

3-22-2018


Energy Demand Response for High-Performance Computing Systems

Kishwar Ahmed

Florida International University, kahme006@fiu.edu

DOI: 10.25148/etd.FIDC006527

Follow this and additional works at: <https://digitalcommons.fiu.edu/etd>

 Part of the [Contracts Commons](#), [Digital Communications and Networking Commons](#), and the [Power and Energy Commons](#)

Recommended Citation

Ahmed, Kishwar, "Energy Demand Response for High-Performance Computing Systems" (2018). *FIU Electronic Theses and Dissertations*. 3569.

<https://digitalcommons.fiu.edu/etd/3569>

This work is brought to you for free and open access by the University Graduate School at FIU Digital Commons. It has been accepted for inclusion in FIU Electronic Theses and Dissertations by an authorized administrator of FIU Digital Commons. For more information, please contact dcc@fiu.edu.

FLORIDA INTERNATIONAL UNIVERSITY
Miami, Florida

ENERGY DEMAND RESPONSE FOR HIGH-PERFORMANCE COMPUTING
SYSTEMS

A dissertation submitted in partial fulfillment of the
requirements for the degree of
DOCTOR OF PHILOSOPHY
in
COMPUTER SCIENCE
by
Kishwar Ahmed

2018

To: Dean John L. Volakis
College of Engineering and Computing

This dissertation, written by Kishwar Ahmed, and entitled Energy Demand Response for High-Performance Computing Systems, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

S. S. Iyengar

Deng Pan

Leonardo Bobadilla

Wujie Wen

Jason Liu, Major Professor

Date of Defense: March 22, 2018

The dissertation of Kishwar Ahmed is approved.

Dean John L. Volakis
College of Engineering and Computing

Andres G. Gil
Vice President for Research and Economic Development and
Dean of the University Graduate School

Florida International University, 2018

© Copyright 2018 by Kishwar Ahmed

All rights reserved.

DEDICATION

To my parents and my wife.

ACKNOWLEDGMENTS

This dissertation would not have been possible without the guidance and help of my advisor, Dr. Jason Liu. In spite of his very busy schedule, he made every effort to guide me throughout the entire process. I am really grateful to him for his tremendous support, and no word of gratitude is sufficient to convey it.

I am also thankful to my committee members, Dr. S. S. Iyengar, Dr. Deng Pan, Dr. Leonardo Bobadilla, and Dr. Wujie Wen for their valuable suggestions regarding my dissertation. I am grateful to Dr. Stephan Eidenbenz and Dr. Nandakishore Santhi from Los Alamos National Laboratory, Dr. Xingfu Wu from Argonne National Laboratory, and Dr. Jesse Bull from Florida International University for their constructive suggestions and contributions on my dissertation. I also want to thank Kazutomo Yoshii, Dr. Rob Ross, and Dr. Misbah Mubarak for the opportunity to work with them at Argonne National Laboratory as an intern. Their valuable guidance and help have benefited me a lot in my doctoral study. I would like to thank Mohammad Atiqul Islam, with whom I collaborated during first few years of my doctoral study.

My utmost gratitude goes to my parents Erfanuddin Ahmed and Shaheda Begum. With their encouragement and valuable suggestions, I have been able to reach this far. My wife, Samia Tasnim, has accompanied throughout my doctoral journey. Her love, care and support have propelled me through the difficult times of doctoral study. I owe her everything. I am grateful to my brother, Shaer Ahmed, who gave inspiration at various times. I would also like to thank my friends Mohammad Chowdhury, Mohammad Obaida, Naeemul Hassan, Tanay Kumar Saha, Jesun Feroz for giving me much-needed breaks during exhausting research times.

ABSTRACT OF THE DISSERTATION
ENERGY DEMAND RESPONSE FOR HIGH-PERFORMANCE COMPUTING
SYSTEMS

by

Kishwar Ahmed

Florida International University, 2018

Miami, Florida

Professor Jason Liu, Major Professor

The growing computational demand of scientific applications has greatly motivated the development of large-scale high-performance computing (HPC) systems in the past decade. To accommodate the increasing demand of applications, HPC systems have been going through dramatic architectural changes (e.g., introduction of many-core and multi-core systems, rapid growth of complex interconnection network for efficient communication between thousands of nodes), as well as significant increase in size (e.g., modern supercomputers consist of hundreds of thousands of nodes). With such changes in architecture and size, the energy consumption by these systems has increased significantly. With the advent of exascale supercomputers in the next few years, power consumption of the HPC systems will surely increase; some systems may even consume hundreds of megawatts of electricity. Demand response programs are designed to help the energy service providers to stabilize the power system by reducing the energy consumption of participating systems during the time periods of high demand power usage or temporary shortage in power supply.

This dissertation focuses on developing energy-efficient demand-response models and algorithms to enable HPC system's demand response participation. In the first part, we present interconnection network models for performance prediction of

large-scale HPC applications. They are based on interconnected topologies widely used in HPC systems: dragonfly, torus, and fat-tree. Our interconnect models are fully integrated with an implementation of message-passing interface (MPI) that can mimic most of its functions with packet-level accuracy. Extensive experiments show that our integrated models provide good accuracy for predicting the network behavior, while at the same time allowing for good parallel scaling performance. In the second part, we present an energy-efficient demand-response model to reduce HPC systems' energy consumption during demand response periods. We propose HPC job scheduling and resource provisioning schemes to enable HPC system's emergency demand response participation. In the final part, we propose an economic demand-response model to allow both HPC operator and HPC users to jointly reduce HPC system's energy cost. Our proposed model allows the participation of HPC systems in economic demand-response programs through a contract-based rewarding scheme that can incentivize HPC users to participate in demand response.

TABLE OF CONTENTS

CHAPTER	PAGE
1. INTRODUCTION	1
1.1 Motivation	1
1.2 Problem Definition and Contributions	3
1.2.1 Rapid Performance Modeling for HPC Systems	3
1.2.2 Emergency Demand Response for HPC Systems	7
1.2.3 Economic Demand Response for HPC Systems	9
1.3 Related Publications	10
1.4 Outline of the Dissertation	11
2. BACKGROUND	13
2.1 Performance Prediction Models	13
2.1.1 Interconnection Network	14
2.1.2 Memory System	18
2.1.3 Processor System	19
2.1.4 HPC Applications	23
2.2 HPC Power Models	25
3. RAPID PERFORMANCE MODELING FOR HPC SYSTEMS	28
3.1 Background	28
3.2 Related Work	31
3.3 Model	34
3.3.1 MPI Model	36
3.3.2 Interconnection Network Models	41
3.3.3 Interconnect Model Validations	49
3.4 Experiments	59
3.4.1 Trace-Driven MPI Simulation	59
3.4.2 SNAP Performance Study	65
3.4.3 Parallel Performance	67
3.5 Summary	69
4. EMERGENCY DEMAND RESPONSE FOR HPC SYSTEMS	70
4.1 Background	70
4.2 Related Work	72
4.3 Demand-Response Model Based on Frequency Scaling	76
4.3.1 Power and Performance Prediction Models	76
4.3.2 Job Scheduling and Resource Provisioning	79
4.3.3 Determining Optimal Frequency	86
4.3.4 Job Eviction	87
4.3.5 Performance Evaluation	88
4.4 Demand-Response Model Based on Power Capping and Node Scaling	95

4.4.1	Exploiting Power-Capping Property	96
4.4.2	Exploiting Power-Capping and Node-Scaling Properties	102
4.4.3	Performance Evaluation	105
4.5	Summary	108
5.	ECONOMIC DEMAND RESPONSE FOR HPC SYSTEMS	110
5.1	Background	110
5.2	Related Work	112
5.2.1	Reducing Energy Cost	112
5.2.2	Demand Response	114
5.2.3	Contract Theory and Applications	115
5.3	Model	116
5.3.1	An Example	117
5.3.2	HPC System Model	119
5.4	Formulation and Algorithm	124
5.4.1	Feasibility and Optimality of Solutions	125
5.4.2	Contract Design with Continuum Type	130
5.5	Experiments	131
5.5.1	Data Set	131
5.5.2	Energy Reduction and Utility	133
5.5.3	Benchmark Comparison	135
5.6	Summary	137
6.	CONCLUSIONS	138
6.1	Summary	138
6.2	Future Directions	140
	BIBLIOGRAPHY	143
	VITA	168

LIST OF FIGURES

FIGURE	PAGE
3.1 An architectural design of PPT.	35
3.2 An example showing running 16 MPI processes on Hopper.	39
3.3 Simulating Cannon’s matrix multiplication.	40
3.4 Cray Gemini ASIC block diagram.	42
3.5 Interconnect model using Simian entities, processes, and services.	43
3.6 Cray Aries block diagram.	47
3.7 A histogram of end-to-end delay between compute nodes of the simulated HPC cluster.	52
3.8 Duration of the <code>MPI_Allreduce</code> call for different number of ranks and data size on the simulated HPC cluster.	53
3.9 MPI throughput from simulation as a function of message size for 1, 2 and 4 MPI processes per node.	54
3.10 Gemini FMA put throughput (as reported in [ARK10]) versus simulated throughput as a function of transfer size for 1, 2, and 4 processes per node.	55
3.11 Aries validation.	56
3.12 Comparison with FatTreeSim and Emulab.	58
3.13 Format of MPI calls in the processed trace file (there is one trace file for each MPI rank).	60
3.14 Comparing the duration of MPI calls between trace and simulation with and without time shift.	62
3.15 Comparison of different architectures for trace-based run.	64
3.16 SNAPSim vs. SNAP Edison strong scaling.	66
3.17 Observed run time and event rate for running Simian with an 156K-rank MPI model on a parallel compute cluster.	68
4.1 Result of the power and performance prediction models for six HPC applications.	78
4.2 The overall design of our job scheduler simulator.	84
4.3 Comparing results from PYSS and our simulator.	86

4.4	Power usage over time with and without power capping.	90
4.5	Comparing performance and energy for different scheduling policies and with different system size.	92
4.6	Impact on the demand response event ratio.	93
4.7	Power stability during the demand response periods.	94
4.8	Impact of power capping on application characteristics.	96
4.9	Power regression model for different applications.	99
4.10	Runtime regression model for different applications.	100
4.11	Node scaling model for different applications.	103
4.12	Benchmark comparison with power capping.	106
4.13	Benchmark comparison with power capping and node scaling.	107
5.1	Energy reduction and electricity pricing from PJM economic demand response on July 18, 2013 [PJM17].	111
5.2	Job arrivals and electricity price data.	132
5.3	Energy reduction and utility.	134
5.4	Incentive compatibility constraint.	135
5.5	Benchmark comparison.	135
5.6	Change in electricity price.	136

CHAPTER 1

INTRODUCTION

1.1 Motivation

High-performance computing (HPC) systems, such as petaflops supercomputers, can consume a tremendous amount of power. For example, as of November 2016, China's 34-petaflops Tianhe-2 supercomputer, which currently consumes the most power in the list of top 500 supercomputers [TOP16], has been reported to consume almost 18 MWs of power, sufficient to power a small town of 20,000 residences. With the advent of exascale supercomputers in the next few years, power consumption of the HPC systems will surely increase: a recent projection indicates that an exascale system would reach 60-130 MWs of power [YZW⁺13]. The massive power consumption of these HPC systems can expound significant stress for the power grid. HPC has also shown significant fluctuations in the power consumption due to the varying job execution profiles and also sporadic maintenance schedules. Effective power saving and power stabilizing methods must be seriously considered when building future HPC systems.

Demand response programs are designed to help the energy service providers to stabilize the power system by reducing the energy consumption of participating systems when the power grid becomes unstable due to a sudden rise in power demand or other emergency incidents. Demand response can be broadly categorized into two types: economic demand response and emergency demand response. In economic demand response, participants voluntarily enroll in the programs (without the need of prior commitment) and willingly reduce the load based on economic incentives offered by the supplier. Emergency demand response requires prior commitment from the participants; once enrolled, it is mandatory for the participants to reduce the

energy consumption to requested levels when supply shortage situations or emergency conditions occur. The U.S. Department of Energy (DoE) and the National Institute of Standards and Technology (NIST) have identified demand response as one of the important policy goals to achieve power grid efficiency [HBH14, Fed16]. In addition to monetary benefits, demand response can also provide the associated environmental benefits, such as reducing carbon emission [PJM14]. We have observed a recent increase in the participation of the demand response programs in various sectors [The13, McA17]. Recent projection also shows that there will be substantial growth in the coming years—an anticipated doubling of the overall participation in the demand response programs in 2020 has been projected [ME13]. Motivated by an increase in popularity of demand response program participation and massive energy consumption of HPC systems, this dissertation aims to explore the opportunity of HPC systems’ energy consumption reduction through emergency and economic demand response participation.

To enable HPC systems’ demand response participation, we need to analyze the power-performance tradeoff of HPC applications, and develop detailed performance prediction models for HPC systems containing thousands of nodes. The rapid advancement towards exascale computing has led to the emergence of novel hardware architecture designs in HPC systems that include accelerator technologies (such as GPUs), high core-count compute nodes with shared memory, deep instruction pipelines, deep memory hierarchies with aggressive memory prefetching strategies, and sophisticated branch prediction for speculative execution. These new architectural features enable massive *parallelism* and *latency hiding* that in principle allow software and codes to scale to next-generation HPC systems. For example, Intel’s Knight’s Corner node features 61 cores with shared main memory (albeit at a non-uniform access speed) that enables thread-level parallelism.

In contrast, NVIDIA’s Tesla GPU accelerators have up to 3,000 CUDA Cores per CPU enabling vector parallelism. Different parallelization strategies were adopted in these cases. CPU-based nodes use a significant fraction of their chip real estate to implement pipelining logic (to enable instruction-level parallelism) and memory prefetching logic at different cache levels (to enable latency hiding), whereas GPU designs tend to maximize core counts with arithmetic logic units (ALUs) for enabling vector parallelism. These novel hardware technologies have turned out to be disruptive to existing software portfolios in many industries and government branches because simple re-compilation does not exploit these features very well. This in turn has led to massive code re-factoring in many sectors, including—and perhaps most pronounced—among users of high-performance computational physics code. Performance prediction on how fast and how energy-efficient a code will run on a platform is at the heart of computational co-design.

1.2 Problem Definition and Contributions

The primary goal of this dissertation is to identify the key challenges and explore the power and performance modeling to enable HPC system’s demand response participation.

1.2.1 Rapid Performance Modeling for HPC Systems

With frequent changes in HPC systems, it is imperative that performance prediction of future HPC systems is properly realized. Of particular importance is the model for the interconnection networks as it is critical to the understanding of the communication cost and thus the performance limitations of large-scale applications on high-performance computing infrastructures. Such large-scale interconnection

network models allow performance prediction of HPC applications on many nodes, and therefore enable analysis of power-performance tradeoff of HPC applications. There has been significant research effort on performance prediction and modeling of extreme-scale interconnection networks (e.g., [LHSJ15, LC11, Per10, PP14]). However, few of these research efforts consider the effect of complex, dynamic application behaviors, such as computational physics code, on the underlying large-scale interconnection network.

The Performance Prediction Toolkit (PPT) is a DOE co-design project that aims at developing a comprehensive prediction capability for computational physics code, algorithms and methods that perform on novel hardware architectures, thus enabling fast adoption of new code by quickly identifying and ruling out unsuccessful refactoring schemes. PPT models both hardware and software at levels of abstraction that are appropriate to the concrete question at hand, by applying a mix of discrete-event simulation, stochastic and analytical models at various layers on the software and hardware stack. PPT relies on Simian [SEL15], a parallel discrete-event simulation engine, and essentially consists of libraries of hardware models, application models, and middleware models. PPT, along with Simian, is designed to be lean, written in Python (or alternatively Lua) with minimal reliance on third-party libraries in an effort to keep the code simple, understandable, and yet offer high performance.

Contributions. Our contributions to performance modeling of large-scale HPC system are summarized below:

1. We present PPT’s interconnection network models to model communication among many nodes in HPC systems and predict performance of HPC applications. Our interconnection network models include widely-used interconnect topologies with emphasis on production networks (both existing and planned

interconnection networks). In today’s top-ranked HPC systems, we see three common network types: torus (e.g., Cray’s Gemini, IBM’s Blue Gene/Q), dragonfly (e.g., Cray’s Aries), and fat-tree (e.g., Infiniband). They constitute a majority of the production network topologies. Our survey on the latest supercomputers (<http://www.top500.org>, June 2016) shows that the three topologies account for 54% of the 500 fastest supercomputers in the world (44% for fast ethernet and 2% proprietary). Among the top 100 supercomputers, the three topologies grow up to 82%. 14 of the top 15 ranked supercomputers are interconnected by the three types. In PPT, separate interconnection network models have been developed and carefully parameterized in PPT to capture various production interconnection networks. We present sufficiently detailed interconnection models for Cray’s Gemini 3-D torus, IBM’s Blue Gene/Q 5-D torus, Cray’s Aries dragonfly, and Infiniband’s fat-tree network.

2. PPT’s interconnection network models are packet-level models, where network transactions (e.g., for MPI send/receive and for collective operations) are modeled as discrete events representing individual packets (typically, around 64 bytes in size) being transferred by the network switches and compute nodes. This is a conscious design decision. Our hypothesis is that in most scenarios, packet-level simulation should be sufficient to capture major network behaviors (throughput, delay, loss, and network congestion) with sufficient accuracy, and as such, should be able to identify potential performance bottlenecks at the interconnection networks while running large-scale scientific applications. Compared to more detailed models, such as those implemented at the phit level (virtual channels), packet-level simulation can easily outperform detailed models by several orders of magnitude. We present extensive validation studies of our MPI and interconnect models, including a trace-based study using

data obtained from executing real-life computational physics code on an existing high-performance computing platform. Our experiments suggest that our packet-level models can provide sufficient accuracy.

3. PPT's interconnection network models can be easily incorporated with the application models. Our interconnection network models interface with the message-passing interface (MPI) model. MPI is the most commonly used parallel programming tools for scientific applications on modern HPC platforms. Our MPI model provides convenient methods for deploying the parallel applications and performing communications on the target parallel platform. We have implemented all common MPI functions, including point-to-point communications (both blocking and asynchronous methods) and collective operations (such as gather/scatter, barriers, broadcast, reduce, and all-to-all). In addition, we implemented MPI groups and communicators so that collective communications can take place among an arbitrary subset of processes. As a result, most scientific applications can be simulated directly using the communication functions provided by the MPI model.
4. We conduct extensive validation study of our interconnect models, including a trace-driven simulation of real-life scientific application communication patterns. We also perform performance study of a computational physics-based parallel application using our interconnect model. All the results show that our interconnect models provide reasonably good accuracy. Moreover, we study the parallel performance of our integrated models on large-scale HPC platforms and show good parallel scaling performance.

1.2.2 Emergency Demand Response for HPC Systems

Being a massive energy consumer of the power grid, the HPC sector can contribute toward ensuring grid stability and energy reduction through its participation in the demand response programs. Recent research has studied the feasibility and identified the associated challenges in the HPC demand response [BGA⁺15, PBG⁺16]. Patki et al. [PBG⁺16] suggested that supercomputing systems in the U.S. may be willing to participate in the demand response programs if tighter and more frequent communications can be established between the supercomputing centers and their energy service providers. Patki’s study is based on a qualitative analysis of cooperative demand-management strategies. We note, however, that there is no related work on the job scheduling and resource provisioning strategies at HPC centers that can operate with demand response. Various energy-efficient HPC job scheduling algorithms (e.g., [SLGK14, PLS⁺15, ECLV12]) and resource provisioning methods (e.g., [GFFC07, LM06, BHC⁺16]) have been proposed in the literature. These studies aim at reducing the overall energy consumption of the HPC systems, but do not consider demand response. In this dissertation, we explore the opportunities of the HPC centers participating in the demand response programs through a study of detailed job scheduling and resource provisioning strategies.

Contributions. Our contributions to power modeling in terms of emergency demand response participation from HPC systems are summarized below:

1. We propose an HPC job scheduling and resource provisioning algorithm for demand response. For job scheduling, we assume first-come-first-serve (FCFS) with possible job eviction and restart in response to the reduced power level during the demand response periods. For resource provisioning, we dynamically scale the frequency of the processors in order to achieve optimal en-

energy conservation and power stability during the demand response periods. During normal periods, the processors in HPC systems operate at maximum frequency for best performance. Also, we develop a simulator for job scheduling and resource provisioning to study the effect of demand response. The simulator is built upon a parallel discrete-event simulation engine capable for handling large-scale models. The simulator has been validated using real-life HPC workload traces.

2. We exploit the power-capping capability in the modern processors to enable HPC system emergency demand response participation. We present prediction models for power and performance prediction with respect to different power-capping values. We propose a demand response participation model for HPC systems based on the prediction models with power capping. We extend the HPC system demand response model by exploiting job malleability. We incorporate an energy-to-solution prediction model to the demand response model in order to determine the optimal job size and power-capping values.
3. We conduct extensive trace-based simulation studies to show the effectiveness of the proposed job scheduling and resource provisioning algorithm for demand response. The results demonstrate that our proposed approach is a viable solution for attracting supercomputing centers to participate in the demand response programs as it can improve power stability and energy reduction with only moderate increase in execution time for the jobs. Moreover, we perform experiments using real-life scientific applications on an existing cluster to measure application performance and power usage under different power-capping values. Using these measurements, we use trace-based simulation to show the effectiveness of our proposed demand-response model and compare it with power-capping policies implemented in processors.

1.2.3 Economic Demand Response for HPC Systems

Demand response program anticipates customers to reduce energy consumption upon requests from the power utility companies during the time periods of high demand power usage or temporary shortage in power supply. Customers are willing to participate in demand response programs in expectation to receive financial or operational benefits from the utility companies. Overall, demand response has become increasingly popular among power utility companies. Revenue earnings from demand response has increased significantly in recent years. For example, a report from PJM Interconnection (a large utility company servicing many states in the U.S.) shows that it has achieved an earning of \$650 million from various demand response participation in 2016, a significant increase from only about \$50 million in 2006 [PJM17]. Much of this growth can be attributed to the economic demand response programs using incentives provided via fluctuations in electricity pricing, through which the power utility companies can signal the consumers to adapt their behaviors at various time granularities (such as hourly). In the final part of this dissertation, we address how to reduce energy cost in HPC systems through a contract-based economic demand response participation.

Contributions. Our contributions are summarized below:

1. We propose an economic demand response participation model for HPC systems for energy reduction. We propose a rewarding scheme to be offered by the HPC operator to the HPC users based on contract design to encourage the willing participation of the users in demand response.
2. Our analyses demonstrate that the proposed contract-based demand response mechanism preserves important properties of the contract theory, including *individual rationality (IR)*, *incentive compatibility (IC)*, and *monotonicity*.

