon E5-4640	32	92.7	12	1 SSD	1
Node 25	Xeon E5-2470	16	80.6	4	1 SSD	1

to schedule for the small and large simulation sizes, respectively. We calculate the power overhead from components (\mathbf{O}_i) by summing the power used by the memory, hard drives, and add-on cards [105]. We assume the *Idle Power* (see Table A.1) of a node consists of the power used by components and the static power of the cores, thus \mathbf{P}_i^{stat} is calculated by subtracting component overhead power (\mathbf{O}_i) from the Idle Power.

The total number of P-states for CPU cores in a node (\mathbf{PS}_i), as well as the frequencies and voltages of the P-states (\mathbf{f}_π and $\mathbf{Vdd}_j\pi$) were obtained from CPU manufacturer documents, experiments from the literature [106, 33, 107], and technical presentations [108, 109]. We used a uniform random variable to assign task types to tasks. The mean and variance values of the task execution times for each P-state in each node ($\mu(\mathbf{t}_{ij}^x, \mathbf{PS}(\mathbf{t}_{ij}^x))$ and $\mathbf{var}(\mathbf{t}_{ij}^x, \mathbf{PS}(\mathbf{t}_{ij}^x))$, respectively) were generated using the Coefficient of Variation (CoV) method from [110] and a scaling procedure. First, the mean values for all execution times on all nodes with cores in P-state P0 were generated using the CoV method, and then the values were scaled for other P-states depending on the task type of a task and the clock frequency. We assume tasks of type \mathbf{TT}_0 are memory intensive, and their mean execution times scale at a ratio of 25% of the frequency (e.g., halving the frequency results in only a 25% increase in execution time). Tasks of type \mathbf{TT}_{15} are compute-intensive, and their execution times scale at a ratio of 100% of the frequency (e.g., halving the frequency results in a 100% increase in execution time). The execution times of task types \mathbf{TT}_1 to \mathbf{TT}_{14} scale at ratios of 30% to 95% of the clock frequency. The variance values were generated in a similar fashion, using the CoV method and the same scaling procedure.

APPENDIX B

GLOBAL AND LOCAL SEARCH IN TABU SEARCH FOR ROBUST ENERGY-AWARE RESOURCE MANAGEMENT

In this appendix that is supplementary to Chapter 2, we examine the trade-offs between long-hops and short-hops in Tabu Search by experimenting with different termination criteria for local search that resulted in different numbers of long-hops versus short-hops over six hours of heuristic execution time for the MO-EC problem on the small simulation size (Figure B.1). We varied our local search termination criteria to stop short-hops and perform long-hops after 100, 200, 500, 1,000, and 1,500 consecutive short-hops with less than 0.1% improvement in makespan-robustness (as shown on x-axis of Figure B.1). This resulted in an average of 118, 38, 6, 2, and 1 long-hop(s) for the different criteria, respectively, as shown in red text in Figure B.1. We observed the best results when performing approximately six long-hops, showing a mixture of short-hops and long-hops is best in our environment, and thus we perform long-hops when 500 consecutive short-hops fail to improve the solution by 0.1%.

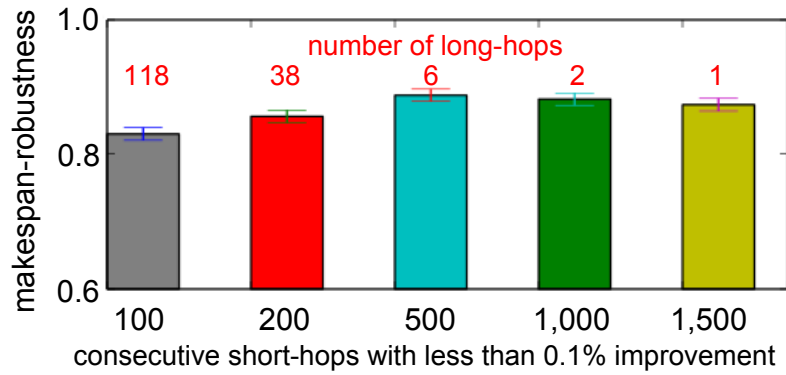


FIGURE B.1. Comparison of short-hop termination criteria for MO-EC using the small simulation size (25 nodes, 458 total cores, and 10,000 tasks) over six hours of heuristic execution time. The system deadline was set to 15,500 seconds, the energy budget was set to 58 MJ, and energy-robustness constraint was set to 90%. The red numbers indicate number of long-hops performed.