3. Through trace-based simulation, we provide the empirical evidence of our proposed approach and further demonstrate the effectiveness of our proposed mechanism compared to other existing approaches. The simulation experiments also support our analytical claims in practice.

1.3 Related Publications

This dissertation is drawn from the following publications:

- Kishwar Ahmed, Jason Liu, and Kazutomo Yoshii. Enabling Demand Response for HPC Systems Through Power Capping and Node Scaling. Submitted to IEEE International Conference on High Performance Computing and Communications (HPCC), 2018.
- Kishwar Ahmed, Jesse Bull, and Jason Liu. Contract-Based Demand Response Model for HPC Systems. Submitted to International Conference on Parallel Processing (ICPP), 2018.
- Kishwar Ahmed, Jason Liu, Abdel-Hameed Badawy, and Stephan Eidenbenz. A Brief History of HPC Simulation and Future Challenges. In 2017 Winter Simulation Conference (WSC), pages 419-430. IEEE, 2017.
- Kishwar Ahmed, Jason Liu, and Xingfu Wu. An Energy Efficient Demand-Response Model for High Performance Computing Systems. In 2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pages 175-186. IEEE, 2017.
- Kishwar Ahmed, Jason Liu, Stephan Eidenbenz, and Joe Zerr. Scalable interconnection network models for rapid performance prediction of HPC applica-

tions. In High Performance Computing and Communications (HPCC), 2016 IEEE 18th International Conference on, pages 1069-1078. IEEE, 2016.

- Kishwar Ahmed, Mohammad Obaida, Jason Liu, Stephan Eidenbenz, Nandakishore Santhi, and Guillaume Chapuis. An integrated interconnection network model for large-scale performance prediction. In Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS), pages 177-187. ACM, 2016.

1.4 Outline of the Dissertation

We discuss the background and related work in Chapter 2. We first provide a snapshot of the existing performance prediction models and simulators in Section 2.1. We consider models for different sub-systems, including processors, memory, and interconnection networks. We also discuss application models that can capture the runtime behavior of the large-scale scientific applications. We discuss analytical models and tools for energy and power prediction of the HPC systems in Section 2.2.

We present how to model HPC applications' performance on underlying architecture, and present our *rapid performance prediction models* for large-scale interconnection network in Chapter 3. Section 3.1 discusses background information and challenges. Section 3.2 describes related works and compares our approach with the existing methods. Section 3.3 provides an overview of our design. In the same section, we also provide the MPI model and the details of our torus, dragonfly, and fat-tree interconnection network models (along with validations). In Section 3.4, we present trace-driven simulation of real-life scientific application communication patterns, as well as performance study of a computational physics-based parallel application using one of our interconnect models.

Chapter 4 describes our approach to address HPC system’s massive power consumption and presents our *energy-efficient emergency demand-response models* for HPC systems. We present background information and contributions in Section 4.1. In Section 4.2, we describe related work and compare the existing approaches with our proposed method. In Section 4.3, we explore the opportunities of the HPC systems participating in the emergency demand programs through a study of detailed job scheduling and resource provisioning strategies. In Section 4.4, we exploit the power-capping property in the modern processors and node scaling of HPC applications to enable HPC system emergency demand response participation.

Chapter 5 presents our *economic demand-response model* for HPC systems. Section 5.1 discusses background information and challenges. Section 5.2 describes related works and compares our approach with the existing methods. Section 5.3 describes the contract-based HPC economic demand response participation model. In Section 5.4, we present the problem formulation and corresponding algorithm. We also describe the necessary conditions for contract design mechanisms in the same section. In Section 5.5, we use real-life data to show effectiveness of our proposed solution. Finally, we present our concluding remarks and provide direction for future work in Chapter 6.

CHAPTER 2

BACKGROUND

2.1 Performance Prediction Models

HPC systems today consist of hundreds of thousands of compute nodes, and can perform tens or hundreds of quadrillion floating point operations per second [Tsa13, Cou16]). HPC architectures have gone through rapid changes to facilitate the increasing computational demand of scientific applications in many areas, such as astrophysics, particle physics, earth and climate science, computational chemistry, computational biology, and so on. Novel technologies, for example, many-core processors, GPUs, persistent memory, and complex interconnection networks, have been introduced constantly to fulfill the increasing scale and performance of such systems. With the changing hardware architectures also comes the changing software design and implementation of scientific applications in order to take best advantage of the new computing resources.

Modeling and simulation plays an important role for performance prediction and analysis of current and future HPC systems. It can be particularly useful for evaluating the whole-system impact when new components are introduced, for comparing the performance of different system design alternatives, and for locating performance issues of computational code on novel HPC platforms even before their realization. It is thus not surprising to see many HPC models and simulation tools created in the past for exactly the same purposes. They model the HPC systems at different granularity: some are created to study specific components of the HPC systems (processors, memory, interconnect, storage, and so on), and others are meant for studying the overall system performance in aggregate. The important difference lies

in the accuracy-performance trade-off that has been applied to effectively capture the salient features of the target system.

2.1.1 Interconnection Network

HPC interconnection network offers a systematic way to connect compute nodes, processors, memory, and storage units. Important aspects of an interconnection network model include accurate representation of the network topology, routing, resource scheduling (such as flow control), and network queuing. Different interconnection network topologies exist in current HPC systems, including fat-tree (e.g., IBM’s Infiniband), torus (e.g., IBM’s Blue Gene/Q, Cray’s Gemini), and dragonfly (e.g., Cray’s Aries). An accurate model of these interconnection networks is important for us to understand the communication cost as one of the most important constraints on the performance of scientific applications running on HPC systems.

BigSim [ZKK04] is an early effort for performance prediction of large-scale parallel machines (e.g., Blue Gene/L machines), based on the model of parallel applications on the target architecture (such as using MPI). The interconnection network models developed in BigSim are relatively simple as they do not consider network congestion [CGL⁺14]. BigSim is implemented using Charm++, an object-based and message-driven parallel programming system [KK93]. The simulator adopts optimistic parallel simulation for scalability, using inherent determinacy of the target parallel applications to reduce the synchronization overhead. Experiments show that BigSim is capable of scaling up to 64K simulated processors.

Structural Simulation Toolkit (SST) is an all-inclusive simulation framework for modeling large-scale HPC systems, including processors, memory, interconnection networks, and I/O systems [RHB⁺11]. SST consists of models of various hardware components with different levels of accuracy and granularity, and attempts to achieve

scalability by using conservative parallel simulation based on the “distance” between the system components. SST supports generic router models which can be used to build different network topologies, such as binary tree, fat-tree, hypercube, flattened 2-D butterfly, 2-D and 3-D mesh, and fully-connected graph. The interconnection network models in SST, however, do not support flow control between routers and the links between routers are assumed to have infinite capacity. SST is an ongoing project and is able to include active contributions of many advanced component models as part of a scalable and open-source simulation framework.

Extreme-scale Simulator (xSim) is a performance-prediction toolkit for future HPC architectures [EL10, BE11]. xSim applies parallel discrete-event simulation using lightweight threads and has achieved good scalability with millions of MPI ranks running a simple MPI program. [JE11] extended xSim to incorporate a network model with different topologies (star, ring, mesh, torus, and tree), and different hierarchical network combinations (such as network-on-chip and network-on-node). However, the xSim models do not consider traffic congestion or any detailed blocking behavior which can be important to applications on real interconnection networks.

$\mu\pi$ [Per10] is an MPI simulator built upon an efficient conservatively-synchronized parallel simulator that presents a feature-oriented world-view. $\mu\pi$ supports simulation of large-scale MPI applications. Experiments show that it can run up to 27 million virtual MPI ranks on as many as 216,000 cores of a Cray XT5. More recently, Perumalla et. al. [PP14] proposed to extend $\mu\pi$ and include direct execution to run even larger number of tasks. The simulator focus on MPI communication of applications, but does not contain any detailed interconnection network models.

Co-Design of Exascale Storage System (CODES) simulator is another comprehensive simulation platform to model various large-scale systems, including storage systems, interconnection networks, HPC and data center applications [CLL⁺11].

CODES provides detailed interconnect models for various interconnect topologies, including torus [LC11], dragonfly [MCRC14], and fat-tree [LHSJ15]. The simulator is built on Rensselaer Optimistic Simulation System (ROSS), a parallel discrete-event simulation engine using reverse computation [CBP02], and is capable of simulating very large interconnect configurations (with millions of nodes). Trace-driven capabilities have also been added to CODES to replay large execution traces for studying network performance [AJB⁺15].

Garnet [AKPJ09] is an on-chip interconnection network simulator, which builds upon the lacking of GEMS [MSB⁺05] and performs “detailed” interconnect communication between on-chip routers. Through modeling detailed router microarchitecture, Garnet is able to capture various details such as virtual channel arbitration and realistic link contention. Garnet has provision for easy-configuration of various parameters (e.g., different network topologies, interconnect bandwidth configuration through flit size, router parameters such as arbitrary number of input and output ports, various routing algorithms). TOPAZ [APM⁺12] is yet another open-source NoC simulator with broader analysis spectrum (e.g., tradeoffs between accuracy, simulation speed for various interconnection network). It has an easy incorporation capability with other simulation tools (e.g., GEMS and gem5) in runtime. TOPAZ is multithreaded, therefore providing more accurate prediction capability without compromising simulation execution time. In addition to being capable of simulating NoCs in multicore processors, TOPAZ also supports simulation of large-scale interconnection networks and is able to simulate networks consisting of millions of routers.

2.1.1.1 A Parallel Discrete-Event Simulation Engine

All our interconnection network models are developed based on Simian, which is an open-source, process-oriented parallel discrete-event simulation (PDES) engine [SEL15]. Simian has two independent implementations written in two interpreted languages, Python and Lua, respectively. Simian uses a conservative barrier-based synchronization algorithm [Nic93] for parallel execution.

Simian has several distinct features. First, Simian adopts a minimalistic design. For example, the Python implementation of Simian consists of only around 500 lines of code. As a result, it requires low effort to understand the code and it is thus easy for model development and debugging. Second, Simian features a very simplistic application programming interface (API). To maximize portability, Simian requires minimal dependency on third-party libraries. Third, Simian takes advantage of just-in-time (JIT) compilation for interpreted languages. For certain models, Simian has demonstrated capable of outperforming the C/C++ based simulation engine.

To develop models on Simian, it is necessary to understand the Simian API, which contains only three main modules: the simulation engine, entities, and processes. A *simulation engine* is a logical process responsible for synchronizing with other logical processes (using a simple window-based conservative synchronization mechanism). *Entities* are containers for state (such as a network switch or a compute node). The entities also contain event handlers (called services in Simian). An entity can communicate with others by scheduling services at the other entities. *Processes* are independent threads of execution on the entities. Simian uses lightweight threads to implement the processes—greenlets in Python and coroutines in Lua.

2.1.2 Memory System

The processor memory has also gone through rapid changes in architectural design for increased capacity and performance. [DBM⁺11] projected that the memory capacity would reach as much as 128 peta bytes, mixing different technologies from DRAM to non-volatile memory with varying performance aspects (in terms of throughput, access latency, etc.) There exist a number of simulators in the literature for modeling the memory system.

Among the early efforts for simulation of memory system, CACTI [WJ96] is perhaps the most versatile tool with capability to model memory hierarchy at various levels: registers, buffers, caches, main memory. Another well-established memory simulator is DRAMsim [WGT⁺05]. It is an open-source cycle-accurate DRAM simulator, which supports various DRAM types, including SDRAM, DDR, DDR II Memory. The simulator considers various components of DRAM (e.g., DRAM memory controllers, DRAM modules for bits/data storage, and buses through which the DRAM modules communicates), and model them at great detail. It can also model the power consumption of DRAMs. DRAMsim2 [RCBJ11] is an extension of DRAMsim where it can simulate DDR II and III memory systems. It is important to note that both DRAMsim and DRAMsim2 are publicly available, and they can be incorporated with other simulators (e.g., gem5 as noted in Subsection 2.1.3) as part of a large system simulation framework.

There are also many recently-proposed memory simulators to support simulation of DRAM memory system. For example, USIMM [CBS⁺12] is a DRAM main memory system simulator with support for power model based on the Micron power calculator [Jan10]. The USIMM simulator is capable of working with multiple workload traces, where each trace represent a different program being executed on a dif-

ferent processor. USIMM provides interfaces to trace-based processor model. The simulator supports a number of memory scheduling algorithms proposed in the literature, including FCFS, credit-fair, power-down, close-page, first-ready-round-robin, and MLP-aware. The trace based nature of the simulator is both an advantage and disadvantage at the same time. DrSim [JYE12] is another open-source cycle-based DRAM memory system simulator. A main feature of this simulator is its focus on achieving flexibility, which makes it easy to incorporate a variety of DRAM system topologies. Ramulator [KYM15] is a recent open-source simulator for current and future DRAM systems. Ramulator aims to be fast, efficient and easily extensible. Currently, it provides cycle-accurate performance models for a variety of DRAM standards (such as DDR III/IV, LPDDR3/4, GDDR5, SALP, AL-DRAM, etc.) Ramulator’s memory model has been validated against an actual implementation of DDR3 memory (with Micron’s DDR3 Verilog model). Compared to the other simulators thus far, Ramulator can be shown to have achieved the best simulation performance.

2.1.3 Processor System

The processor architecture in HPC system has gone through perhaps the most rapid changes in recent years. Introduction of multicore and manycore architectures, support for various instruction sets, and the arrival of accelerator technologies (such as GPUs) are some examples of recent changes. Many simulators exist; a few have been proposed recently to incorporate new processor architectures with modified capabilities. They differ mostly in terms of how many instructions can be executed per second, how many cores they can support, and how accurately they can replicate the instruction execution behavior. In this subsection, we present some of the well-known processor models.

Processor simulators have gone from a single processor (core) trace-based, functional, atomic, in-order, and in-order with Cycles Per Instruction (CPI) of one, to out-of-order cycle-accurate multicore simulators. The most famous and most widely used single processor/core simulator is SimpleScalar [ALE02]. It simulated almost all of the complex interactions a “modern” superscalar processor (at the time) would have from reorder buffer, multiple issue, register renaming, complex branch prediction schemes, caches, and Simultaneous multithreading [TEL95] (SMT, now known as Hyperthreading in Intel Processors). SimpleScalar implemented Alpha and PISA Instruction Set Architectures (ISAs). After Alpha died as a commercial processor, SimpleScalar suffered the stigma of an outdated ISA even though RISC ISAs were still similar to what Alpha implemented. Depending on which version and which research group used it, the simulator lacked a realistic memory system that simulated contentions on buses connecting the caches to the memory system and the contention in the DRAMs themselves and by the time that was widely available, Chip Multiprocessors (CMPs) now known as multicores had taken the processor world by storm. The advent of such complex processors pushed researchers to implement multicore simulators that spanned a wider range of capabilities. RSIM [PRA97] was the only multiprocessor simulator that was publicly available but it was not maintained and thus was outdated by the time multicores were fashionable.

Some of the most known multicore processor simulators that came out publicly were: SIMICS [MCE⁺02], GEMS [MSB⁺05], M5 [BDH⁺06] and lately GEMS merged with M5 generating the gem5 [BBB⁺11]. gem5 is a simulator that tries to simulate with varying degrees of accuracy and speed for different components of a multicore system. It has the entire spectrum from atomic cores to cycle-accurate out-of-order cores. It does have a memory subsystem including caches and coherence protocols. It also can accommodate an on-chip interconnection net-

work (i.e., Network on Chip “NoC”) model/simulator (e.g., Topaz [APM⁺12] or Garnet [AKPJ09]) both independently implemented. One can implement his own coherence protocol and plug it in seamlessly. As we pointed out in Subsection 2.1.2, DRAMSim and DRAMSim 2.0 are two examples of how flexible gem5 is. It allows hooking up a detailed cycle-accurate memory system simulator seamlessly. GPGPUs as accelerators that have taken their fair share of publicity also has an existence in gem5 where there is an implementation of a GPGPU integrated into gem5 (called gem5GPGPU [WSS⁺12]).

Another aspect of gem5 is that it can perform either in System Call Emulation (SE) mode or in Full System (FS) mode. This last point pertains to the interaction of the programs running on the processor and the operating system (OS). In system call emulation mode, the simulator fakes the existence of an OS by only implementing the minimal set of services that a program needs to run (e.g., reading and writing files). Whereas, in Full System mode, the simulator really boots up an OS (e.g., Linux or Android) and the user gets a command prompt to run his or her program(s) on the simulator. There is a trade-off between these two modes, one gives a more realistic view of the interaction in a real system where the OS interacts and influences the benchmarks/applications (e.g., database applications, OLTP, Data Mining etc.), as opposed to the other applications where such interactions are not present and only the application/benchmark performance is observed (e.g., most scientific applications). Full system mode is far slower than system emulation mode and likewise an out-of-order cycle-accurate core will be far slower than an in-order CPI of one core. At the end of the day, it is a judgment call and an experimental design parameter that has to be made consciously by the research team.

The main advantages of gem5 are the facts that it is a community research project and that it is highly extensible. It does support a variety of instruction

sets spanning commercially available CPUs such as x86 and ARM as well as famous outdated essentially extinct ISAs such as SPARC and Alpha. A new branch is implementing the RISC-V [WLPA11] open-source ISA. gem5 has taken the place of SimpleScalar in the processor simulation world where it is the defacto processor simulator.

There are many other recent multicore and manycore simulators. For example, McSimA+ [ALSJ13] is a lightweight, flexible, open-source simulator with detailed models for the micro-architecture of uni-core, multicore and manycore processors. The overall simulator design is divided into two parts: the (front-end) functional simulation and the (back-end) timing simulation. The functional simulation is based on Pin, a dynamic binary instrumentation tool [LCM⁺05] for generating the instruction stream. The timing simulation is based on an event-driven engine, responsible for calculating the correct timing for different operations (such as cache access, and packet traversal). McSimA+ supports simulation of various asymmetric core structures, hierarchical cache architectures (e.g., private, shared, and non-blocking), NoCs (e.g., buses, crossbars, ring, and 2-D mesh), memory controller, and main memory. McSimA+ has been shown to achieve good accuracy by comparing with previously published results and with real machine runs. McSimA+ has demonstrated to be able to scale to a processor with thousands of cores. Like McSimA+, ZSim [SK13] is another multicore simulator that has been shown to be lightweight, accurate, fast, and scalable. The simulator is lightweight through the use of a user-level virtualization technique. It is accurate for its instruction-driven timing models and leveraging dynamic binary translation. The simulator is fast and scalable, and runs in parallel.

Manifold [WBB⁺14] is another multicore simulator. It can support full system simulation, including, for example, operating system and system binaries, and

can incorporate various models for transient and steady-state simulation of power, thermal, energy and reliability. It has an open architecture with a component-based design, so that new components can be easily incorporated through community efforts. Manifold uses QSim [KRY12], which is a multicore emulator based on dynamic binary translation. The emulator communicates with the back-end timing model for transparent parallel discrete-event simulation. Experiment has not shown, however, that Manifold can scale to larger systems with more than 64 cores. Also, its performance can only achieve a few thousand instructions per second. This probably can be attributed partially to the fact that the simulator considers full system multicore architecture simulation (including processors, cache, memory). Graphite [MKK⁺10] is yet another multicore processor simulator that aims to scale to thousands of cores. One of its main strengths is that it is designed to run in parallel. Other processor simulators exist such as ESESC [AR13], SimFlex [WWF⁺06], and MARSS [PACG11].

2.1.4 HPC Applications

HPC applications exhibit distinct behaviors. An accurate model for these applications is crucial for determining the impact of technological advances in novel HPC architectures. Some of the HPC applications are data-intensive (e.g., molecular dynamics simulation and computational fluid dynamics applications). Some are communication-intensive (e.g., NERSC MIMD Lattice Computation application and NAS parallel benchmark applications). And yet others are I/O-intensive. There exist many application models and analysis tools in the literature to describe the behavior of the HPC applications with various details.

Vampir [KBD⁺08] is a performance analysis tool for parallel MPI/OpenMPI applications. Vampir consists of two major components: a runtime instrumentation

and measurement system and a visual analysis tool. The former supports program instrumentation in different programming languages. It also supports different types of programs: sequential programs, MPI programs, OpenMP programs, and hybrid MPI and OpenMP programs. Vampir also supports various types of instrumentation, including compiler instrumentation, library instrumentation, and manual instrumentation. It provides runtime measurement capability to capture dynamic application behaviors (such as application’s memory usage, I/O performance, user-defined performance counters, etc.)

Tuning and Analysis Utilities (TAU) is a well-established, flexible, portable, robust performance instrumentation, measurement, analysis and visualization framework for HPC applications [SM06]. TAU provides flexible instrumentation capability, allowing the user to select performance instrumentation at different levels of application code. The instrumentation provides various performance information, including various system events and user-defined events, which can be later used for profiling and tracing. HPCTOOLKIT [ABF⁺10] is another application performance measurement, analysis, and presentation toolkit for both sequential and parallel applications. Instead of using source code instrumentation, HPCTOOLKIT works directly with application binaries. HPCTOOLKIT provides effective application analysis by providing measurement ability for a number of derived performance metrics (e.g., peak and actual performance difference rather than simple raw data such as operation counts).

The above tools can be used for measuring and displaying the runtime performance of specific applications. There also exist several analytical models to capture high-level performance. Performance and Architecture Lab Modeling Tool (Palm) is an analytical performance model for parallel applications [TH14]. Palm performs static and dynamic analysis of the source code and generates a tree-like hierarchical

data structure following some well-defined rules. Palm combines top-down semantic insight and bottom-up static and dynamic analysis capability for parallel application execution. Aspen [SV12] is a domain-specific language for analytical performance modeling to enable exploration of novel algorithms and architectures. A formal definition in Aspen includes application behavior (e.g., parameters, kernels, control flow) and the abstract machine (e.g., node, interconnect, cache, memory, core).

2.2 HPC Power Models

HPC systems consume a massive amount of energy to support their operation. It is well understood that effective energy-saving techniques must be considered in future HPC systems. Power and performance prediction models are essential for designing and evaluating energy-saving algorithms for these HPC systems. Many power and performance modeling techniques have been proposed in the literature.

There are a number of analytical models for power and performance prediction. [SBM09] proposed an analytical model for real-time prediction of processor and system power consumption. Performance Monitoring Counters (PMCs) are used to estimate power consumption of the processors. The proposed prediction model is based on linear regression models where the power consumption is assumed to be a piece-wise linear function. Similarly, [SBK13] proposed a unified quasi-analytical performance and power model, which combines analytical modeling and empirical analysis. The proposed method combines application analysis description with different computation and communication parameters obtained through micro-benchmarking to assess the impact of different applications on the performance and energy efficiency of the target HPC system.

Next, we discuss some of the existing tools for power prediction. [HRT⁺12] proposed a power, area and thermal modeling framework with leakage power prediction capability for large-scale HPC simulation on SST. The framework provides power and performance prediction capability of the entire HPC system (including processor, memory and network subsystems) by incorporating different power model libraries, including McPAT [LAS⁺09], HotSpot, IntSim, and ORION [KLPS09], to support analysis of different system components (such as core, shared cache, memory controller and network-on-chip). Multiple Metrics Modeling Infrastructures (MuMMI) environment provides a platform to facilitate analysis, modeling and prediction of power, performance, and power-performance tradeoffs of parallel applications on multicore systems [WCM⁺13]. For example, [WTCM16] proposed performance and power models based on hardware performance counters with CPU frequency. They used non-negative multivariate regression analysis to build models for application execution time, system power, CPU and memory power, using a small set of major performance counters and CPU frequency.

A different line of work on energy modeling and prediction focuses on data communication in scientific applications. [DGML13] proposed a power and energy prediction method for MPI communication operations. The method takes two steps: at the calibration step, the power consumption of node and switch are measured experimentally using an external power meter, considering the underlying features of the supercomputer architecture; during the estimation step, the model estimates the energy consumption of the communication operations using data obtained from the calibration step with the user-provided program and runtime parameters (such as the number of nodes, number of cores per node, etc.) The proposed method has been validated at a large-scale HPC platform (Grid5000), and has been shown to achieve accurate energy prediction for different broadcast algorithms. [GRP⁺13]

also proposed an energy-performance tradeoff for large-scale complete HPC system, with particular consideration of energy consumption during communication phases. The proposed analytical model uses a compiler-based architecture-independent application analysis tool (Byfl) to identify different data-centric operations, including the number of memory accesses, and the number of operations performed by the application. The model captures energy consumption both for MPI communications and shared memory communications.

CHAPTER 3

RAPID PERFORMANCE MODELING FOR HPC SYSTEMS

Interconnection network is a critical component of high-performance computing architecture and application co-design. For many scientific applications, the increasing communication complexity poses a serious concern as it may hinder the scaling properties of these applications on novel architectures. It is apparent that a scalable, efficient, and accurate interconnect model would be essential for performance evaluation studies. In this part of the dissertation, we present interconnect models for predicting the performance of large-scale applications on high-performance architectures. In particular, we present sufficiently detailed interconnect models for Cray’s Gemini 3-D torus, IBM’s Blue Gene/Q 5-D torus, Cray’s Aries dragonfly, and Infiniband’s fat-tree network.

3.1 Background

Recent years have witnessed dramatic changes in high-performance computing to accommodate the increasing computational demand of scientific applications. New architectural changes, including the rapid growth of multi-core and many-core systems, deeper memory hierarchies, complex interconnection fabrics that facilitate more efficient data movement for massive-scale scientific applications, have complicated the design and implementation of the HPC applications. Translating architectural advances to application performance improvement may involve delicate changes to sophisticated algorithms, to include new programming structures, different data layouts, more efficient buffer management and cache-effective methods, and alternative parallel strategies, which typically require highly skilled software architects and domain scientists.

Modeling and simulation plays a significant role, in identifying performance issues, evaluating design choices, performing parameter tuning, and answering what-if questions. It is thus not surprising that there exists today a large body of literature in HPC modeling and simulation, ranging from coarse-level models of full-scale systems, to cycle-accurate simulations of individual components (such as processors, cache, memory, networks, and I/O systems), to analytical approaches. We note, however, that none of the existing methods is capable of modeling a full-scale HPC architecture running large scientific applications in detail.

To do so would be both unrealistic and unnecessary. Today’s supercomputers are rapidly approaching exascale. Modeling and simulation needs to address important questions related to the performance of parallel applications on existing and future HPC systems at similar scale. Although a cycle-accurate model may render good fidelity for a specific component of the system (such as a multi-core processor) and a specific time scale (such as within a microsecond), the model cannot be naturally extended to handle arbitrarily larger systems or longer time durations. Partially this is due to the computational complexity of the models (both spatial and temporal). More importantly, no existing models are known capable of capturing the entire system’s dynamics in detail. HPC applications are written in specific programming languages; they interact with other software modules, libraries and operating systems, which in turn interact with underlying resources for processing, data access, and I/O. Any uncertainties involved with the aforementioned hardware and software components (e.g., a compiler-specific library) can introduce significant modeling errors, which may undermine the fidelity achieved by the cycle-accurate models for each specific component.

George Box, a statistician, once said: *“All models are wrong but some are useful.”* In order to support full-system simulation, we must raise the level of modeling

abstractions. Conceptually, we can adopt an approach, called “*selective refinement codesign modeling*”, where we begin with both architecture and application models at coarse level, gradually refine the models with potential performance bottlenecks, and eventually stop at models sufficient to answer the specific research questions. This iterative process is based on the assumption that we can identify performance issues from the models in a timely manner. To do so, we need to develop methods that facilitate rapid and yet accurate assessment and performance prediction of large-scale scientific applications on current and future HPC architectures.

Performance prediction of large-scale parallel computers consisting thousands of node and more is a challenging task. In recent years, we have witnessed the fast growth in supercomputer design that can perform operations at scale of quadrillions of calculations per second. The tremendous rise in the computational power is in part attributed to the government agencies that have been supporting (and encouraging) the growth of large-scale supercomputing infrastructures. For example, significant investment by the U.S. Department of Energy (DOE) on building state-of-the-art supercomputers through programs (such as FastForward [VSC12], and recently FastForward 2 [Dep14]) support the fact that exascale computing will continue to receive attention in years to come. Consequently, the community faces a significant challenge for complex large-scale scientific and engineering applications to keep up and take full advantage of the fast growth of supercomputing capabilities.

We design and develop a simulator, called the *Performance Prediction Toolkit (PPT)*. Four major aspects distinguish our effort from other existing approaches. First, our simulator needs to easily integrate large-scale applications (especially, computational physics code) with full-scale architecture models (processors, memory/cache, interconnect, and so on). Second, our simulator must be able to combine selected models of various components, potentially at different levels of modeling ab-

straction, providing a trade-off between the computational demand of the simulator and the accuracy of the models. Third, the simulator needs to adopt a minimalistic approach in order to achieve a short development cycle. It is important that new models can be easily incorporated in the simulator; the simulator needs to keep up with the fast refresh rate of HPC systems. Last, the simulator must be able to achieve scalability and high performance; it needs to be capable of handling extremely large-scale models, e.g., using advanced parallel discrete-event simulation techniques.

PPT relies on the Simian [SEL15], a parallel discrete-event simulation engine, and essentially consists of libraries of hardware models, application models, and middleware models. PPT, along with Simian, is designed to be lean, written in Python (or alternatively Lua) with minimal reliance on third-party libraries in an effort to keep the code simple, understandable, and yet offer high performance. We also present scaling runs of our interconnect model, which confirm the scalability of the underlying simulation engine.

3.2 Related Work

Many HPC simulators exist. Here we focus on those that provide interconnection network models. Some of these simulators aim at full-system simulation, where parallel applications are simulated to their behavior on the target architecture. BigSim [ZKK04] falls into this category. BigSim is built on Charm++ for scalable performance, which is an object-based and message-driven parallel programming system [KK93]. BigSim adopts an optimistic approach using the inherent determinacy of the target parallel applications to reduce the overhead of the optimistic scheme. Experiments show that BigSim is capable of scaling up to 64K processors.

The interconnection network model implemented in BigSim, however, is relatively simple. For example, it does not consider network congestion in detail [CGL⁺14]. To study the performance of large-scale MPI applications, $\mu\pi$ is an MPI simulator based on an efficient conservatively-synchronized parallel simulator that features a process-oriented world-view [Per10]. Experiments show that the simulator is capable of simulating hundreds of millions of MPI ranks running on parallel machines. However, $\mu\pi$ does not have any reasonably detailed interconnection network model.

The Extreme-scale Simulator (xSim) is a performance-prediction toolkit for future HPC architectures [BE11]. xSim applies parallel discrete-event simulation using lightweight threads to achieve scalability up to millions of application processes [EL10, Eng14]. xSim also incorporates different network topologies, including star, ring, tree, mesh, and torus [JE11]. However, unlike our interconnect model, network congestion is omitted in xSim to gain scalability. As such, their simulator cannot accurately model the blocking behavior of the target interconnection network which may be of importance to the architecture/application co-design.

The Structural Simulation Toolkit (SST) [RHB⁺11] is a comprehensive simulation framework for modeling large-scale HPC systems, including processors, memory, network, and I/O systems. It attempts to achieve scalability using a conservative parallel simulation approach. SST can model hardware components with different granularity and accuracy. SST's network model in particular contains a variety of interconnect topologies: binary tree, fat-tree, hypercube, butterfly, mesh, and so on. The interconnect model, however, does not provide the necessary details for capturing important network behaviors for performance prediction. For example, it does not support network flow control and also the links are assumed to have infinite capacity. Our interconnect model, on the contrary, provides packet-level

details that can support realistic network scenarios, such as the transient network congestion occurred during the execution of large complex applications.

The CODES simulator [CLL⁺11] is a comprehensive simulation platform that can model various large-scale HPC systems, including storage systems, interconnection networks, HPC and data center applications. CODES is built on ROSS, a parallel discrete-event simulation engine using reverse computation [CBP02]. CODES provides detailed models for various interconnect topologies, including torus [LCC⁺12], dragonfly [MCRC12], and fat-tree [LHSJ15]. CODES has shown capable of simulating large-scale interconnect configurations (with millions of nodes). A recent paper has proposed a trace-driven simulator (TraceR) to replay large execution traces to predict and understand network performance and behavior [AJB⁺15]. TraceR is built upon ROSS-based CODES simulator and has been shown to be able to simulate a network consisting of half million nodes using traces produced by running BigSim applications.

Although CODES is complementary to our work, there are three major differences. First, the interconnection network models in CODES are phit-level models that can capture more detailed transactions related to virtual channels than the corresponding packet-level models in PPT. While conceptually, more detailed models may render higher simulation fidelity, the computational demand would be much higher (by as much as several orders of magnitude). As such, a performance study using CODES typically would only focus on simple operations (for example, a random send/receive pattern or one collective call) and at a much smaller time scale, while using PPT we can study more complex application behaviors with greater efficiency and flexibility. In terms of accuracy, our experiments show that PPT's interconnection models can reasonably produce performance results that match from other empirical studies.

Second, CODES does not have a full-fledged MPI model. On the contrary, the interconnection network models in PPT are fully integrated with the MPI implementation from design. In this way, one can easily model complex application behaviors in PPT.

Last, the interconnection network models in PPT are designed to reflect real implementations (e.g., Cray’s Gemini, Aries, IBM Blue Gene/Q, and Infiniband). In doing so, we can study the performance of applications over various interconnection networks of real (either existing or planned) HPC systems.

3.3 Model

To design interconnect model for performance prediction, one need to take several important factors into account:

- *Scale*: The interconnect model must be able to accommodate high-performance computing platforms and applications at extreme scale.
- *Performance*: The interconnect model must run reasonably fast so that it can be used to explore design alternatives of system architectures, software, and parallel applications.
- *Accuracy*: The interconnect model must provide high fidelity sufficient to represent the effect of important design decisions, constraints and optimizations. Simple analytical models may not be sufficient for projecting the performance of dynamic, complex applications.
- *Integration*: The interconnect model must be easy to integrate with other models, including those for processors, memory, and file systems. It is also important that the model can be readily integrated with common software tools, such as MPI,

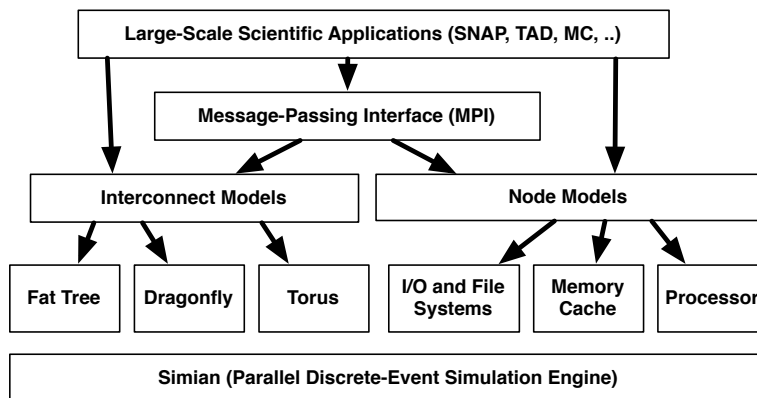


Figure 3.1: An architectural design of PPT.

so that various scientific applications can be easily incorporated in the performance study.

PPT is designed specifically to allow rapid assessment and performance prediction of large-scale scientific applications on existing and future high-performance computing platforms. More specifically, PPT is a library of models of computational physics applications, middleware, and hardware that allows users to predict execution time by running stylized pseudo-code implementations of physics applications. Fig. 3.1 presents an architectural design of PPT.

PPT models are highly parameterized for applications, middleware, and hardware models, allowing parameter scans to optimize parameter values for hardware-middleware-software pairings. PPT does not yield cycle-accurate performance metrics. Instead, the results from PPT are used to examine underlying algorithmic trends and seek bottlenecks to on-node performance and scaling on HPC platforms. The conclusions of such analysis may range from optimization of current methods to further investigation of more substantial algorithmic variations.

An application is a stylized version of the actual application that captures the loop structure of important numerical kernels. Not all elements of the code are

included in a PPT application, and it does not predict numerical accuracy of an algorithm, but rather predicts the execution time of a given job instance.

Middleware models in PPT currently include only MPI. It interfaces with the PPT application models and implements the communication logic in the loop structure of the actual applications.

Hardware models exist for interconnection networks and compute nodes. PPT's interconnection models are fully integrated with the MPI model. The interconnection models implement different network topologies and can be set up with different configurations. The library consists of configurations for various common production interconnection networks.

The node models in PPT use hardware parameters clock-speed, cache-level access times, memory bandwidth, etc. Application processes can advance simulated execution time by calling a compute-function with a task list as input, which consists of a set of commands to be executed by the hardware, including, for example, the number of integer operations, the number of floating-point operations, the number of memory accesses, etc. The hardware model uses this information to predict the execution time for retrieving data from memory, performing ALU operations, and storing results.

3.3.1 MPI Model

The message-passing interface (MPI) model is commonly used by parallel applications and is one of the most popular parallel programming tools on today's HPC platforms.

The design of an MPI implementation is intricately influenced by the underlying interconnection network. For example, Cray's MPI implementation for the Gemini interconnect uses Fast Memory Access (FMA). It allows a maximum of 64 bytes

of data transfer for each network transaction. A network transaction initiates a single request from the source to the destination, which triggers a response from the destination back to the source. A large message will get broken down into many individual 64-byte transactions. There are two types of transactions. A typical PUT transaction sends 64 bytes of data from a source to a destination. A PUT message consists of a 32-phit request packet (i.e., 96 bytes, where each phit is 24 bits). Each PUT message is followed by a 3-phit response packet (9 bytes) from destination to source. A typical GET transaction consists of a 8-phit request packet (24 bytes), followed by a 27-phit response packet (81 bytes), including 64 bytes of data.

To design an MPI model for Gemini, we need to incorporate the FMA request and response scheme at the level of each network transaction. Cray’s MPI implementation uses both PUT and GET protocols, the decision of which to use depends on the data size [PVB⁺13]. It was observed that, for data size up to 4K bytes and also beyond 256K bytes, Cray’s MPI uses PUT. For data size between 4K and 256K bytes, MPI chooses GET. In our implementation, we only use PUT for simplicity. Since both PUT and GET transactions have a total of 105 bytes of traffic for each request and response pair between the source and the destination, we expect that the effect of selecting between PUT and GET, both in terms of network latency and bandwidth, would be rather insignificant.

In our model, upon receiving a send request of a large MPI message, the MPI sender needs to break down the message into individual PUT requests of at most 64 bytes each. Each message will be sent over the network with an extra 32-byte message overhead. Upon receiving the PUT request, the MPI receiver responds with a 9-byte ACK. We implemented a message retransmission mechanism to ensure reliable data delivery of the MPI messages.

As another example of how the underlying interconnection network determines the details of the MPI implementation, we present an example based on dragonfly topology. Cray’s XC series network uses Aries dragonfly interconnect [AFKR12]. MPI uses Fast Memory Access (FMA) for message passing. Messaging are performed as either GET or PUT operations (depending on the size of the message). A PUT operation initiates data flow from the source to the target node. When a packet reaches destination, a response from the destination is returned to the source. FMA allows a maximum of 64 bytes of data transfer for each network transaction—larger messages must be broken down into individual 64-byte transactions. A PUT message consists of a 14-phit request packet (i.e., 42 bytes, where each phit is 24 bits). Each request packet is followed by a 1-phit response packet (3 bytes) from destination to source. A GET transaction consists of a 3-phit request packet (9 bytes), followed by a 12-phit response packet (36 bytes) with 64 bytes of data.

To easily incorporate scientific applications that use MPI, we take advantage of Simian’s process oriented design. As we mentioned earlier, each compute node (host) is by itself a Simian entity. Different compute nodes communicate by sending and receiving events (via scheduling services in Simian). We implemented each user MPI process as a Simian process on the compute node. This allows each user MPI process to run independently from other MPI ranks as well as other system-level simulation processes.

Fig. 3.2 provides an example showing how to start the MPI processes on a simulated HPC cluster. The program starts by calling `HPCSim` to instantiate the model for the entire cluster, including the interconnect model and the compute nodes. Model parameters are passed as an argument in the form of a python dictionary. Most common hardware configurations are preset in PPT for easy reuse and cus-

```

from ppt import *

# config hopper (17x8x24 gemini interconnect)
model_cfg = { # a dictionary
    "intercon_type" : "gemini",
    "host_type" : "mpihost",
    "torus" : configs.hopper_intercon,
    "mpiopt" : configs.gemini_mpiopt,
}
model = HPCSim(model_cfg, ..)

# mpi main function, n is matrix dimension
def cannon(mpi_comm_world, n):
    ... # we describe this later

# start 16 mpi ranks, pass matrix dimension
model.start_mpi(range(16), cannon, 10000)

# simulation starts
model.run()

```

Figure 3.2: An example showing running 16 MPI processes on Hopper.

tomization, including those parameters that are needed by the MPI implementation for specific interconnection networks.

The `start_mpi` function creates the MPI processes on the designated compute nodes. To allow maximum flexibility, we require the users to specify a mapping from the MPI ranks to the host IDs. The first argument to `start_mpi` is a list. In the example, the simulator creates 16 MPI processes and maps them to 16 compute nodes. On the other hand, if a compute node contains multiple cores (say, 4), one may want to allocate as many MPI ranks to the compute node. This can be easily achieved by specifying a list in python, like: `[i/4 for i in range(n)]`.

Each MPI process is simply a python function that takes at least one argument: `mpi_comm_world`. Like in a real MPI implementation, it is an opaque data structure that represents the set of MPI processes among which communication may take place. Our design, to a large extent, resembles the MPI API. To illustrate its use,

```

# cannon's algorithm on matrix multiplication
def cannon(mpi_comm_world, n):
    p = mpi_comm_size(mpi_comm_world)
    id = mpi_comm_rank(mpi_comm_world)
    # use p, id to calc i, j, and neighbor ranks

    # time for reading/initing submatrices
    sleep(sometime) # proportional to m^2

    # shift A(i,j) left by i columns
    mpi_sendrecv(left_i, None, m*m*8, right_i, mpi_comm_world)
    # shift B(i,j) up by j rows
    mpi_sendrecv(up_j, None, m*m*8,
                 down_j, mpi_comm_world)

    for r in range(sqrt(p)-1):
        # time for multiplying A(i,j) and B(i,j)
        sleep(sometime) # proportional to m^3

        # shift A(i,j) to the left
        mpi_sendrecv(left, None, m*m*8,
                    right, mpi_comm_world)
        # shift B(i,j) upward
        mpi_sendrecv(up, None, m*m*8,
                    down, mpi_comm_world)

    mpi_finalize(mpi_comm_world)

```

Figure 3.3: Simulating Cannon's matrix multiplication.

we use a simple example of Cannon's matrix multiplication algorithm [Can69]. The algorithm applies a 2-D block decomposition of the matrices. Suppose the dimension of the matrices is $n \times n$, each processor would be in charge of calculating a sub-matrix of size $m \times m$, where $m = n/\sqrt{p}$, and p is the total number of MPI ranks (assuming it's a square number).

Fig. 3.3 shows a simulation of the Cannon's algorithm. As we see, the program captures the main execution skeleton of the algorithm. The timing calculation for loading and initializing the sub-matrices and for multiplying the sub-matrices depends on the processor, cache/memory, and file system models that we

Table 3.1: Implemented MPI Functions

<code>MPI_Send</code>	blocking send (until message delivered to destination)
<code>MPI_Recv</code>	blocking receive
<code>MPI_Sendrecv</code>	send and receive messages at the same time
<code>MPI_Isend</code>	non-blocking send, return a request handle
<code>MPI_Irecv</code>	non-blocking receive, return a request handle
<code>MPI_Wait</code>	wait until given non-blocking operation has completed
<code>MPI_Waitall</code>	wait for a set of non-blocking operations
<code>MPI_Reduce</code>	reduce values from all processes, root has final result
<code>MPI_Allreduce</code>	reduce values from all, everyone has final result
<code>MPI_Bcast</code>	broadcast a message from root to all processes
<code>MPI_Barrier</code>	block until all processes have called this function
<code>MPI_Gather</code>	gather values from all processes at root
<code>MPI_Allgather</code>	gather values from all processes and give to everyone
<code>MPI_Scatter</code>	send individual messages from root to all processes
<code>MPI_Alltoall</code>	send individual messages from all to all processes
<code>MPI_Alltoallv</code>	same as above, but each can send different amount

ignore here. The MPI calls are mapped to the real MPI functions. We implemented most common MPI functions. Table 3.1 summarizes the functions included in our MPI model. In addition to the MPI functions specified in the table, we also support functions that deal with groups and communicators. They include: `MPI_Comm_split` (create new sub-communicators), `MPI_Comm_dup`, `MPI_Comm_free`, `MPI_Comm_group` (new group), `MPI_Group_size`, `MPI_Group_rank`, `MPI_Group_incl` (new subgroups), `MPI_Group_excl`, `MPI_Group_free`, `MPI_Cart_create` (new cartesian communicator), `MPI_Cart_coords`, `MPI_Cart_rank` and `MPI_Cart_shift`.

3.3.2 Interconnection Network Models

In this subsection, we present details of our interconnection network models. First, we outline implementation of torus-based Gemini and Blue Gene/Q interconnects. Next, we describe our dragonfly interconnect model and then focus specifically on Cray’s Aries interconnect that has been applied in many real HPC systems. Finally,

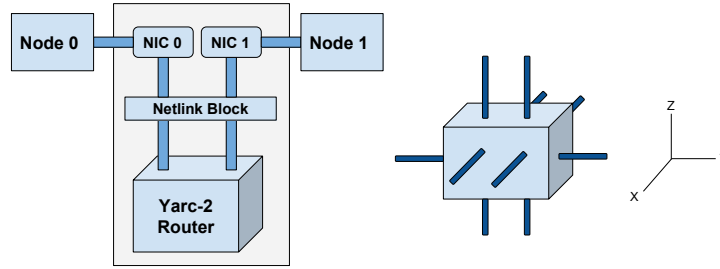


Figure 3.4: Cray Gemini ASIC block diagram.

we present our fat-tree topology model design and an implementation of Infiniband interconnect architecture based on fat-tree topology.