LIMITING THE SEARCH SPACE TECHNIQUE FOR ROBUST ENERGY-AWARE RESOURCE MANAGEMENT

In this appendix that is supplementary to Chapter 2, we describe our “limiting the search space” technique, then explain how a feasible initial population is generated for the GA and GALS heuristics. The “limiting the search space” technique simply rejects infeasible solutions [43]. This method requires that the individuals of the initial population are feasible in the GA and GALS, or that the initial solution is feasible in Tabu Search. Modifications to any solution must result in a feasible solution as well, or else the offspring are eliminated in the GA and GALS, or the solution resulting from a local search move is reverted to its previous (feasible) state in Tabu Search. This constrained optimization technique uses the same definitions to distinguish “better” solutions as the “superiority of feasible solutions” technique. To generate solutions that meet the makespan-robustness constraint (for EO-MC) in the GA and GALS, we start with all tasks placed in an unmapped batch, and considering the tasks in an arbitrary order, we find for each task the node that minimizes the *expected* execution time of the task in P-state 0. Within that node, the task is mapped to the core with maximum makespan-robustness if the assignment does not violate the constraint. If the constraint will be violated, the task is placed back into the unmapped set. Once all tasks have been attempted to be mapped to their minimum expected execution time node, all tasks left in the unmapped set are tried in the same fashion on the node that is second-best at minimizing expected execution time. This process is repeated until all tasks have been mapped. Our technique for generating solutions that meet the energy-robustness constraint (for MO-EC) is similar to that above, except by assigning tasks to cores and

P-states that minimize expected energy instead of expected execution time, and making sure assignments result in nonzero makespan-robustness. For the platform sizes and task execution time distributions we considered in this study, our technique resulted in feasible initial populations.

APPENDIX D

TABLE OF NOTATIONS FOR
ROBUST ENERGY-AWARE RESOURCE MANAGEMENT

TABLE D.1. Table of Notations

Notation	Description
c	Weighting constant for penalized objective function
C_i	Load capacitance of a core in node i
$core_{maxM}$	Core that has the greatest probability of finishing its assigned workload by the deadline
$core_{minM}$	Core that has the least probability of finishing its assigned workload by the deadline
δ	System deadline
Δ	Energy budget
d_ϕ	Distance from feasibility for MO-EC
d_Ψ	Distance from feasibility for EO-MC
η	Energy-robustness constraint for MO-EC
F_i	Maximum expected finishing time among cores in node i
F_{ij}	Expected finishing time of core j in node i
$f_{j\pi}$	Frequency of core j in P-state π
γ	Variance of energy required to complete the workload
Γ	Makespan-robustness constraint for EO-MC
EO-MC	The maximizing energy-robustness with a makespan-robustness constraint problem
$Mean_{ij\pi}^{dyn}$	Expected dynamic energy spent by core j of node i in P-state π
$Mean_{ij\pi}^{stat}$	Expected static energy spent by core j of node i in P-state π
MO-EC	The maximizing makespan-robustness with an energy-robustness constraint problem
$\mu(t_{ij}^x, \pi)$	Mean execution time of task t_{ij}^x in P-state π
N	Number of heterogeneous nodes
$\mathcal{N}(F_{ij}, \sigma_{ij}^2)$	Completion time distribution of core j in compute node i
n_i	Number of homogeneous cores in node i
O_i	Overhead power used by additional components in node i
ϕ	Energy-robustness of a resource allocation
$P_{ij\pi}^{dyn}$	Dynamic power of a core j on node i operating in P-state π
P_i^{stat}	Static power consumption of a core within node i
PS_i	Number of P-states available in node i
Ψ	Makespan-robustness of a resource allocation
ψ_ϕ	Penalized objective function for EO-MC
ψ_Ψ	Penalized objective function for MO-EC
$rank(t, i)$	Rank of task t on node i
σ_{ij}^2	Variance of completion time distribution of core j in node i
σ_i^2	Variance of core with maximum expected finishing time in node i
T	Number of tasks in the bag-of-tasks
T_{ij}	Tasks in T that have been assigned to core j in node i
t_{ij}^x	Task from set T_{ij} where $1 \leq x \leq T_{ij} $
$var(t_{ij}^x, \pi)$	Variance of execution time of task t_{ij}^x in P-state π
$Var_{ij\pi}^{dyn}$	Variance of dynamic energy spent by core j of node i in P-state π
$Var_{ij\pi}^{stat}$	Variance of static energy spent by core j of node i in P-state π
$Vdd_{j\pi}$	Supply voltage of core j in P-state π
ζ	Expected energy required to complete the workload