3.3.2.1 Torus-Based Interconnect Model

We designed and implemented a relatively detailed model for the Gemini interconnection network. Gemini is a part of the Cray’s XE6 architecture. Cray XE6 is a system currently used by many large-scale high-performance computing systems, including, for example, Hopper at National Energy Research Scientific Computing Center (NERSC), Cielo at Los Alamos National Laboratory (LANL), Blue Waters at the National Center for Supercomputing (NCSA), Titan at the Oak Ridge National Laboratory, and ISTeC at Colorado State University.

Each Cray XE6 compute node has two AMD Opteron processors, coupled with its own memory (either 32 GB or 64 GB) and communication interface. The Gemini network was first introduced in 2010 in Cray XE6 systems and was the most notable difference from the earlier Cray XT systems. In Gemini, the two AMD Opteron nodes are connected to the Gemini Application-Specific Integrated Circuit (ASIC) through two Network Interface Controllers (NICs). The NICs have their own HyperTransport (HT) 3 link to connect to the nodes, where the link offers up to 8 GB/s bandwidth per node and direction [PCD⁺13]. The NICs within an ASIC

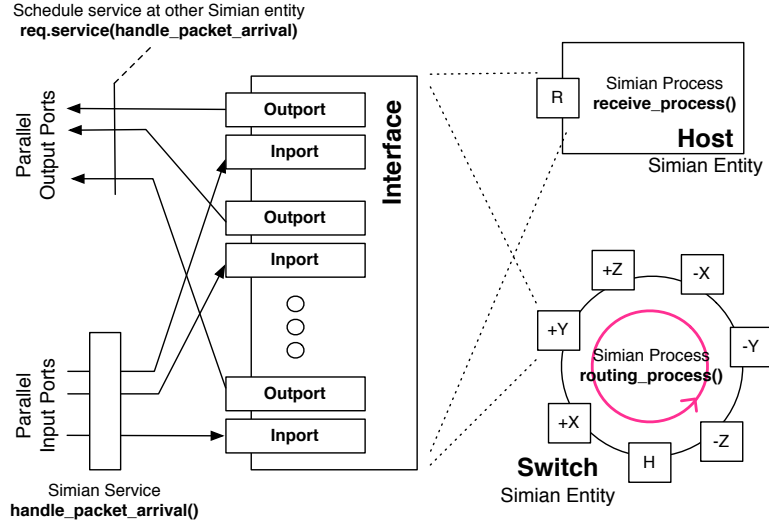


Figure 3.5: Interconnect model using Simian entities, processes, and services.

are connected through a Netlink block, enabling internal communication between the NICs. At the heart of Gemini is a 48-port YARC router (shown in Fig. 3.4), which is configured to construct a 3-D torus topology. The router is connected to Netlink block through 8 links. Each router gives ten torus connection: two connections per direction in the “X” and “Z” dimension and one connection per direction in the “Y” direction.

Unlike some other interconnection networks, such as fat-tree, torus is a blocking network. It is possible that congestion may happen in the network where queuing delays may negatively affect the performance of parallel applications in a significant fashion. It is thus important to model the traffic behavior in the network imposed by high-level applications. To do that, we need to provide a detailed queuing model to capture the interactions of network transactions.

We implemented each compute node (which is also called a host) or interconnect switch as a Simian entity. Fig. 3.5 shows a diagram of the design. The hosts and switches are connected via network interfaces that simulate the queuing behavior. A network interface may consists of multiple ports to handle parallel connections

between the switches (e.g., in the “X” and “Z” dimensions). Each port consists of an output port (“outport”) and an input port (“inport”) for sending and receiving messages. To send data from the output port, Simian schedules a service (i.e., an event handler), called `handle_packet_arrival`, at the next node (which can be either a switch or a host), with a delay that is the sum of the current queuing delay at the output port, the packet transmission time, and the link propagation delay between the two nodes. Upon a packet’s arrival at a switch, the `handle_packet_arrival` service inserts the packet into the buffer of the corresponding input port and informs the routing process. The routing process is a Simian process that takes packets from the input ports, calculates the next hop using the selected routing algorithm, and then forwards the packet to the corresponding output port. If a host receives the packet, the `handle_packet_arrival` service inserts the packet into the input buffer of the host interface and informs the receive process, which hands the packet to the corresponding MPI receiver accordingly.

Gemini supports multiple routing algorithms, such as deterministic, hashed, and adaptive [VDP12]. Each routing algorithm follows dimension-order routing, where “X” dimension is always traversed first, then “Y” dimension, and finally “Z” dimension [PVB⁺13]. Different routing algorithms provide different level of flexibility in using links at dimensions. For example, the deterministic dimension-order routing provides least amount of flexibility, where links are predetermined in each dimension. The adaptive dimension-order routing provides most flexibility, allowing packets to be adaptively scheduled to lightly-loaded links. In our implementation, the user can explicitly select the routing algorithm when configuring the interconnect model.

We extended the Gemini model and developed a generic model for the torus topologies for all dimensions (i.e., 5-D torus, 6-D torus etc.). We now specifically

focus on the interconnect for the Blue Gene/Q architecture, which is based on a 5-D torus topology.

IBM’s Blue Gene/Q is a system currently used by many large-scale high-performance systems (e.g., Sequoia and Vulcan at Lawrence Livermore National Laboratory, Mira at Argonne National Laboratory). Blue Gene/Q is a 5-D torus-based interconnect architecture, where each switch node connects to ten neighboring switches (two in each direction in five dimensions). The network is optimized for both point-to-point and collective MPI communications [IDR16]. The message unit (MU) in Blue Gene/Q supports both direct PUT and remote GET, where messages are packetized for transmission [CEH⁺11]. Data portion of packets increments in chunks of 32 bytes, up to 512 bytes [CEH⁺11]. Packets contain 32-byte header: 12 bytes for the network and 20 bytes for the MU.

3.3.2.2 Dragonfly-Based Interconnect Model

Dragonfly topology was first proposed in [KDSA08]. It is a cost-efficient topology, which reduces network cost through exploiting the economical, optical signaling technologies and high-radix (virtual) routers. Dragonfly topology has a three-tier network architecture [KDSA08]. At first level, each router is connected to p nodes, usually through backplane printed circuit boards (PCBs) links. At second level, a group is formed through connecting a routers to each other. The local connections used in this level are referred to as *intra-group* connections. These connections are typically built using short-length electrical cables. At the last level, each router has h *inter-group* connections to routers in other groups. These global connections are usually built using longer optical cables. The maximum network size of such dragonfly topology is $ap(ah + 1)$ nodes.

In our work, we consider a dragonfly topology with *local link arrangement* to be completely-connected (i.e., a 1-D flattened butterfly). Therefore, each router has $a - 1$ local connections to the other routers in the group. *Global link arrangement* specifies how switch in a group is connected to switch of the other group. Several alternatives for global link arrangements are defined and evaluated against each other in the literature (e.g., consecutive, palmtree, circulant-based [CVB15]). In this work, we consider the *consecutive* arrangement of global links, as also considered in the paper where dragonfly was initially proposed [KDSA08, CVB15]. The consecutive arrangement connects routers in each group consecutively, where groups are also numbered consecutively.

In our dragonfly implementation, we support two types of routing: minimal (MIN) and non-minimal (VAL). MIN routing is ideal for benign traffic patterns (e.g., uniform random traffic). Since we consider completely-connected local channels, any packet can reach destination in at most three hops: one hop within the source group to reach the switch with global connection to the destination group, one hop to traverse the global link and one hop within the destination group to reach the destination node. VAL routing [Val82], on the other hand, is suitable for adversarial traffic patterns. Following the principal of this algorithm, we route a packet to a randomly chosen intermediate group first and then route to the final destination. As a result of using 1-D local channels, a packet generally reaches destination through traversing two global channels and three local channels.

Another dragonfly routing variation is Universal Globally-Adaptive Load-balanced routing (UGAL), which chooses between MIN and VAL on a packet-by-packet basis and sends packet to paths with least queuing delay to alleviate congestion. Since for a large-scale system, it is infeasible to know the queue information on all other queues (UGAL-G), the switching between MIN and VAL can be performed based on

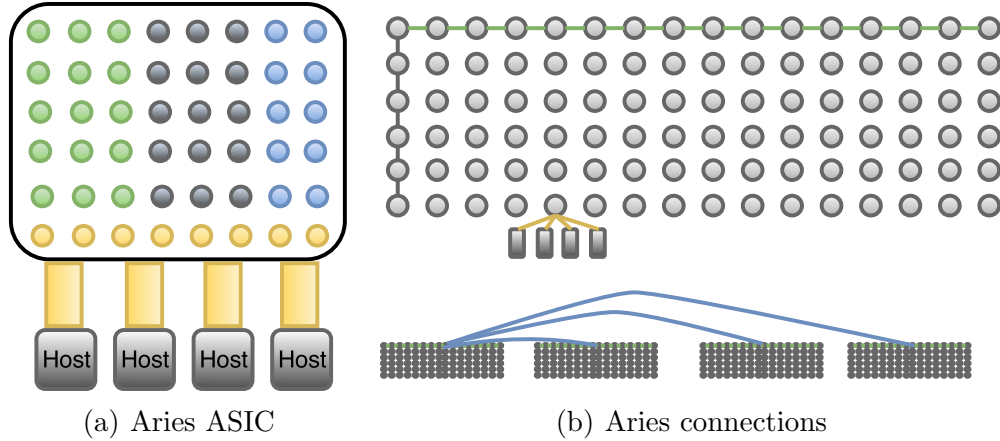


Figure 3.6: Cray Aries block diagram.

local queue information (UGAL-L). We do not consider adaptive routing at all in our current implementation. Our logics behind such decision are the followings: 1) We expect adaptive routing to play a minor role in the overall performance prediction of large-scale HPC applications compared to other factors; and 2) The additional overhead of message communication necessary for implementing adaptive routing would significantly impact scalability of our simulation.

Cray’s Aries network (developed as part of Defense Advanced Research Projects Agency’s or DARPA’s program) uses dragonfly topology [FBR⁺12, AFKR12]. Aries contains a 48-port router, four network interface controllers (NICs) connecting four nodes and a multiplexer known as Netlink. The system is built from four-node Aries blade (where each blade contains a single Application-specific Integrated Circuit or ASIC). A simplified block diagram for Aries ASIC is shown in Fig. 3.6(a). Aries network consists of Cascade cabinets, where each pair of cabinets can house up to 384 nodes. There are three chassis per cabinet, each chassis contains 16 Aries blades. Two such cabinets construct a group (i.e., each group in Cray Aries contains six chassis or 96 Aries blades).

Both minimal (MIN) and non-minimal (VAL) routings are supported in Aries architecture [Alv12]; we implemented both in our model. There are two cases. For *intra-group routing*, MIN routing requires at most two hops, while VAL routing selects a random switch inside the group and thus requires up to four hops. For *inter-group routing*, MIN routing requires at most five hops to reach destination (two local link traversals each for source and destination groups, and one global link traversal), where VAL routing selects a random intermediate group. For the latter, a packet needs to be routed to a random intermediate switch in each of the source group, destination group and intermediate group, thus requiring at most fourteen hops before a packet is reaching its destination.

3.3.2.3 Fat-Tree-Based Interconnect Model

Fat-tree is one of the most widely-used topologies for current HPC clusters and also the dominant topology on Infiniband (IB) technology [Mel16]. Besides, this interconnect has also received significant attraction in data center networks [AFLV08]. Some unique properties that make fat-tree much popular among HPC and data center networks are: deadlock avoidance without use of virtual channels, easier network fault-tolerance, full bisection bandwidth, and so on.

Since its first introduction, many variations of fat-tree have been proposed in the literature (e.g., [LCH04, PV97]). Among them, m -port n -tree fat-tree proposed in [LCH04] and k -ary n -tree [PV97] are the most popular ones. In this work, we implement the m -port n -tree variations due to its wide popularity compared to other existing fat-tree variations [Bog14]. As such, it has been considered in recent literature as the fat-tree topology variation for large-scale systems [LHSJ15].

An m -port n -tree is a fixed-arity fat-tree consisting of $2(m/2)^n$ processing nodes and $(2n - 1)(m/2)^{n-1}$ m -port switches [LCH04]. The height of the tree is $n + 1$.

Each of the switches in an m -port n -tree have a unique identifier based on the level and value of m and n . The processing nodes are the nodes in the leaf and are also denoted uniquely. We use notation scheme outlined in [LCH04] to denote both switches and processing nodes. We connect the switches to each other in both upward and downward directions and also to processing nodes based on the conditions specified in [LCH04].

We implemented routing in the fat-tree network as two separate phases: upward phase and downward phase. In upward phase, the packet is forwarded from a source towards the direction of one of the root switches. In downward phase, the packet is forwarded downwards towards one of the leaf nodes as the destination. Transition between these two phases takes place at the lowest common ancestor (LCA) switch. The LCA switch can reach both the source and destination using downward ports of that switch. Many efforts exist to improve the routing performance in fat-tree network (e.g., Valiant algorithm [VB81], ECMP [Tha00], Multiple LID routing scheme [LCH04]). We implement the Multiple Local Identifier (MLID) routing scheme for Infiniband network, as presented in [LCH04]. MLID routing scheme relieves the link congestion through exploiting multiple paths available in fat-tree topology.

3.3.3 Interconnect Model Validations

Now we describe the experiments for validation of our interconnect models.

3.3.3.1 3-D Torus Model Validation

For these experiments, we measure the model-predicted MPI performance on Cray’s Gemini network and compare that with published results in the literature to validate our interconnect model.

We consider a large-scale interconnect system in real deployment. Hopper was built by National Energy Research Scientific Computing Center/NERSC (a high-performance computing facility of the U.S. Department of Energy (DOE)[WSB⁺11]). It is a Cray XE6 system that consists of 6,384 compute nodes connected via the Gemini interconnect¹. Each compute node contains two 12-core AMD Magny Cours processors running at 2.1 GHz, and DDR3 1.3 GHz RAM (32 GB for each of the 6,000 nodes and 64 GB for each of the rest 384 nodes). The entire system contains a total of 153,216 cores, 212 terabytes of memory, and 2 petabytes of disk. The peak floating point operations per node is 201.6 Gflops. The peak performance of the system has been demonstrated to reach 1.3 petaflops [KT11].

As mentioned earlier, Cray’s Gemini interconnect is a 3-D torus interconnect of high performance [ARK10]. Dimensions of Hopper’s torus network are $17 \times 8 \times 24$. As outlined in the original design and considered in various literature [KBVH14], the peak link speed across the X and Z dimensions is 9.375 GB/sec and in the Y dimension is 4.68 GB/sec. Inter-node latency is measured about $1.27 \mu s$ between the nearest nodes and $3.88 \mu s$ between the farthest nodes across the system. Although topologically it is a regular 3-D torus, Hopper’s interconnect is wired specifically to optimize for the application performance, in which case the hosts are not necessarily named consecutively. To account for that in our model, we provide a mapping from the host IDs to the 3-D torus coordinates of the corresponding interconnect

¹Cray XE6 has been used by many of the largest supercomputing systems over the last decade [KBVH14].

switches [NER15b]. Using this one-to-one mapping, we design the hopper interconnect to closely represent the communication behavior of the applications running on the compute nodes.

The end-to-end latency between two end nodes is determined by the link (propagation) delay and the number of hops between the nodes. For Hopper, the inter-node latency has been reported to be $1.27 \mu s$ between the nearest nodes. Consequently, we configure the link delay between the compute nodes and the corresponding switch in our model to be half of that, which is 635 nanoseconds. The inter-node latency for the farthest nodes on Hopper is measured to be $3.88 \mu s$. Since the network diameter for a $17 \times 8 \times 24$ torus is 24, we can subtract two node-switch link delays from the inter-node latency and divide the results by the network diameter. In this way, we obtain the link delay between the adjacent torus switches to be 108.75 nanoseconds. The result per-hop latency seems to be consistent with the empirical measurement reported in the original design paper [ARK10].

We did a latency test by having an MPI process to send a 4-byte data to all other MPI processes mapped on different compute nodes and measure the end-to-end delay. Fig. 3.7 shows the histogram of the end-to-end delay. The delays are measured between $1.27 \mu s$ and $4.07 \mu s$, which are considered within expectation. We also conducted a latency measurement for MPI collective operations. In particular, we measured the duration of a call to `MPI_Allreduce`, as we vary the number of MPI ranks and the data size. Fig. 3.8 shows the results. As expected, the collective operation has a logarithmic cost in the number of processes under the normal situation. When the number of processes increases along with the data size, part of the network becomes congested and the delay increase superlinearly.

To measure the MPI throughput, we select two compute nodes to run multiple MPI processes; we designate one compute node to run only the MPI senders and

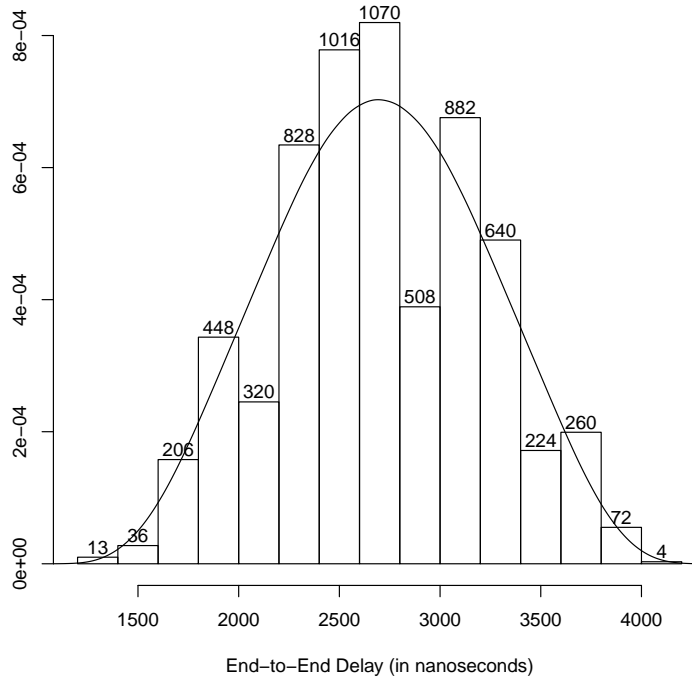


Figure 3.7: A histogram of end-to-end delay between compute nodes of the simulated HPC cluster.

the other only the MPI receivers. We vary the number of the sender and receiver pairs (i.e., the number of processes per node, PPN) to be 1, 2 and 4. Each MPI sender sends a series of MPI messages of a given fixed size back-to-back, using `MPI_Send` call, to the designated MPI receiver on the other host. The MPI receiver simply loops and calls the `MPI_Recv`. We run different experiments varying the size of the MPI messages from 8 bytes to 128K bytes doubling each time between the experiments. To get reasonable bounds of the throughput, we select two extremes: one with the two compute nodes next to each other, and the other with the two nodes farthest apart over the interconnect network.

Fig. 3.9 shows the aggregate throughput of all MPI senders as a function of MPI message size. The performance levels off at 6.75 GB/s when the traffic becomes

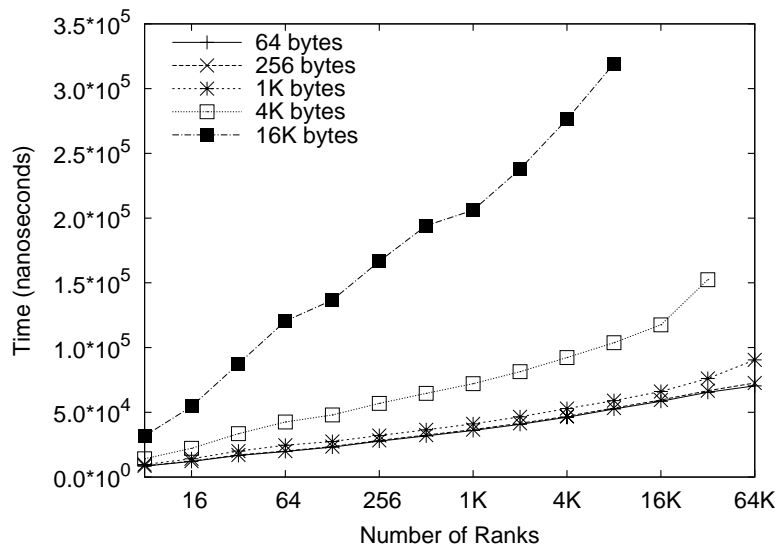


Figure 3.8: Duration of the MPI_Allreduce call for different number of ranks and data size on the simulated HPC cluster.

largely bandwidth constrained. As expected, multiplexing MPI sends at the source host achieves proportionally higher aggregate throughput for small data sizes when the total is less than the bandwidth cap. The throughput between the farthest nodes is lower than that between the nearest nodes due to the increased end-to-end latency.

In Gemini, Fast Memory Access (FMA) is a mechanism for user processes to generate network transactions. In our model, we implemented MPI only as FMA put, where the source can write up to 64 bytes at a time. In Fig. 3.10, we reproduce the Gemini FMA put throughput (solid lines) as a function of transfer size for 1, 2 and 4 processes per node (as published in [ARK10]). We noticed that the FMA put throughput is significantly higher than what we have achieved using MPI, especially at small transfer sizes, although both level off at above 6 MB/s for large transfer sizes. We speculated that this is due to the MPI overhead. On a quiet network, remote put has an end-to-end latency of less than 700 nanoseconds. But with MPI, the end-to-end latency increases to 3.88 μ s between the farthest nodes. To verify

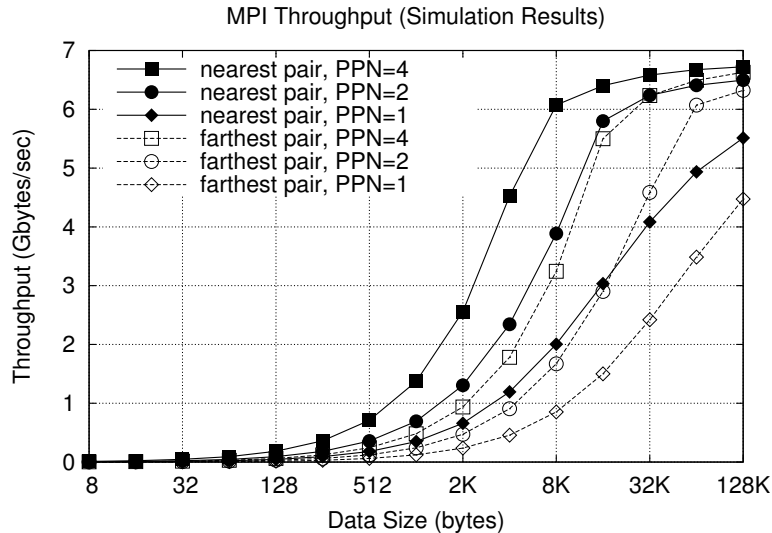


Figure 3.9: MPI throughput from simulation as a function of message size for 1, 2 and 4 MPI processes per node.

that this is indeed the cause of the lowered throughput of our MPI performance, we artificially reconfigured the link delay so that the end-to-end delay for MPI becomes 700 nanoseconds. The results are shown in Fig. 3.10 (dashed lines), which clearly indicates a much closer match of the simulated results with the empirical measurements.