NLP FORMULATION FOR RATE-BASED RESOURCE MANAGEMENT

In this appendix that is supplementary to Chapter 3, we show the formulation for the non-linear programming technique [56]. The decision variables are: the desired fraction of time tasks of each type are executing on each core ($DF(i, k)$), the outlet temperature of each CRAC unit (TC_i^{out}), and the P-state of each core k when running a task type i ($PS(i, k)$). The following equation shows the assignment problem for our NLP that uses CRR as the objective function:

$$(68) \quad \text{maximize } CRR$$

subject to

$$(1) \quad \sum_{i=1}^T DF(i, k) \leq 1, k = 1, \dots, NC$$

$$(2) \quad \sum_{k=1}^{NC} CER(i, k) \leq \lambda_i, i = 1, \dots, T$$

$$(3) \quad \sum_{j=1}^{NN} PN(j) + \sum_{i=1}^{NCR} PC(i) \leq \phi$$

$$(4) \quad T_j^{in} \leq T^{redline}, j = 1, \dots, NN$$

The objective function is the total reward rate (actual reward after considering interference effects). The first constraint guarantees that the total fraction of time a core k spends executing tasks does not exceed 100%. Constraint 2 guarantees that the sum of execution rates of a task type over all cores does not exceed the arrival rate of that task type. Constraint

3 guarantees the power constraint. Finally, Constraint 4 ensures that the red-line temperatures of nodes are not violated.

This problem is a mixed-integer non-linear program (MINLP) for the following two reasons. First, the above problem contains integer decision variables, because the P-state decision variables are integers, and therefore the problem is mixed-integer. Second, the CoP function used to calculate CRAC power in Equation 5 is non-linear (see Sections 3.3.4 and 3.5.3). Because the power of a CRAC unit i is non-linear with respect to the decision variable \mathbf{TC}_i^{out} , Constraint 3 a non-linear constraint. We relax this problem to be solvable in a reasonable amount of time by following the steps outlined in Section 3.4.1.

APPENDIX F

SIMULATION PARAMETERS FOR RATE-BASED RESOURCE MANAGEMENT

In this appendix that is supplementary to Chapter 3, we summarize our simulation parameters and provide error estimates of the thermal and co-location models. Table F.1 gives the idle power consumption measurements of each of the compute node types. Tables F.2 - F.7 give the coefficients of the co-located execution time model (Equation 21). Table F.10 gives the power consumption of the cores in different node types when executing the different PARSEC benchmarks in the highest power P-state, and Table F.11 gives the power consumption of the cores in different node types when executing the different PARSEC benchmarks in the lowest power P-state. Table F.8 gives the measured execution times of the PARSEC benchmarks on the three node types in the highest power P-state, and Table F.9 gives the measured execution times of the PARSEC benchmarks on the three node types in the highest power P-state. The memory intensity thresholds (number of last-level cache misses to number of instructions executed for that application) to classify task types into Class I was between 1 and 0.01, Class II was between 0.01 and 0.001, Class III was between 0.001 and 0.00001, and Class IV was less than 0.00001.

Training and testing data for the co-location interference model were collected across the PARSEC benchmarks on the servers we considered in this work [52]. The mean percent error in the testing set when using the linear co-location interference model with the features we listed was found to be approximately 6%. That is, the mean percent error between the actual execution times of the benchmarks when under co-location interference effects, and the execution times predicted by the model was 6%.

TABLE F.1. Idle power consumption (watts) of each node type

	CPU Type	idle power (W)
Node Type 1	Xeon E3-1225v3	30.0
Node Type 2	Xeon E5649	142.4
Node Type 3	Xeon E5-2697v2	156.0

TABLE F.2. $u(m, k)$ Co-efficients

	Class I	Class II	Class III	Class IV
Node Type 1	23.4	17.7	56.7	26.3
Node Type 2	21.0	7.7	8.1	8.1
Node Type 3	5.3	6.2	2.6	2.0

TABLE F.3. $v(m, k)$ Co-efficients

	Class I	Class II	Class III	Class IV
Node Type 1	1.2	1.0	0.9	0.9
Node Type 2	1.2	1.4	1.0	0.2
Node Type 3	0.9	0.7	1.1	1.3

TABLE F.4. $w(m, k)$ Co-efficients

	Class I	Class II	Class III	Class IV
Node Type 1	21.0	-1.6	-76.3	-66.3
Node Type 2	43.9	-9.3	27.4	-241.2
Node Type 3	0.9	0.7	1.1	1.3

TABLE F.5. $x(m, k)$ Co-efficients

	Class I	Class II	Class III	Class IV
Node Type 1	725	1,191	-625	1,702
Node Type 2	1,083	875	1,795	1,917
Node Type 3	300	1043	273	350

TABLE F.6. $y(m, k)$ Co-efficients

	Class I	Class II	Class III	Class IV
Node Type 1	39	-3,054,443	149,591	-38,176,035
Node Type 2	51	-21,936,224	5,230	-182,980,864
Node Type 3	13,733	-102,387	-39,953	1

TABLE F.7. $z(m, k)$ Co-efficients

	Class I	Class II	Class III	Class IV
Node Type 1	-121	25,871	257	490
Node Type 2	-184	120	-75	1,331
Node Type 3	-192	447	-76	206

Likewise, the error of the thermal model we used was calculated [75]. The average absolute error between their model and CFD simulations was found to lie within the 0.1-0.4°C range, while the maximum absolute error was found to lie in the 0.4-2.8°C range.

TABLE F.8. Execution times (seconds) of PARSEC benchmarks on our lab servers at **highest** frequency P-state

	CPU Type	<i>canneal</i>	<i>cg</i>	<i>ua</i>	<i>sp</i>	<i>lu</i>	<i>fluidanimate</i>	<i>blackscholes</i>	<i>bodytrack</i>	<i>ep</i>	<i>swaptions</i>
Node Type 1	Xeon E3-1225v3	148	110	211	214	276	359	236	178	273	297
Node Type 2	Xeon E5649	195	132	316	343	368	559	315	297	397	485
Node Type 3	Xeon E5-2697v2	178	146	171	328	229	365	222	184	328	277

TABLE F.9. Execution times (seconds) of PARSEC benchmarks on our lab servers at **lowest** frequency P-state

	CPU Type	<i>canneal</i>	<i>cg</i>	<i>ua</i>	<i>sp</i>	<i>lu</i>	<i>fluidanimate</i>	<i>blackscholes</i>	<i>bodytrack</i>	<i>ep</i>	<i>swaptions</i>
Node Type 1	Xeon E3-1225v3	345	333	687	752	942	1,403	717	657	1,117	1,234
Node Type 2	Xeon E5649	250	195	474	561	624	977	492	452	715	873
Node Type 3	Xeon E5-2697v2	385	280	457	742	636	1,005	525	507	742	797

TABLE F.10. Power consumption (watts) of a core when executing PARSEC benchmarks on our lab servers at **highest** frequency P-state

	CPU Type	<i>canneal</i>	<i>cg</i>	<i>ua</i>	<i>sp</i>	<i>lu</i>	<i>fluidanimate</i>	<i>blackscholes</i>	<i>bodytrack</i>	<i>ep</i>	<i>swaptions</i>
Node Type 1	Xeon E3-1225v3	5.6	7.1	7.1	6.5	5.1	8.6	7.1	3.9	7.9	8.5
Node Type 2	Xeon E5649	6.5	7.9	10.1	9.3	11.6	9.9	7.7	10.6	9.0	9.6
Node Type 3	Xeon E5-2697v2	4.3	6.7	3.3	6.7	9.5	6.0	8.8	3.3	5.3	3.3

TABLE F.11. Power consumption (watts) of a core when executing PARSEC benchmarks on our lab servers at **lowest** frequency P-state

	CPU Type	<i>canneal</i>	<i>cg</i>	<i>ua</i>	<i>sp</i>	<i>lu</i>	<i>fluidanimate</i>	<i>blackscholes</i>	<i>bodytrack</i>	<i>ep</i>	<i>swaptions</i>
Node Type 1	Xeon E3-1225v3	1.2	1.7	1.9	2.0	2.1	1.2	0.9	1.4	0.9	0.8
Node Type 2	Xeon E5649	4.0	5.5	5.0	6.4	6.5	4.9	4.0	5.5	4.1	4.5
Node Type 3	Xeon E5-2697v2	2.3	3.4	1.0	4.4	5.5	2.4	4.8	1.0	1.4	1.1