3.3.3.2 5-D Torus Model Validation

Now we present a validation of our 5-D torus-based Blue Gene/Q interconnect model. We considered a real HPC system, IBM Sequoia supercomputer, that deploys Blue Gene/Q interconnect architecture. IBM Sequoia was built by IBM and is maintained by Lawrence Livermore National Laboratory. Sequoia consists of 96 racks containing 98,304 compute nodes connected via the 5-D torus topology of dimensions $16 \times 12 \times 16 \times 16 \times 2$ [CEH⁺12]. The bandwidth along the links is 2 GB/s [CEH⁺11]. The link delay is set to be 40 ns [CEH⁺12]. We measured the end-to-end latency between two end nodes for a Blue Gene/Q system (to compare

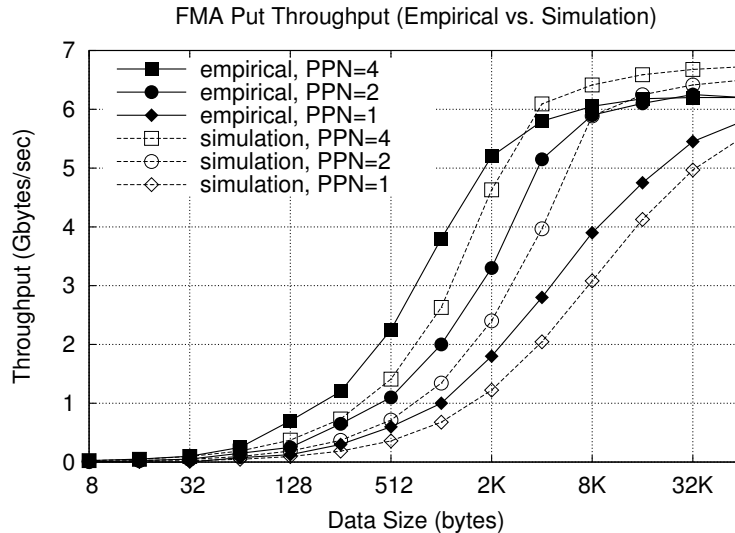
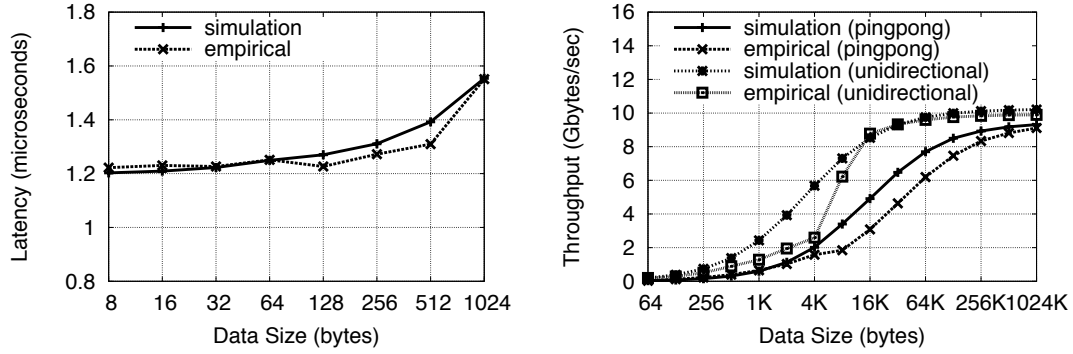


Figure 3.10: Gemini FMA put throughput (as reported in [ARK10]) versus simulated throughput as a function of transfer size for 1, 2, and 4 processes per node.

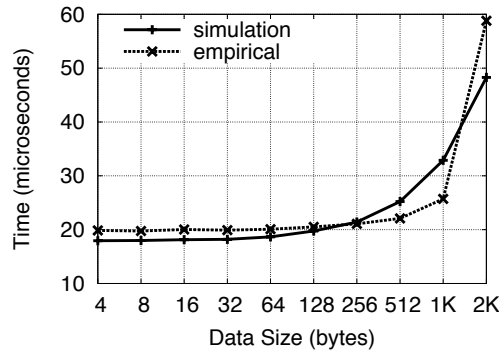
with the latency values reported in [CEH⁺11]). The end-to-end latency is measured by the propagation delay and the number of hops between the two end nodes. In this latency test, an MPI process sent an 8-byte data to all other MPI processes (mapped on different compute nodes). The measured delays are between 700 ns and 1300 ns. The results are consistent with the data reported in [CEH⁺11] (where the Blue Gene/Q system end-to-end latency is reported to be between 718 ns and 1264 ns).

3.3.3.3 Dragonfly Model Validation

Aries has 96 switches per group, 4 hosts per switch. Each switch has 48 network ports: 40 of which are used to connect the switches together and 8 are used to connect the switch to processors. We set the inter-group link bandwidth to be 4.7 GB/s per direction and intra-group link bandwidth per direction to be 5.25 GB/s [BJL⁺15]. The bandwidth of the interface connecting a host to its router



(a) Comparison of Aries latencies in terms of message size (b) Comparison of Aries MPI throughput in terms of message size



(c) Comparison of MPI Allreduce time

Figure 3.11: Aries validation.

is set to be 16 GB/s [AFKR12]. The link latency is set to be 100 ns (in a quiet network, measured router-to-router latency is reported to be 100 ns [AFKR12]).

For validation, we considered a large-scale interconnect at a recently-developed HPC system for validation of Aries. Trinity is being built at Los Alamos National Laboratory by U.S. Department of Energy (DOE). Trinity uses a Cray XC40 system that consists of 9436 nodes [WNC⁺15] connected via the Aries dragonfly network. We measured average end-to-end latency for considered interconnect system as a function of transfer size. Fig. 3.11(a) shows measured MPI latencies from our simulator. Fig. 3.11(a) also compares our latencies with the empirical result reported in [AFKR12]. As shown in the figure, our measured MPI latencies closely resembles the published results.

We measured the MPI throughput between two different nodes for different message sizes and compared it with the empirical values also published in [AFKR12]. The results are demonstrated in Fig. 3.11(b). We considered two different types of traffic for throughput comparison: pingpong and unidirectional. Fig. 3.11(b) shows that, in case of pingpong traffic, the simulation closely resembles the empirical results until 4K data size and after 4K data size, we can observe a slight shift. For unidirectional traffic, a good match is observed above 16K data size. Overall, the model has a good prediction of the throughput in general. There are many factors that may affect the throughput, including buffer management at both the sender and receiver, and also system overheads that may not be included in our model.

We also conducted a latency measurement for MPI collective operations. For this experiment, we used the configuration of interconnect deployed at supercomputer Darter. Darter was built by National Institute of Computational Sciences (NICS) [FBCM14]. It is a Cray XC30 system that consists of 748 compute nodes and 10 service nodes in two groups. The nodes are connected using dragonfly network topology with Cray Aries interconnect. Fig. 3.11(c) shows the result of measured time for `MPI_Allreduce`, as we vary the message size. As expected, the latency of collective operation increases with increase in message size. When message size becomes much higher, delay increases due to congestion in the network. We compared our results for Allreduce time with the one reported at [BCY13], for a similar configuration (i.e., the Darter supercomputer). The compared results show close correspondence to each other.

3.3.3.4 Fat-Tree Model Validation

Now we present validation of our m -port n -tree fat-tree-based Infiniband interconnect system. As an example of HPC system deploying fat-tree interconnect, we

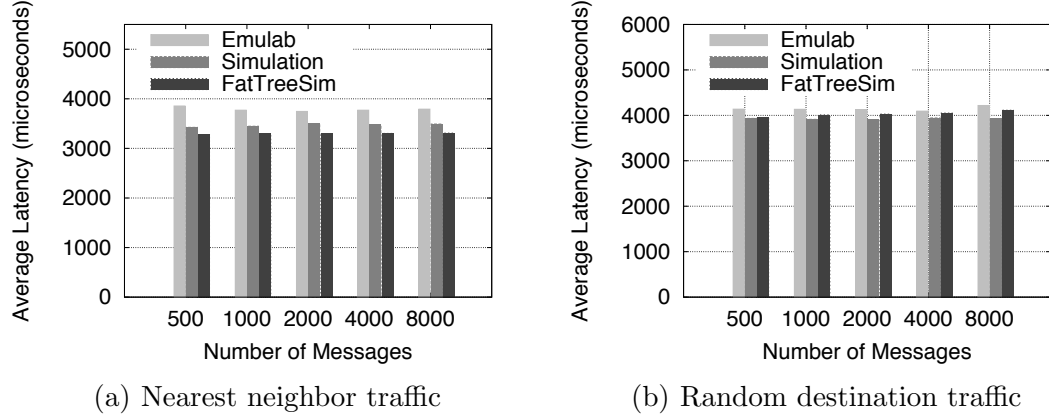


Figure 3.12: Comparison with FatTreeSim and Emulab.

consider Stampede supercomputer specifications. Stampede is built by National Science Foundation (NSF) at the Texas Advanced Computing Center (TACC), U.S. Stampede consists of 6,400 nodes connected via fat-tree-based Infiniband FDR network [Tex17]. The FDR Infiniband consists of 56 Gb/s Mellanox switches [Tex17] and we use this configuration in both uplink and downlink bandwidth of our fat-tree model. We assign $0.7 \mu s$ as uplink and downlink latency for our considered fat-tree interconnect.

We compare our model with the output reported by a recently-proposed fat-tree simulator, FatTreeSim [LHSJ15]. We also compare with Emulab (a network testbed) output reported at [LHSJ15] for similar system setup. We consider a 4-port 3-tree fat-tree interconnect with total 16 processing nodes and 20 switches. We set the message size to 1,024 bytes. We consider two traffic patterns for comparison: nearest neighbor and random destination. We vary the number of messages from 500 to 8,000 for conducting similar comparison to data reported in [LHSJ15]. Figs. 3.12(a) and 3.12(b) show comparison with Emulab and FatTreeSim for the nearest neighbor and random destination traffic, respectively. As evident from both figures, average latency calculated for each message in our model demonstrates close correspondence

to the result from both Emulab and FatTreeSim and for both types of traffic with varying number of messages.

3.4 Experiments

In this section, we first present a trace-based simulation study to demonstrate the capability of our model for incorporating realistic applications. Next, we describe a performance study of a parallel application (computational physics) using one of our interconnection network models and show that our model can accurately predict the strong-scaling trends of the application. Finally, we present a study on the parallel performance of the interconnect model.

3.4.1 Trace-Driven MPI Simulation

Now we present a trace-based simulation study to demonstrate the capability of our interconnect model of incorporating realistic application behaviors, and further validate our model by comparing the communication cost predicted by our model against the actual performance of running the scientific applications on target HPC platforms.

In this study, we use real application communication traces provided by the National Energy Research Scientific Computing Center (NERSC). These traces are used for characterizing the demand of various DOE (US Department of Energy) mini-apps run at various large-scale computing facilities [NER15a]. The traces contain single-node execution profiles of the mini-apps, which include the execution time, the execution speed (the number of instructions per second), the workload (the number of floating-point operations), as well as other cache/memory performance metrics, such as cache miss ratios at different levels. The traces also provide

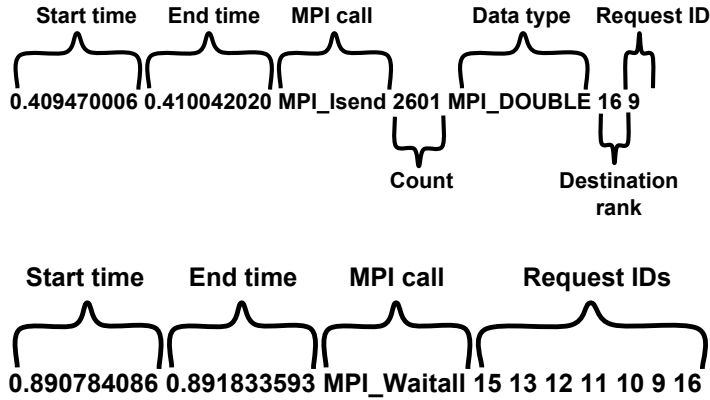


Figure 3.13: Format of MPI calls in the processed trace file (there is one trace file for each MPI rank).

parallel speedup performance and MPI communication operations. The latter is of particular interest in our study.

The DOE mini-apps in the trace collection were run at DOE’s three co-design centers, each covering two main applications. ExMatEx (Extreme Materials at Extreme Scale) [Law11] contains the traces of the Neutron Transport Evaluation and Test Suite (HILO) and the Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH). CESAR (Center for Exascale Simulation of Advanced Reactors) [Off13] contains the traces for the MOC emulator and Nekbone, which solves a poison equation using conjugate gradient iteration with no preconditioner on a block or linear geometry. ExaCT (Exascale Simulation of Combustion in Turbulence) [ASC17] contains traces for a multigrid solver and CNS, a stencil-based algorithm for computing the Compressible Navier-Stokes equations.

The MPI traces was performed on Hopper (described earlier) using the open-source DUMPI toolkit [Lab15] for different number of cores (e.g., 64, 256, and 1024 cores). For each run of the given application, there are a set of trace files, one for each MPI rank. The original trace files are in a binary format. We converted the binary files to text files, using the SST DUMPI toolkit [Lab15] and then processed the files

to assemble the necessary information of each MPI call in order, which includes the measured start and end time of the MPI call, and the specific parameters associated with the call, such as the source or destination rank, data size, etc. As an example, Fig. 3.13 shows two entries of a processed trace file, one for `MPI_Isend` and the other for `MPI_Waitall`.

`MPI_Isend` is a non-blocking send; the function is expected to return immediately with a request handle, which the user can later use to query or wait for the completion of the corresponding non-blocking MPI operation. An entry associated with the `MPI_Isend` call includes the start time and the end time of the MPI call. The count indicates the number of data elements to be sent. Using the count and the data type, one can easily determine the true size of the MPI message. In the example, 2,601 elements of the `MPI_DOUBLE` type (8 bytes each) would give 20,808 bytes of data which is scheduled to be transferred for this MPI non-blocking call. The entry also provides the destination MPI rank and an ID to represent the request handle returned by the MPI call. `MPI_Waitall` waits for a list of MPI requests to complete. Accordingly the corresponding entry in the trace provides a list of the request IDs. The MPI function will not return until all corresponding non-blocking operations (which may include both `MPI_Isend` and `MPI_Irecv` calls) are completed.

To run the trace, we start the simulation with the same number of simulated MPI ranks. At each MPI rank, we read the corresponding processed trace file for the rank, one entry at a time. For each entry, we first advance the simulation clock to the exact start time of the MPI call shown in the trace, by having the simulation process to sleep for the exact amount time equal to the difference between the MPI start time and current simulation clock. We then call the same MPI routine in our model and measure the time it takes to complete the MPI call in simulation. We

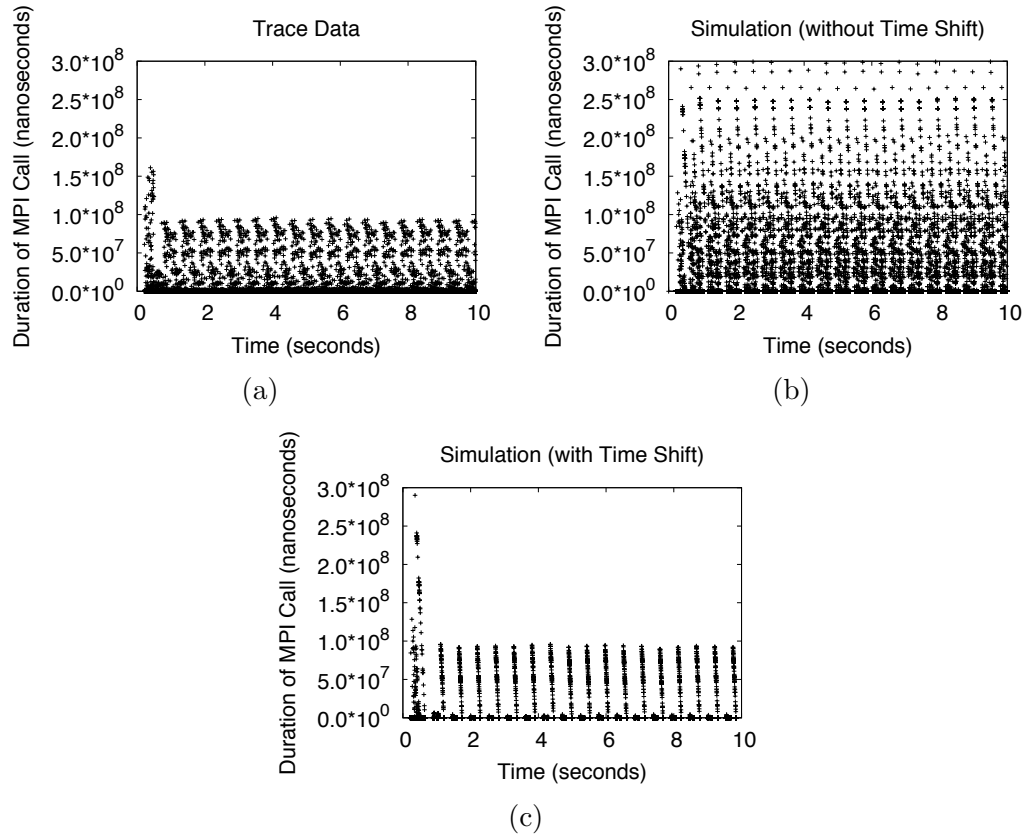


Figure 3.14: Comparing the duration of MPI calls between trace and simulation with and without time shift.

record the time and later compare it against the end time of the MPI call in the trace.

Fig. 3.14 shows the results of our trace-driven simulation for LULESH from Ex-MatEx running on 64 MPI processes. Our method can be generally applied to all other traces. LULESH is a mini-app that approximates a typical hydrodynamics model and solves Sedov blast wave problem in 3-D [Law15]. It is a widely-studied proxy application, which can efficiently run on various platforms and has been ported to a number of programming models (including MPI, OpenMPI, Chapel, and Charm++) [KBK⁺13]. The particular trace runs for approximately 55 seconds. There are a total of 123,336 calls to `MPI_Isend` and the same number for

`MPI_Irecv` and `MPI_Wait`. There are 12,864 calls to `MPI_Waitall`, 6,336 calls to `MPI_Allreduce`, 64 calls each to `MPI_Barrier` and `MPI_Reduce`.

Fig. 3.14(a) shows the duration of MPI calls observed from the trace (by subtracting the start time from the end time). For easy exposition, we show only the first 10 seconds of the experiment (later time exhibits similar behavior). Fig. 3.14(b) shows the trace-driven simulation result. At first glance, the simulation shows very similar pattern, yet the duration of the MPI calls spreads as much as three times of the empirical results. A closer inspection shows that the simulation clock sometimes may go beyond the start time of the MPI calls in trace. This is possible since the simulated process may take longer time to complete the previous MPI operation.

To eliminate this bias for comparing the duration of the MPI calls between the simulation and the empirical measurements, we introduce time shift for the trace. When the simulation process detects that its clock goes beyond the time of the trace, we shift the start time of all subsequent MPI calls in the trace by the difference so that the delay of the previous MPI calls in the simulation will not affect the subsequent calculations of the duration of the MPI calls.

Fig. 3.14(c) shows the result of simulation with this time shift. We observe that the duration of the MPI calls becomes much lower. The outstanding spikes (up to around 100 milliseconds) are from `MPI_Waitall`. The staggering pattern seems to be related to the skew in the wall-clock time of the participating compute nodes in the original trace. This would explain the spread of the durations of the MPI calls observed in the original trace (in the top plot).

Also, we perform a brief comparison study of different topology performance based on the application communication traces. We run the trace for each of the interconnect models we have presented (i.e., Aries, Infiniband, Gemini, Blue Gene/Q). We use configurations of Trinity and Stampede interconnects to represent architec-

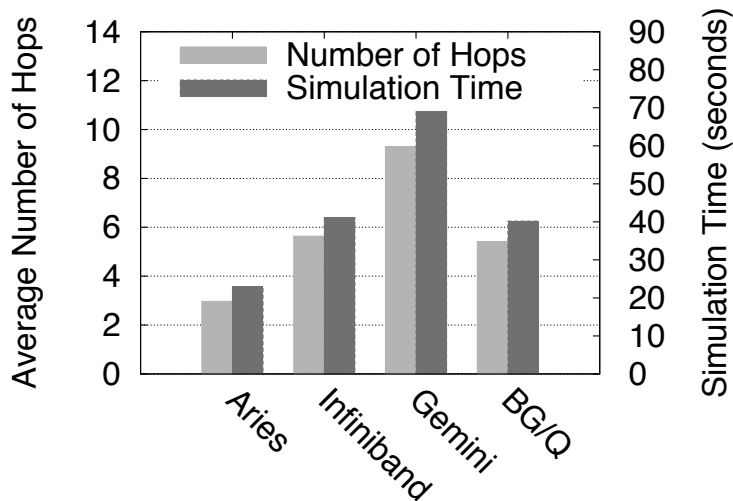


Figure 3.15: Comparison of different architectures for trace-based run.

tures of Aries and Infiniband, respectively. We use interconnect configuration of Hopper (a supercomputer built by NERSC [NER15b]) in our Gemini interconnect validation. Hopper contains 6,384 nodes connected via the Gemini interconnect at $17 \times 8 \times 24$. For Blue Gene/Q architecture, we use the interconnect configuration of Mira (a supercomputer at Argonne National Laboratory, which uses 5-D torus-based Blue Gene/Q at dimensions: $8 \times 12 \times 16 \times 16 \times 2$) [ZYL⁺15].

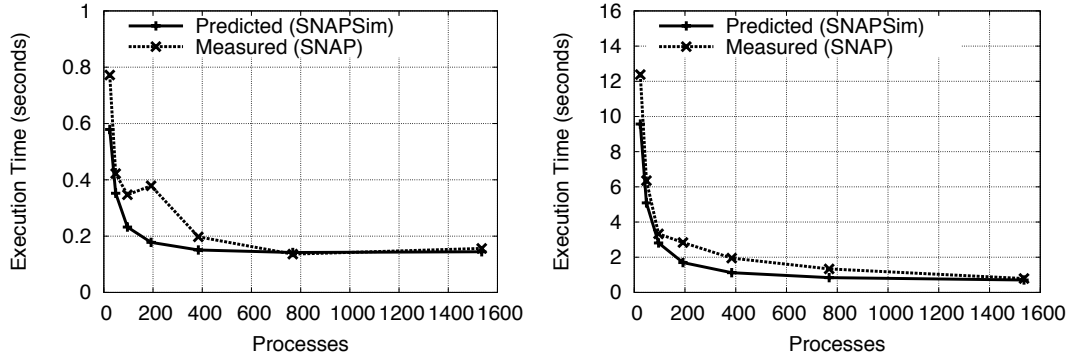
For this experiment, we collected the traces for DOE mini-app Big FFT (which solves 3-D FFT problem) on 100 processes from [NER15a]. There are a total of 400 calls to `MPI_Alltoallv` and 500 calls to `MPI_Barrier`. The trace also contains a number of group and sub-communication MPI calls: 4000 calls to `MPI_Group_free`, 2000 calls to `MPI_Group_incl` and 2000 calls each to `MPI_Comm_create` and `MPI_Comm_group`. Running such trace-based simulation serves two purposes: 1) It demonstrates that our designed topology models are capable of supporting real communication applications, and 2) It provides a comparison among different interconnect topologies with respect to their effect on parallel applications.

Fig. 3.15 shows average number of hops traversed by each of the different topologies considered in this work. As can be seen in the figure, Aries incurs the minimum number of hops in average, while Gemini has the maximum number of hops. Fig. 3.15 also shows that the simulation time differs in accordance with the number of hops: Aries incurs minimum simulation time, while Gemini takes the most simulation time, among all four types of topologies.

3.4.2 SNAP Performance Study

The SN Application Proxy (SNAP) [ZB13] is a “mini-app” based on the production code PARTISN [ABD⁺15] at Los Alamos National Laboratory (LANL). PARTISN is a code for solving the radiation transport equation for neutron and gamma transport. The resulting solution is the distribution of these sub-atomic particles in space, direction of travel, particle speed, and time. These dimensions of space, direction, speed, and time form a phase space that is discretized to formulate a linear system of equations. Solving this system of equations in parallel in the most efficient manner may depend on the architecture of the supercomputer employed. Simulation capabilities provided by the PPT will allow faster exploration of the optimizations and variations necessary when considering different computing systems.

A group of researchers at Los Alamos National Laboratory implemented an application model for the parallel wavefront solution technique of SNAP, called SNAP-Sim. SNAPSIm uses similar input variables as described above to parameterize the problem to describe the size of the problem and the size of the individual tasks. Although SNAP requires iterations to formulate a solution, SNAPSIm uses a fixed number of iterations, which is a sufficient abstraction for modeling the cost of instructions, data flow, and communications for an actual SNAP simulation. The sum of all work is broken up into work chunks, where each chunk represents the



(a) Study #1: smaller problem demonstrates scaling limitation (b) Study #2: larger problem for continued scaling

Figure 3.16: SNAPSim vs. SNAP Edison strong scaling.

solution for some chunk of spatial cells, all directions captured by a single octant of a unit sphere, and all particles binned into a single energy group. The nature of the problem permits that some work chunks be performed concurrently, and other chunks wait for these upstream tasks to be completed before progressing. The model captures this scheduling and estimates the compute time associated with a single chunk for different architectures. The time is computed per a hardware simulator that uses machine-specific details to estimate the computation time.

To test the SNAPSim model using the MPI and interconnect simulators provided by the PPT, we use the PPT hardware and interconnect model of NERSC’s Edison supercomputer. Edison is a Cray XC30 system that uses the Aries interconnect with a dragonfly topology. Each node in Edison is composed of two sockets, each with 12 Intel Ivy Bridge cores and 32 GB of main memory.

The first problem uses a $32 \times 32 \times 48$ spatial mesh, 192 angles, and 8 groups. Chunk sizes for spatial parallelism contain 8 cells in the x -dimension and a number that ranges from four cells to one cell in both the y - and z -dimensions as the number of processes is increased. We always start from 24 cores or one compute node to focus on the effects of off-node communication on scaling. Fig. 3.16(a) shows

that this problem has a low computational load at high core counts. The small computation-to-communication ratio leads to diminishing returns with increasing cores and perhaps even worse performance. While not in exact agreement, the simulator is capturing the trend from measured SNAP simulations quite well.

The second problem is larger with a $64 \times 32 \times 48$ spatial mesh, 384 angles, and 42 energy groups. Each chunk has 16 cells in the x -dimension and a number of cells in the other dimensions that varies with increasing processes. Fig. 3.16(b) shows how this problem maintains a much more consistent scaling trend due to the larger amount of computation between communications. And importantly the PPT model again predicts the measured trend well.

The results show the PPT can accurately predict strong scaling trends for SNAP on modern hardware with a well-understood interconnect. Succeeding in this validation exercise permits future deployment of SNAPSIm and the PPT for optimizing performance according the strong scaling properties without requiring a system allocation and extensive testing.

3.4.3 Parallel Performance

To assess the parallel performance of our integrated model, we conducted a set of experiments on a 1,500-node compute cluster located at Los Alamos National Laboratory. Each compute node in the cluster is equipped with a 12-core Opteron 6176 12C 2.3GHz CPU. The compute nodes are connected by an Infiniband QDR interconnect.

To obtain strong-scaling results, we simulated 156,672 MPI processes running on the Hopper. That is, there is one MPI process running at each core of the target supercomputer platform. For the experiment, the MPI processes perform a collective operation, using `MPI_Allreduce`, with different data size (1K or 4K bytes).

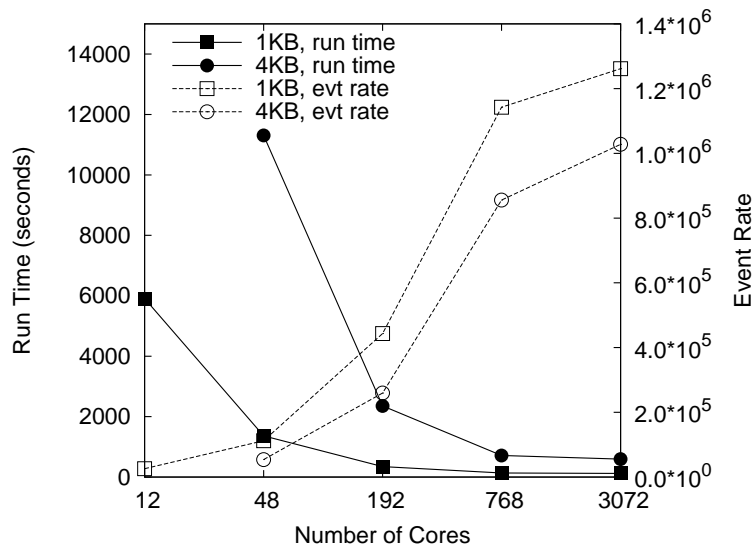


Figure 3.17: Observed run time and event rate for running Simian with an 156K-rank MPI model on a parallel compute cluster.

Fig. 3.17 shows the performance results. We ran the model varying the number of compute nodes, from 1 (12 cores) to 256 nodes (that’s 3,072 cores). For data size of 4KB, we ran the model with at least 48 cores to save compute time. The results demonstrate decent parallel performance of the simulator as we see the run time steadily decreases as we increase parallelism. However, the cost of using Simian’s Python implementation is also obvious. The aggregate event rate is low, even for 3,072 cores. For this experiment, we did not use Python just-in-time (JIT) compilation, which is expected to significantly improve the performance. We are in the process of translating our model to Lua, for which Simian has demonstrated superior performance. Using JIT and with sufficient event granularity, Simian has been shown to achieve as much as three times the event rate of an optimized C++ parallel simulator [SEL15].

3.5 Summary

In this part of the dissertation, we present integrated HPC interconnect models for performance prediction of HPC applications. Performance prediction for large-scale scientific applications require an accurate representation of the communication cost between an extremely large number of compute nodes. Our interconnect models are fully integrated with an MPI implementation that includes all common point-to-point communication functions and collective operations with packet-level accuracy. We present interconnect models for three widely-used interconnect topologies and corresponding interconnect architectures: torus (Cray’s Gemini and IBM’s Blue Gene/Q), dragonfly (Cray’s Aries) and fat-tree (Infiniband). We conducted extensive validation study of our integrated model, including a trace-driven simulation of real-life scientific application communication patterns. We also performed performance study of a computational physics-based parallel application using our interconnect model. All the results show that our interconnect models provide reasonably good accuracy for predicting the network behavior, while at the same time allowing for good parallel scaling performance.

CHAPTER 4

EMERGENCY DEMAND RESPONSE FOR HPC SYSTEMS

Demand response refers to reducing energy consumption of participating systems in response to transient surge in energy demand or decrease in energy supply from the power grid, possibly resulted from an emergency incident. Demand response is particularly important for maintaining power grid transmission stability, as well as achieving overall energy saving. HPC systems can be considered as ideal participants for demand-response programs, due to their massive energy demand. In this part of the dissertation, we explore the opportunity of HPC emergency demand response by proposing new HPC job scheduling and resource provisioning schemes.

4.1 Background

Demand response aims at energy reduction during peak electricity periods or other emergency events, and as such provides financial incentives to its participants. Various demand response programs are offered by energy service providers to encourage energy reduction from participants. Among these programs, emergency demand response is most widely adopted, taking up 87% of all the demand response capabilities across the U.S [Man14]. When supply shortage situations or emergency conditions occur (e.g., extremely cold/hot weather, natural disasters), energy consumers reduce the energy consumption to requested levels and collectively prevent the power grid from getting into blackouts, potentially saving billion of dollars' loss. Thus, many demand response resources, e.g., office buildings and residential customers, are emerging and sought to participate in emergency demand response. Many electricity markets (e.g., PJM, NYISO, and ISO-NE) serving major states in the United States contribute to power grid stability through demand response participation [Ene14]. Prominent companies (such as Apple [FP16] and Equinix [Mis15]) are participating

in demand response based on electricity price. The literature includes a large body of research and field studies of demand response for various sectors, such as data centers and smart buildings (e.g., [WLLMR14, BCPC16, GGMP12]).

HPC systems are generally large infrastructures containing thousands of nodes with a low latency interconnect and distributed file system. Scientific applications with high computation and communication requirements are generally executed on HPC systems. HPC systems can consume an enormous amount of energy during their operation. In the U.S., the Department of Energy has set a limit in the total power consumption of an upcoming exascale system to be within 20 MWs. It is projected nevertheless that future HPC systems in many other countries can easily exceed this amount; some systems may even consume hundreds of megawatts of electricity [GR17]. Apparently, the energy cost is a major component of the overall cost of operation. Any reduction in the electricity bill can be a significant benefit for HPC facilities. Furthermore, the energy consumption of HPC systems can fluctuate drastically due to workload diversity, temperature fluctuation, and dynamic power saving technologies such as clock gating and power gating. Being able to predict and control the massive energy demand can be important for maintaining stability of the energy provider. We argue that by participating in the demand response program and earning rewards from such participation, HPC systems can both reduce the overall cost of operation and contribute to the power system stability. However, to enable HPC systems' demand response participation, the potential loss of performance must be weighed against the possible gain in power system stability and energy reduction.

4.2 Related Work

In this section, we discuss related work in performance and power prediction models, dynamic voltage and frequency scaling (DVFS) methods for energy saving, HPC job scheduling and resource provisioning strategies, demand response techniques for data centers, and power allocation methods.

Many power and performance prediction models have been proposed in the literature. For example, Singh, Bhadauria, and McKee [SBM09] proposed an analytical model for real-time prediction of processor and system power consumption. Performance monitoring counters are used to estimate the power consumption of the processors. Shoukourian et al. [SWAB14] proposed an analytical model for application-specific power and energy prediction. Based on historical energy usage by specific applications, the model predicts future power and energy usage; the model can also adapt the prediction accuracy with further execution of the applications. Shoukourian et al. extended the AEPCP model and proposed the Lightweight Adaptive Consumption Prediction (LACP) model to predict application execution time, power, energy for different number of nodes and CPU frequency [SWA⁺15]. The LACP model, however, does not predict application characteristics for different power capping values. Song, Barker, and Kerbyson [SBK13] proposed a unified quasi-analytical performance and power model. Their model combines application analysis with different computation and communication parameters obtained through micro-benchmarking to assess the impact of different applications on performance and energy efficiency of HPC systems. Olschanowsky et al. [ORS⁺10] proposed energy prediction for non-existent machines using existing application traces with performance counters. The method, however, requires instrumentation of the applications (at the basic block level) and predicts the total energy cost based on

the average energy cost at each operation. Wu et al. [WTCM16] also presented performance and power models based on hardware performance counters with CPU frequency. They used non-negative multivariate regression analysis to build models for application execution time, system power, CPU and memory power, using a small set of major performance counters and CPU frequency. They implemented a counter-ranking method to identify the model contribution of the measured counters. The model can be used to suggest modifications of applications to improve execution time and power consumption.

Different energy saving techniques have also been proposed for HPC systems. They include energy-efficient design for hardware components, including CPU, memory, and interconnection network. Saving energy intuitively implies a reduction in power consumption, runtime, or both. Wu et al. [WTCM16] classified the methods in this area into three categories: reduce time and power, reduce time but allow an increase in power, and reduce power while allowing an increase in time.

Energy-saving methods that exploit the dynamic voltage and frequency scaling (DVFS) capabilities on processors have been introduced (e.g., [GFFC07, LM06, RLDS⁺09, FPK⁺05]). CPU MISER [GFFC07] is an early effort that includes a runtime DVFS-based HPC power management scheme, which exploits different application phases using performance measurements (in cycles per instruction) during the execution of the applications. Freeh et al. [FPK⁺05] proposed an energy saving approach exploiting the energy-time tradeoff of MPI programs. Adagio [RLDS⁺09] performs runtime CPU frequency scaling and exploits the variations in the energy consumption during computation and communication phases of an application to reduce the overall energy consumption without impacting the overall execution time of the application. A more recent effort on DVFS by Bao et al. [BHC⁺16] automat-

ically selects the optimal frequency and core count at compile-time to achieve lower energy.

Job scheduling and resource provisioning methods have also been proposed for HPC systems to save energy. They assume bounded energy consumption of the systems (e.g., [SLGK14, PLS⁺15, ECLV12]). Yang et al. [YZW⁺13] proposed a job scheduling approach to exploit the variable electricity price and power consumption profile of the jobs. A day is divided into two parts based on the electricity price: on-peak and off-peak. Jobs are classified based on their power profiles (derived from past execution data). Low power-consuming jobs are executed preferably during the on-peak time periods, while high power-consuming jobs are executed preferably during the off-peak time periods. Two power-aware job scheduling solutions are proposed: a greedy policy and a 0-1 knapsack-based policy, where fairness is ensured through a window-based scheduling mechanism. Sarood et al. [SLGK14] proposed an online job scheduling and resource allocation approach to achieve power-efficiency in HPC systems. The resource management system leverages over-provisioning, power-capping and job malleability (i.e., dynamic shrinking and expanding the job size) to optimally allocate power and nodes. The optimization objective is to maximize throughput with power constraints by dynamically allocating resources to new and running jobs. They also proposed a prediction model that estimates an application power characteristics at any scale. The proposed dynamic scheduling method, however, may not be practical for HPC systems, where most jobs are not malleable and a scheduling policy needs to ensure fairness. Pointing out these limitations, Patki et al. [PLS⁺15] proposed a more practical resource management scheme with improved power utilization and application performance (in terms of average turnaround time). Cao, He, and Kondo [CHK16] recently proposed a job scheduling algorithm to limit the overall system power consumption within a given

power budget and improve the system throughput and resource utilization. While we also implement power-capped job scheduling in our proposed model, our goal is to improve power grid stability and energy saving for HPC demand-response participation. We schedule jobs and allocate resources to achieve optimal energy (only during demand response events), while theirs is to achieve power capping.

Workload scheduling and resource provisioning in data center with consideration of demand response scheme has been studied quite extensively. Load shifting in time, geographical load balancing, speed-scaling, server consolidation, power-capping are some of the approaches proposed in the literature for data center's demand response [WLLMR14]. However, these approaches are applicable for internet transaction-based data center workload, not for HPC applications. For data center workload, the service time is typically assumed to be uniform and delay intolerant (most jobs need to be serviced within the hour from which they are submitted). HPC jobs are much less uniform both in terms of service time and job size (requested resources in the number of processors). Also, most HPC jobs can tolerate some delays (given that some jobs may take hours or days to finish). As such, the data center demand-response models cannot be applied to HPC systems.

Various power-capping-aware methods have been proposed in the literature to optimize power, performance, and energy in HPC systems. Before the emergence of hardware-level power-capping mechanisms, dynamic voltage frequency scaling, idle cycle injection, and clock cycle modulation were popular approaches to limit the power consumption of processors. Hardware-level power capping is the most lucrative choice, since it gives the highest opportunity to save energy [GAO14]. RAPL, an implementation of hardware-level power capping, was first introduced in Sandy Bridge processors [RADS⁺12]. Patki et al. [PLR⁺13] performed an extensive study of application performance for an entire cluster while limiting the power usage at the

node level. An optimal power allocation scheme was proposed with consideration of application parallel efficiency and memory intensity to achieve the best application performance. Recently, Liu et al. proposed FastCap [LCD⁺16], a system-wide power-capping approach based on both CPU and memory DVFS to achieve optimal system performance within a given budget for systems with a large number of cores.

4.3 Demand-Response Model Based on Frequency Scaling

In this section, we present our HPC demand response model based on processor frequency scaling. At first, we present performance and energy models based on frequency scaling in section 4.3.1, which we use for determining a job’s runtime and energy consumption in the proposed job scheduling and resource provisioning algorithm, which we describe next in section 4.3.2. The algorithm involves dynamically adjusting the processors’ frequency for all running jobs in order to achieve optimal energy conservation, which we describe in section 4.3.3. The algorithm possibly involves evicting running jobs during a demand-response period if the power consumption exceeds the set limit. We describe an optimal job eviction method in section 4.3.4. Section 4.3.5 presents a performance evaluation study.

4.3.1 Power and Performance Prediction Models

Consider the set of jobs to be executed on an HPC system is $\{1, 2, \dots, J\}$. For each job j , we denote the average power consumption running at CPU frequency f as $p(j, f)$, and the execution time as $t(j, f)$. There are quite a number of models existed for predicting a job’s average power consumption and execution time. Here, we use a rather simple regression-based model. We derive the relationship between CPU power and runtime with respect to frequency using linear regression. To do so,

we first observe the average power and runtime characteristics of each job running at different frequency values. We then determine a polynomial fitting function based on the observed data.

Similar approaches can be found in other studies (e.g., [ABB⁺14, AW14]). The purpose of this study is to assess the feasibility of having HPC centers to participate in the demand response programs, by proposing a job scheduling and resource provisioning algorithm that can improve power stability and energy conservation while maintaining good application runtime performance. Here we choose simple prediction models, which can be later improved for more general applications.

For determining the average power consumption, we use a similar model as proposed in [WTCM16]. The average power consumption of job j running on a processor at frequency f can be estimated using the following third-order polynomial function:

$$p(j, f) = a + b \cdot f + c \cdot f^2 + d \cdot f^3 \quad (4.1)$$

where a , b , c , and d are constants determined from empirical analysis of average power relation with different frequency values. Here, a represents the static power consumption while running the job.

In a similar approach, we can determine the execution time of job j at frequency f using the following function:

$$t(j, f) = \alpha + \beta \cdot f + \gamma \cdot f^2 \quad (4.2)$$

where α , β , and γ are regression coefficients determined from polynomial fitting function using empirical data.

We assume that a job j runs with the same CPU frequency f on all n_j processors. As such, the total energy consumption of the job can then be determined as follows:

$$e(j, f) = n_j \cdot p(j, f) \cdot t(j, f) \quad (4.3)$$

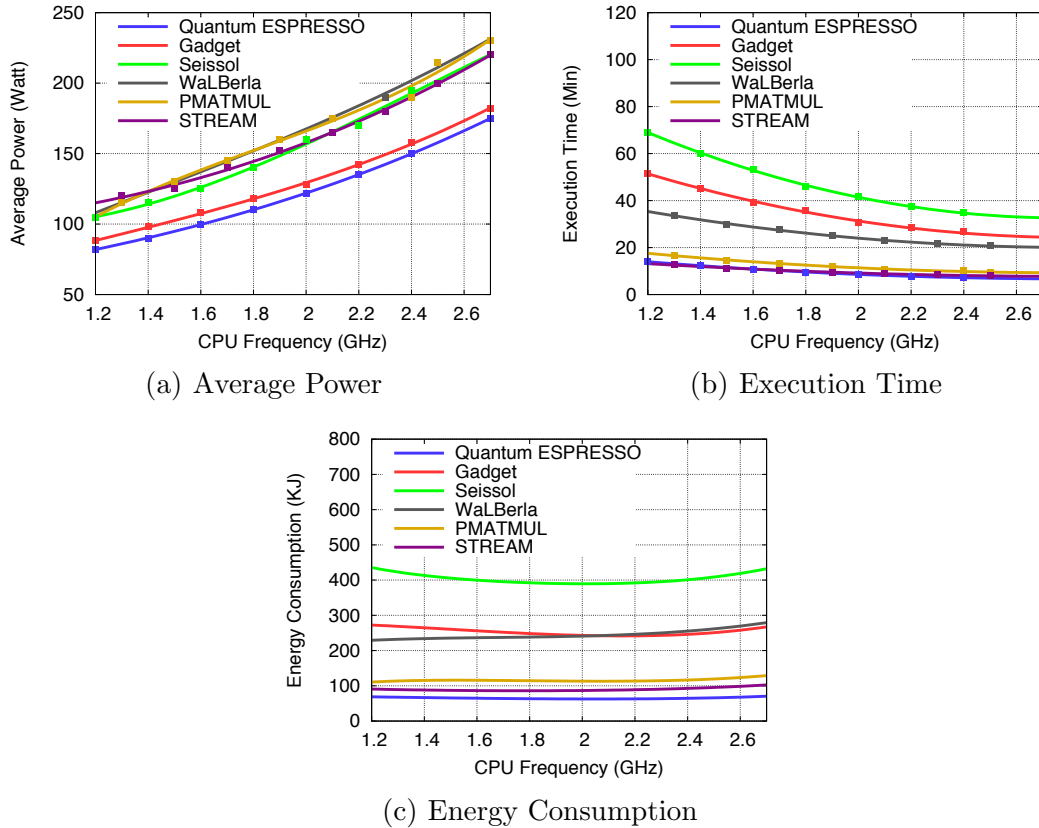


Figure 4.1: Result of the power and performance prediction models for six HPC applications.

To illustrate the power and performance prediction models, we collected frequencies and frequency-related power variations for six HPC applications from an existing study [ABB⁺14]. More specifically, the six applications include four scientific applications (including Quantum ESPRESSO [GGB⁺09], Gadget [Spr05], Seissol [KPC⁺08] and WaLBerla [FDK⁺11]) and two synthetic benchmarks (where PMATMUL is a parallel benchmark for dense matrix multiplication, and STREAM is a benchmark for measuring sustainable memory bandwidth [McC02]). Measurements were collected when running these applications with different CPU frequencies, ranging from 1.2 GHz to 2.7 GHz.

We use the regression models to derive the least square polynomial fitting functions representing the relationship of average power consumption and execution time with the scaling frequency for each application. Fig. 4.1 shows the result from the power and performance regression models for the six HPC applications, as well as their total energy consumption. Fig. 4.1(a) and Fig. 4.1(b) show the empirical data and fitted polynomial function for average power and execution time, respectively. Fig. 4.1(c) shows the total energy consumption for different frequencies, derived from the average power and execution time polynomial fitting function models.

In general, the average power consumption of the applications increases as we increase the CPU frequency. The execution time decreases as we increase the CPU frequency. The total energy for running the applications is the product of the average power and execution time, which may either increase, decrease, or have its minimum somewhere in the given frequency range, depending on the applications.

4.3.2 Job Scheduling and Resource Provisioning

We describe the proposed job scheduling and resource provisioning algorithm in this subsection. When a job is submitted, it is inserted in the waiting queue Q and the job scheduling algorithm is invoked. Here we use the first-come-first-serve (FCFS) policy, although other job scheduling policies can be applied as well.

The job scheduler keeps the list of running jobs R . Each job $j \in R$ runs on n_j processors as requested. The scheduler determines the frequency of the processors, f_j , that the job is run. The frequency can be changed dynamically during the job's execution depending on the current running jobs and the available power limit. (We assume all processors running the same job maintain the same frequency nevertheless.)

Algorithm 1 HPC Demand Response Job Scheduler

```
1: find the first eligible job  $j$  in  $Q$ 
2: if job  $j$  exists then
3:   dequeue job  $j$  from  $Q$ 
4:   allocate  $n_j$  processors to run job  $j$ 
5:    $R \leftarrow R \cup \{j\}$ 
6:   goto line 1
7: end if
8: determine optimal frequency  $\forall j \in R$  (section 4.3.3)
9: if no optimal solution exist then
10:  evict jobs to reduce power consumption (section 4.3.4)
11:  goto line 8
12: end if
13: reset processor frequency if changed  $\forall j \in R$ 
```

Let f_{min} and f_{max} denote the minimum and maximum frequency allowed by the HPC processor architecture. That is,

$$f_{min} \leq f_j \leq f_{max} \quad (4.4)$$

Let \hat{p} be the current power limit set by the energy service provider. The power cap \hat{p} can be set to a lower value during a demand response period and infinite otherwise. In any case, the scheduling algorithm needs to ensure that the average power consumption of all the running jobs, p_{run} , is bounded by the current power limit. That is,

$$p_{run} = \sum_{j \in R} p(j, f_j) \leq \hat{p} \quad (4.5)$$

The pseudo-code of the proposed HPC scheduler is shown in Alg. 1. When invoked, the scheduler first checks to see if there is an eligible job to run in the job waiting queue (line 1). We scan the waiting jobs from the head of the queue to the tail of the queue according to the FCFS policy. The job w is eligible to run, (a) if there are enough available processors, that is,

$$n_w + \sum_{j \in R} n_j \leq \hat{n} \quad (4.6)$$

where \hat{n} is the total number of processors in the system, and (b) if the average power consumption of all running jobs remains within the power limit (assuming we run job w with the minimum allowed frequency):

$$p(w, f_{min}) + p_{run} \leq \hat{p} \quad (4.7)$$

If such a job is found, we remove the job from the waiting queue (line 3) and allocate the processors to run the job (line 4). The new job is placed into R , which maintains the set of all currently running jobs (line 5). This process is then repeated until we find all eligible running jobs from the waiting queue.

Before the new jobs begin, we first need to determine the optimal frequency of the processors to run them (line 8). Also, it may be necessary to adjust the frequency of existing running jobs (not just the new arrivals) to achieve the optimal energy conservation. We discuss the details of calculating optimal frequencies in section 4.3.3.

The job scheduler is invoked when a new job is submitted or when a running job has finished execution. In the latter case, the completed job is simply removed from R and the scheduler is invoked so that other eligible jobs can be scheduled to run. Another possible case for invoking the scheduler is when the energy service provider changes the power limit \hat{p} of the HPC system. This can be the start of a demand response event, in which case a lower power limit is imposed, or at the end of a demand response event, when the power limit returns to normal (e.g., infinite or some higher values for hardware overprovisioned systems [PLR⁺16]). If the power limit is reduced for a demand response event, it is possible that no optimal solution can be found for frequency scaling of the existing running jobs. In that case, one or more running jobs must be terminated prematurely to preserve power (line 10). We discuss this step in more detail in section 4.3.4 on how to choose the victims

so that we can minimize the overall impact. Once the eviction is done, we need to calculate again the optimal frequency of the remaining running jobs.

In the last step (line 13), we change the frequency of the processors of the running jobs, as long as their newly calculated frequency is different from the previous settings. The job scheduler finishes the current invocation and will wait until it is invoked again in response to either a new job arrival, a job departure, or a demand response event.

4.3.2.1 Job Scheduler Simulator

We use simulation to study the effect of the proposed job scheduling and resource provisioning algorithm both on performance and energy. Plenty job scheduler simulators exist. For example, PYSS (Python Scheduler Simulator) is an open-source HPC workload scheduling simulator written in Python [PSL10]. The simulator was developed by the Experimental System Lab at the Hebrew University, and has been used to study various scheduling algorithms (e.g., [GGRT15, MV15, BGK⁺16, LW15]). CQSim is another event-based simulator to study the detailed queuing behavior of job schedulers using real system workload [Ill12]. The simulator was developed by Illinois Institute of Technology and has been used to evaluate fault-aware utility-based job scheduling [TLDB09], adaptive metric-aware job scheduling [TRLD12], and so on. Current HPC simulators provide only limited capabilities for studying job scheduling. For example, SST/Macro contains only limited support for running multiple jobs via trace replay [JAC⁺12]. CODES offers similar capabilities using trace replay to study multi-job workload of proxy applications and their impact on communication over different interconnection networks [JBW⁺16].

In general, job scheduling is relatively straightforward to simulate and validate. We developed our own simulator with trace-driven capabilities so that we can have

the flexibility to incorporate new scheduling functions, power-aware methods, as well as demand response models.

Our job scheduler simulator is developed based on Simian, which is an open-source, process-oriented, parallel discrete-event simulation engine [SEL15]. Simian has several unique design features that make it more attractive for us to build our scheduler simulator. First, Simian has a very simple application programming interface (API). The simulator adopts a minimalistic design with only a handful of core functions. The code base is around 500 lines at its core, which makes it easier to understand and debug the applications. Simian also supports process-oriented world view for easy model development. Second, Simian is developed using interpreted languages, including Python, LUA, and Javascript. Simian takes advantage of just-in-time (JIT) compilation and, for some models, has demonstrated capable of even outperforming simulators using compiled languages, such as C or C++. Simian is also a parallel discrete-event simulator, capable of running large-scale models on parallel platforms. Third, there has been a significant ongoing effort in developing models for HPC architectures and applications using Simian (e.g., [CESP15, CNEP16, AOL⁺16, ALEZ16]). Our job scheduler can take advantage of these models.

The overall design of the simulator is illustrated in Fig. 4.2. The job scheduler simulator consists of five major components: a job dispatcher, a job executioner, scheduling policies, application models, and a resource manager. The job dispatcher takes four different types of events: job arrival, job departure, job eviction (when an executed job is interrupted and removed in the middle of the run), and power demand change (when the power service provider of the HPC center changes the current power limit either at the start or the end of a demand response event). When a job is submitted, it enters the job waiting queue and invokes the job dispatcher.

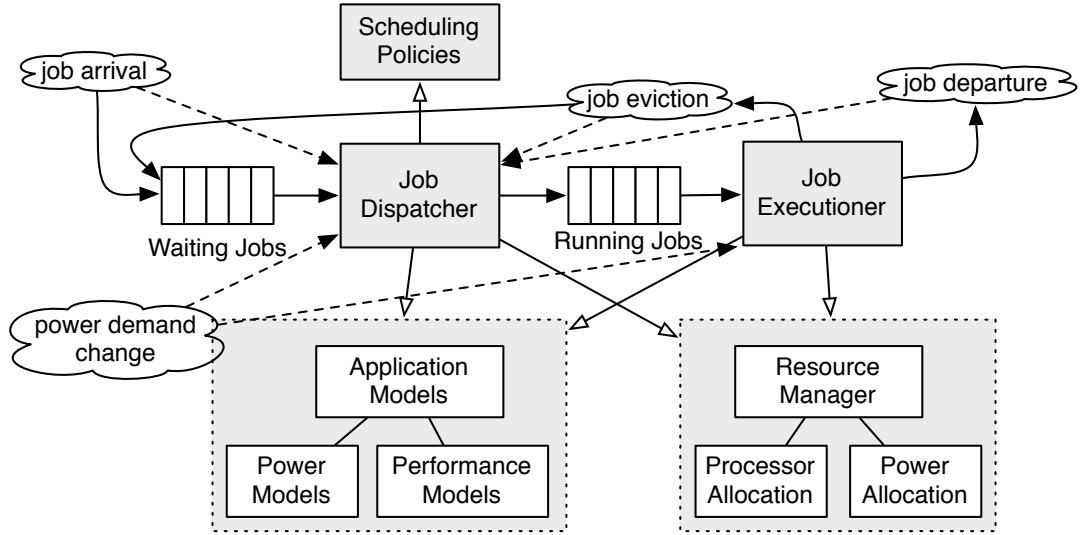


Figure 4.2: The overall design of our job scheduler simulator.

The job dispatcher determines whether the job is eligible to run according to the application models (that describe the job’s power and performance characteristics) and the current available resources from the resource manager. The job dispatcher processes the jobs from the waiting queue according to the scheduling policies. For this study, we only use FCFS, although other policies, such as backfilling [SKSS02], may be incorporated as well.

When a job is scheduled to run, the job dispatcher removes the job from the waiting queue and put it in the list of running jobs. The job executioner allocates the resources using the resource manager to represent the occupied processors (at the specified frequencies) with associated power consumption for running the job. The job executioner then simulates the job’s execution accordingly. For this study, it is sufficient to simulate using the job’s execution time and power consumption according to the estimates from the power and performance models. Detailed job execution can also be simulated for specific computation and communication demands, in case one needs to model the application’s runtime behavior. When a job

completes its execution, the job executioner removes the job from the list of running jobs, reclaims the resources occupied by the job, and then invokes the job dispatcher to select new eligible jobs to run.

Our simulator has also been augmented to handle demand response. We can schedule an event to indicate the power demand change, with a lower power limit upon the arrival of a demand response event, or an another when the power limit returns to level for normal operations. In the former case, the job executioner may evict jobs if the current power level is no longer sufficient to support all running jobs. In the latter case, the job scheduler may start new jobs to run.

4.3.2.2 Simulator Validation

We conducted experiments to validate the basic functions of our job scheduler simulator. We used the real system workload traces, obtained from the Parallel Workloads Archive [FTK14]. The workload traces contain runtime information collected at the San Diego Supercomputer Center (SDSC) during the time period from May 1998 through April 2000. The runtime information contains the job start time, the job run time, the requested number of processors, and the job wait time, etc. We use this dataset for validation by comparing the performance of our simulator with that of PYSS [PSL10], which has been previously validated against empirical results.

The results are shown in Fig. 4.3. The specific workload trace we use contains 5,000 jobs running on a system with 512 processors. The top plot shows the length of the job waiting queue as it fluctuates over time. The bottom plot shows the number of available processors in the system. In both cases, we can observe that our simulator generates results that match well with those from PYSS.

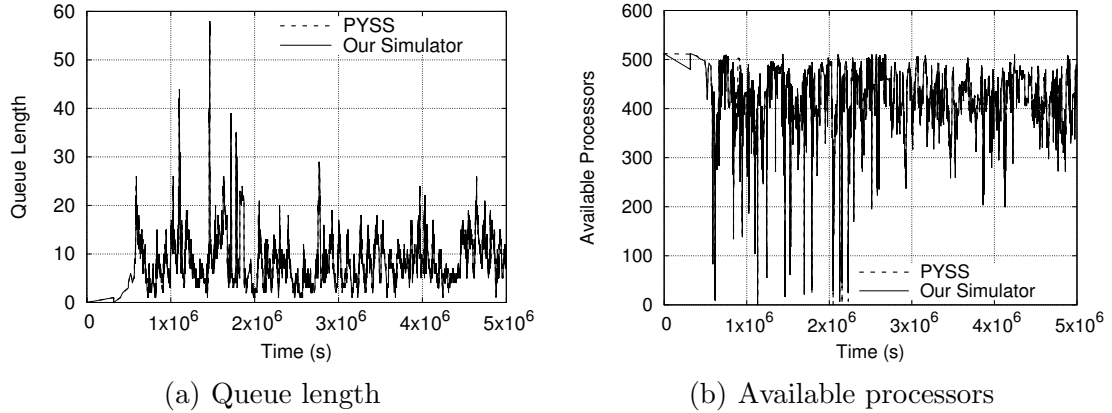


Figure 4.3: Comparing results from PYSS and our simulator.

4.3.3 Determining Optimal Frequency

This step is to calculate the frequency of the processors running the jobs. During the normal operating time, the power limit should be infinite (or set to be the peak power), in which case the jobs can run at the maximum allowed CPU frequency, i.e., f_{max} , to achieve the best performance. This is a conscious decision. HPC systems are designed for high performance. A willing participation of supercomputing centers in the demand response programs should not alter the main design purpose of these HPC systems. We want to minimize the overall impact of demand response, in this case, by recovering the potential performance loss by maximizing the application performance outside the demand response periods.

However, once a demand response event happens, we need to resort to the energy conservation mode. In this case, we want to select the proper CPU frequencies of all running jobs so that we can minimize the energy use while observing the reduced power limit set by the energy service provider. By reducing the energy demand, we can contribute to stabilizing the power grid which may encounter possible emergency situations. This frequency selection problem can be formulated as an optimization

problem, as follows:

$$\begin{aligned} & \text{Minimize: } \sum_{j \in R} e_R(j, f_j) \\ & \text{subject to constraints (4.4) and (4.5)} \end{aligned}$$

where $e_R(j, f_j)$ denotes the remaining energy expected to be consumed if running job j at frequency f_j . It can be calculated as follows:

$$e_R(j, f_j) = (1 - \alpha_j) \cdot n_j \cdot p(j, f_j) \cdot t(j, f_j) \quad (4.8)$$

where α_j is the percentage of job j that has been completed thus far. This quantity can be accumulated by the job scheduler upon each time the job is updated with a new frequency.

It is commonly believed that the energy and frequency observe the convexity property under certain conditions [DVMJC13], such as a job's average power consumption and execution time are monotonic functions of the frequency within a given range. This convexity property suggests the existence of an optimal frequency where energy consumption can be minimalized. We can therefore solve the optimization problem with the sequential least squares programming algorithm using the Han-Powell quasi-Newton method [Pow78]. The optimization problem solver returns the frequencies $f_1, f_2, \dots, f_{|R|}$ for all running jobs in $j \in R$. Note that in practice, processors can choose from a certain set of frequencies dictated by hardware. In this case, we can pick the closest allowed frequency that is no larger than the optimal frequency value.

4.3.4 Job Eviction

As mentioned earlier, with reduced power limit during a demand response event, it is possible that no optimal frequencies can be found for the existing running jobs,

in which case some jobs have to be terminated to preserve power. In this section, we provide an algorithm for choosing the jobs so that we can minimize the impact.

We represent the selection of job j by a binary variable, $x_j \in \{0, 1\}$. We formulate an optimization problem to determine the optimal subset of the running jobs such that the power bound constraint can be satisfied, with the objective of maximizing the energy that has already been spent by the running jobs. The idea is that we want to keep the jobs that have consumed more energy, because evicting them would mean this energy would be wasted as they need to rerun.

We formulate the optimization problem as follows:

$$\begin{aligned} & \text{Maximize: } \sum_{j \in R} (x_j \cdot e_X(j)) \\ & \text{subject to } \sum_{j \in R} (x_j \cdot p(j, f_{min})) \leq \hat{p} \end{aligned}$$

where $e_X(j)$ is the energy that has so far been spent running job j . On the one hand, the job scheduler can accumulate $e_X(j)$ using power measurement. On the other hand, we can adopt an easier alternative, by estimating the energy cost of a job using the job's completion percentage value α_j and the projected energy used under the current frequency f_j . That is,

$$e_X(j) \approx \alpha_j \cdot n_j \cdot p(j, f_j) \cdot t(j, f_j) \tag{4.9}$$

We can convert this optimization problem into a 0-1 knapsack problem and solve it directly. In this case, we treat \hat{p} as the knapsack capacity, $p(j, f_{min})$ as the weight associated with each job, and the spent energy $e_X(j)$ as the job's value.

4.3.5 Performance Evaluation

In this subsection, we present an elaborate trace-based simulation study to evaluate the effectiveness of the proposed HPC job scheduling and resource provisioning algorithm for emergency demand response.

4.3.5.1 Data Sets for Benchmarking

We use real-life workload trace to evaluate our design. More specifically, the trace was collected at the San Diego Supercomputer Center (SDSC SP2), which contains 5,000 jobs. This trace has been used widely in the literature, and referenced in a number of studies throughout the years to generate useful workloads (e.g., [DSRI13, KMY15]).

We use the performance and power data at different frequencies for four HPC applications (Quantum ESPRESSO [GBB⁺09], Gadget [Spr05], Seissol [KPC⁺08] and WaLBerla [FDK⁺11]), as outlined in Section 4.3.1. We use discrete frequency values for the processors, ranging from 1.2 GHz to 2.4 GHz at 0.2 GHz intervals and 2.7 GHz. The peak power of the processors was set to 220 W (determined from the power consumption of the four HPC applications when running at the maximum frequency). We target three HPC systems, which consist of 128, 256, and 512 processors, respectively. The peak power capacity for the 512-processor system can reach 112.64 KW.

To evaluate the performance of our job scheduling and resource provisioning algorithm for demand response, we compare it with two scheduling policies that do not consider demand response. *Performance-policy* is one of the CPU frequency scaling policies implemented in the Linux kernel [Arc17]. It always chooses the maximum frequency to ensure best application runtime performance [PS06, BHC⁺16]. *Powersave-policy* is the opposite to the previous one, also implemented in the Linux kernel [Arc17]. Under this policy, the processors are run instead with the minimum frequency, to minimize the power consumption for application execution.

In the following, we show the results from our simulation study. We first present the power capping capability of our demand-response algorithm. We then com-

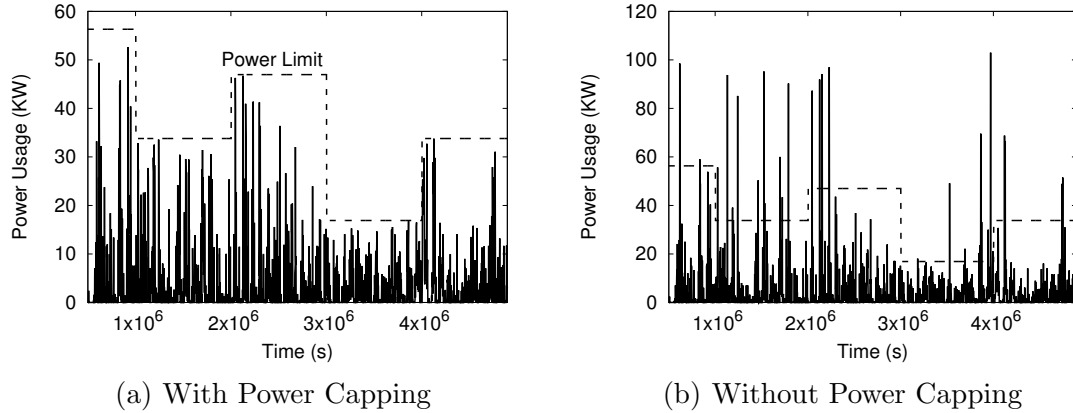


Figure 4.4: Power usage over time with and without power capping.

pare results from the demand response algorithm with those from the two demand-response-agnostic policies, both in terms of average job turnaround time and average energy consumption. Finally, we show the potential improvement in the power stability achieved by the demand response algorithm.

4.3.5.2 Power Capping

During a demand response event, the proposed job scheduling and resource provisioning algorithm switches to the energy conservation mode. In addition, the system’s power consumption is also kept to be within a given power limit in order to improve power stability. To show its effect of power capping, we designed an experiment by changing the power limit at different time intervals to demonstrate that our algorithm can schedule jobs according to the set power constraint at the time.

In this experiment, we arbitrarily set different power limit over time. Fig. 4.4 shows the result when we set the power limit at regular intervals to be 50%, 30%, 41.7%, 15%, and 30% of the system’s peak power. The figure shows the power usage of the system over time, with and without power capping. In the former case,

we used our demand response algorithm. In the latter case, we used the default performance-policy, which selects the maximum CPU frequency to run the jobs. The figure shows that our demand response algorithm can adapt to the changes in the power limit and schedule jobs accordingly under the power constraint.

4.3.5.3 Energy versus Performance

We conducted second set of experiments to study the effect of the proposed scheduling algorithm for demand response on the job’s energy consumption and execution time.

For this study, we vary the system size to be 128, 256, or 512 processors. We assume that a demand response event happens randomly during the system’s operation and lasts for 25% of the entire duration of operation. When the demand response event happens, we expect the power limit of the system to drop to 80% from the peak power. We measure the job turnaround time to be between the time when the job is submitted and the time when the job has completed its execution. We report the average job turnaround time and the average energy among all jobs. For the demand response algorithm, we also make a distinction of the average turnaround time and average energy between jobs that start inside or outside the demand response period. Finally, we compare the results of our demand response algorithm with those from using the performance-policy and the powersave-policy.

Fig. 4.5(a) shows the average job turnaround time decreases for all scheduling policies when we increase the system size (from 128 to 256 to 512 processors). This is expected: the average job waiting time would decrease due to less contentions when more resources are available. The scheduler running performance-policy has the smallest job turnaround time among the three scheduling algorithms since it always uses the maximum CPU frequency to achieve best application runtime performance.

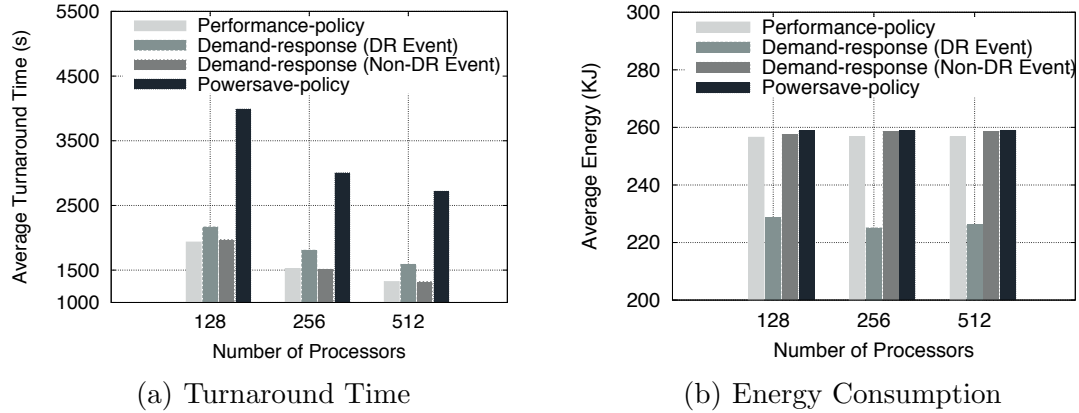


Figure 4.5: Comparing performance and energy for different scheduling policies and with different system size.

Our demand response algorithm operates in the same way as the performance-policy during the normal operation time (non-DR event), but performs slightly worse than the performance-policy during the demand response time period (DR Event). The processors may be set to run jobs with less than the maximum frequency in order to achieve the optimal energy conservation during the demand response period.

Fig. 4.5(b) shows the average energy consumption of the jobs. The per-job energy consumption is largely independent of the system size. The difference between performance-policy and powersave-policy is almost negligible, which is not unexpected. As shown previously in Fig. 4.1(c), the energy consumption of the four applications we choose for our study (e.g., Seissol) is at a similar level at both frequency extremes (at 1.2 GHz and 2.7 GHz). Considerable energy savings (around 15%) are achieved during the demand response period, when our algorithm finds the optimal CPU frequencies to achieve the best energy conservation for running the jobs.

We observed that both average job turnaround time and average job energy consumption depend on the demand response event ratio (i.e., the percentage of time in the system operation that demand response happens). In the next experiment,

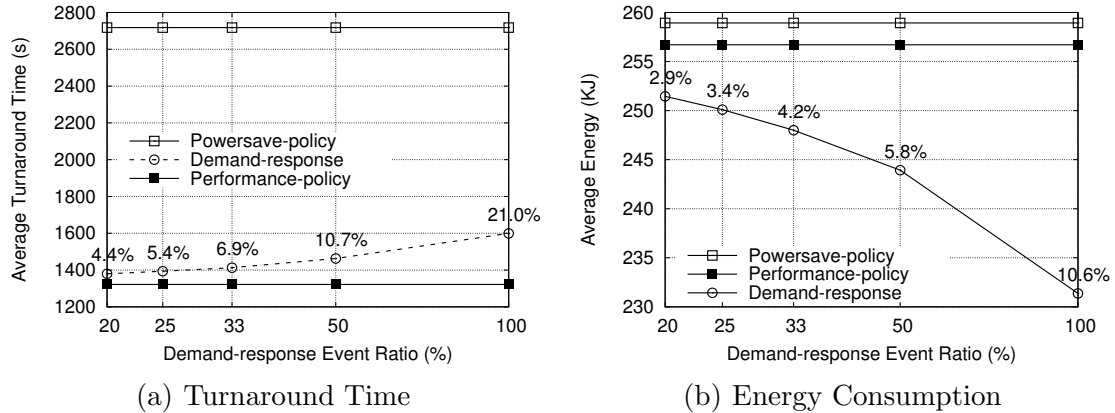


Figure 4.6: Impact on the demand response event ratio.

we fixed the system size to be 512 processors and varied the demand response event ratio from 20% to 100%. Fig. 4.6 shows the results. Fig. 4.6(a) shows that the average turnaround time increases only slightly for our scheduling algorithm when the demand response event lasts longer. Relative to the average job turnaround time achieved by performance-policy (which does not change with the demand response event ratio), we see that the demand response algorithm may introduce an increase between 4.4% and 21.0% in the average turnaround time.

Fig. 4.6(b) shows that the average job energy consumption decreases with the longer demand response event, since our scheduling algorithm would be more likely to operate in the energy-conservation mode. Relative to the average energy achieved by powersave-policy, we see that the demand response algorithm can achieve energy savings from 2.9% to 10.6%.

4.3.5.4 Power Stability

An important aspect of the demand response program is that it is expected to help stabilize the power system during the demand response periods when the power grid may encounter instability, either due to the sudden rise in the demand or because

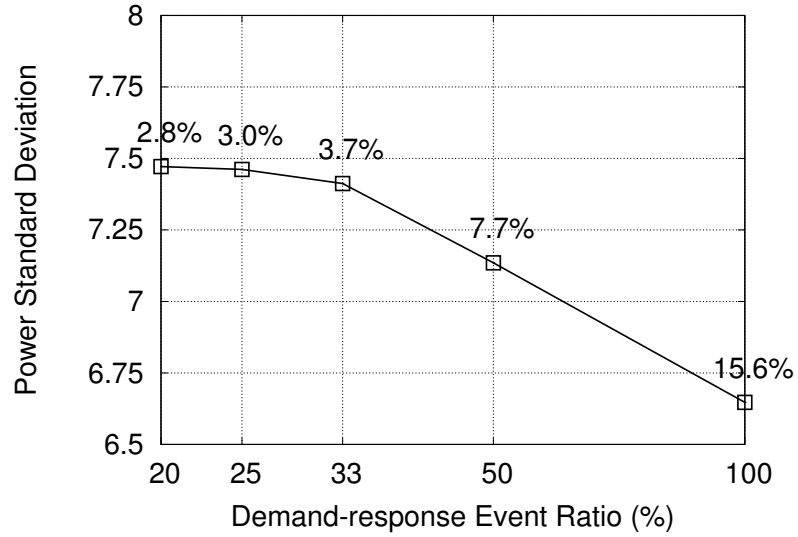


Figure 4.7: Power stability during the demand response periods.

of some emergency incidents. In this case, we would like to be able to minimize the fluctuations in the power demand of the HPC systems during the demand response events.

Fig. 4.7 shows the standard deviation of power usage of the system over time. In this experiment, we use the same simulation setup as in the previous experiment. We observe that the standard deviation of the power consumption decreases as the demand response event ratio increases. Relative to the standard deviation of the power usage under performance-policy, the demand response algorithm is shown to have achieved a reduction from 2.8% to 15.6%, and thus can contribute to the stability of the power system.

4.4 Demand-Response Model Based on Power Capping and Node Scaling

To cope with the variations in processing, modern processors are becoming adaptive, providing hardware-level power-capping capabilities that can opportunistically adjust their core frequency based on thermal and energy constraints (e.g., Intel’s Turbo Boost Technology). Power capping is the allocation of power to nodes mainly to achieve an overall HPC cluster power limit. Power-capping capability is becoming a standard feature for modern processors through various programming interfaces, such as Intel’s running average power limit (RAPL) [Int14], AMD’s advanced power management link (APML) [How09], and NVIDIA’s NVIDIA management library (NVML) [Nvi15]. New intelligent features have been introduced in processors to achieve energy efficiency through power capping at various locations in the system hierarchy. For example, Intel’s Intelligent Power Node Manager and Data Center Manager allow power capping at the node level and component level (e.g., processor, memory) to achieve energy efficiency with different granularity [SCC⁺12]. Power capping not only helps achieve power reduction in the node but also optimizes application performance within a power budget.

Power capping can be optimized to control performance and power of applications. With changes in the power limit, application execution time and average power consumption may change, which can impact the energy-to-solution of an application. Fig. 4.8 shows an example application behavior under different power-capping values. Fig. 4.8(a) presents our measured energy consumption and execution time for running an HPC application (Intel’s DGEMM application from the HPCC suite [LBD⁺06]) on a cluster. As can be observed in the figure, energy consumption has a convex characteristic that can be exploited to achieve optimal energy

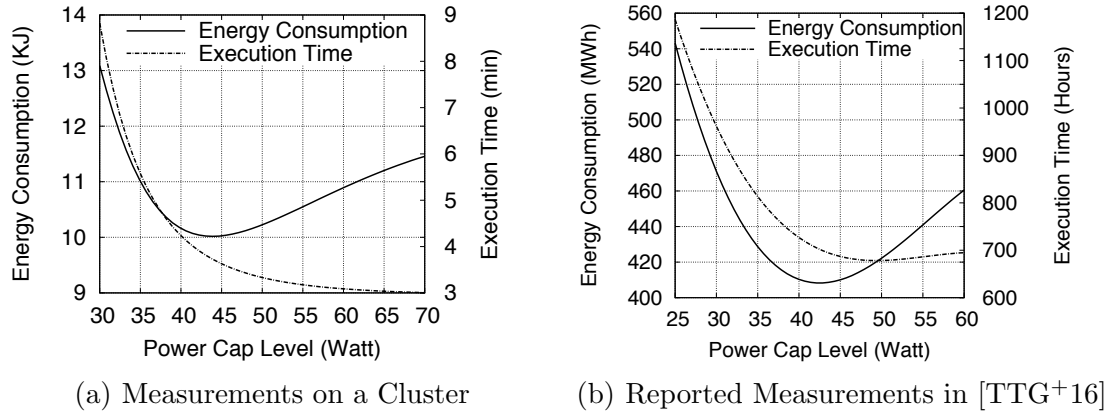


Figure 4.8: Impact of power capping on application characteristics.

consumption. We exploit the property to enable demand response participation in this study. Such convexity can also be observed in the literature. For example, Fig. 4.8(b) presents application characteristics reported in [TTG⁺16], where the scientific applications were collected from the Rodinia benchmark suite and the NPB benchmark suite. As evident from this figure, energy consumption has a convex relation with changes in the power-capping values.

In this section, we propose an emergency demand-response model for HPC systems based on power capping and node scaling.

4.4.1 Exploiting Power-Capping Property

In this subsection, we present the performance prediction models we consider and the optimization problem we implement for demand response participation of HPC systems. We leverage the power-capping property in each node to enable demand response participation.

4.4.1.1 Power and Performance Prediction Models

We now present the power-capping prediction model that we use to predict application behavior under different power-capping values. The average power consumption of job j running on a processor at power-capping level P can be estimated by the following polynomial function:

$$p(j, P) = a_j + b_j \cdot P + c_j \cdot P^2 + d_j \cdot P^3, \quad (4.10)$$

where a_j , b_j , c_j , and d_j are constants determined from empirical analysis of the average power relation with different power-capping values. In particular, a_j represents the static power consumption while running the application.

In a similar approach, we can determine the execution time of job j at power-capping level P using the following equation:

$$t(j, P) = \alpha_j \cdot e^{\beta_j \cdot P} + \gamma_j, \quad (4.11)$$

where α_j , β_j , and γ_j are regression coefficients determined from polynomial fitting function using empirical data.

To gain confidence in the proposed models, we present a validation study of the power and execution time prediction models based on real-life measurements of application running on a cluster.

We used a system monitoring/controlling tool, called `pycoolr` [Yos15], to sample per-CPU core temperatures and CPU/DRAM power consumption. The tool uses the Intel RAPL interface to take measurements and reports the results in the JavaScript Object Notation (json) format for later analysis. The tool can also be used to set the upper limit of CPU power consumption. In the validation study, we used `pycoolr` to set the processor's power limit and to characterize the behavior of HPC applications, particularly focusing on performance, temperature change, and actual power consumption with various power-capping values.

We selected various HPC applications from different sources, such that they can be representative of different application characteristics (e.g., compute-intensity vs. communication-intensity). In particular, we chose the applications from the CORAL benchmarks [Oak14], the NAS Parallel Benchmarks (NPB) [BBB⁺91], and the HPCC Suite [LBD⁺06].

The CORAL initiative is a collaboration among Lawrence Livermore National Laboratory, Oak Ridge National Laboratory, and Argonne National Laboratory and contains a number of HPC benchmarks, representing various DOE applications. We selected applications from the following divisions: scalable science benchmarks (applications expected to run at full scale on the CORAL systems), throughput benchmarks (applications representing large ensemble runs), data-centric benchmarks (applications representing data-intensive workloads, such as integer operations, instruction throughput, and indirect addressing), and skeleton benchmarks (proxy applications that investigate various platform characteristics including network performance, and multithreading overheads.) The following applications were chosen in particular from CORAL: (1) Nekbone, a compute-intensive application supporting various operations (such as MPI Allreduce, vector operations, and matrix-matrix multiplication); (2) LULESH, an application that models hydrodynamics for unstructured meshes and solves a simple Sedov blast problem; (3) Hash, an application that evaluates the performance of architecture integer operations; and (4) XSBench, an application that stresses system through memory capacity.

The NPB suite includes a small set of programs designed to help evaluate the performance of parallel applications. From this suite, we selected the CG application (class C), which uses a conjugate gradient algorithm for solving particular systems of linear equations. From the HPCC suite, we selected Intel’s DGEMM application for our study. DGEMM is a double-precision general dense-matrix multiply routine

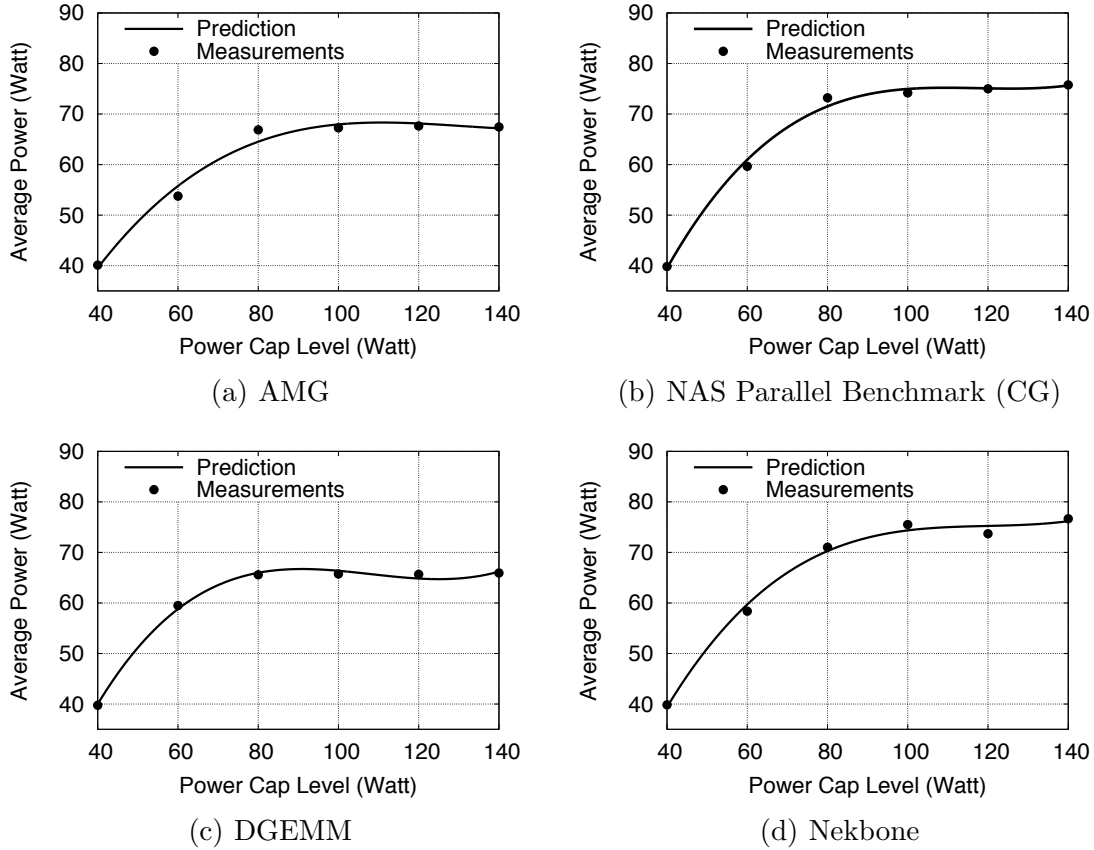


Figure 4.9: Power regression model for different applications.

in the Intel MKL library. The application is designed to measure the sustained, floating-point computational rate of a single node.

For our validation study, we varied the power-capping level from 40 W to 140 W at the increment of 20 W. We measured the average power usage and execution time of the applications using the `pycoolr` tool. Fig. 4.9 shows the average power consumption of running different applications: AMG, CG, DGEMM, and Nekbone, respectively. We plot both the measured experiment data and the fitted model data in the same figures. We observe that the prediction model matches with the application power usage generally well. Fig. 4.10 shows the execution time of different applications with different power-capping values. Similar to the power prediction model, the execution time prediction model is reasonably accurate, as evident from

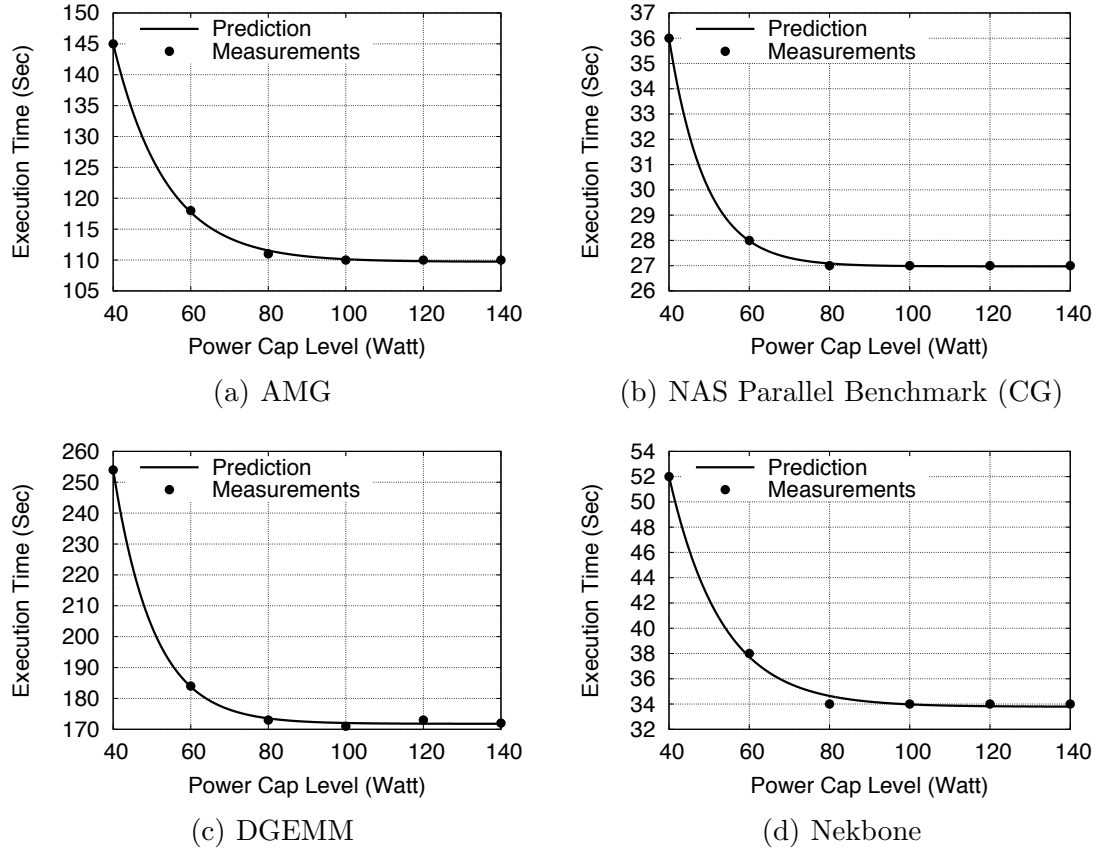


Figure 4.10: Runtime regression model for different applications.

the figure. We later use the measured data and prediction models to demonstrate the effectiveness of our proposed HPC system demand-response model.

4.4.1.2 Determining Optimal Power Cap

Next, we present an optimal power cap allocation algorithm for HPC system demand response participation.

During normal operating time, we set the power-capping value to the maximum limit. This is to ensure that the applications are run with maximum performance, such that demand response participation from HPC system does not impact the original target: to achieve high-performance capability for running the applications.

When a demand response event happens, we exploit the power-capping property and select an appropriate power-capping value to reduce energy consumption.

For simplicity, we assume that a job j runs with the same power cap P on all n_j processors, where n_j is the job size. The total energy consumption of the job can be determined as follows:

$$e(j, P) = n_j \cdot p(j, P) \cdot t(j, P). \quad (4.12)$$

Let P_{min} and P_{max} denote the minimum and maximum power cap allowed by the HPC processor architecture, respectively. We determine the power cap of the processors P_j for job j such that

$$P_{min} \leq P_j \leq P_{max}. \quad (4.13)$$

During the demand response period, we resort to the energy conservation mode by selecting the appropriate power cap values of all running jobs so that we can minimize the overall energy usage. The power cap limit allocation problem can be formulated as an optimization problem, as follows:

$$\text{Minimize: } \sum_{j \in R} e_R(j, P_j) \quad (4.14)$$

subject to constraint (4.4),

where $e_R(j, P_j)$ denotes the remaining energy expected to be consumed if running job j at power cap P_j , which can be calculated as follows:

$$e_R(j, P_j) = (1 - \alpha_j) \cdot n_j \cdot p(j, P_j) \cdot t(j, P_j), \quad (4.15)$$

where α_j is the percentage of job j that has been completed thus far. As outlined earlier, the energy consumption of applications at different power-capping values has been in general shown to be convex. We can therefore solve the optimization problem using standard optimization solver and determine optimal power-capping values for each job.

4.4.2 Exploiting Power-Capping and Node-Scaling Properties

In this subsection, we extend our demand-response model to incorporate node scaling for jobs that can vary the job size (i.e., with job malleability). In the preceding section, we considered only power capping. We do so when one cannot change the job size, that is, when the jobs are specified with a fixed number of nodes upon arrival. In this section, we relax this constraint for malleable jobs and exploit both node-scaling and power-capping capabilities for HPC system demand response participation.

4.4.2.1 Energy-to-Solution Prediction Model

We first present a prediction model that incorporates the effect of both node scaling and power capping for demand response. We consider various regression prediction models (e.g., linear interpolation, spline interpolation) for predicting the energy-to-solution (EtS) of the same HPC applications. We determine the type of predictor to be used in the demand-response model based on the root-mean-square-error value determined from the learned data and predicted data.

We implemented the following five predictor functions: (1) Linear, which captures the linear behavior of predicted function; (2) Spline, which captures the non-linear behavior of the prediction; (3) Linear + Spline, which is a combination that first captures the linear and then the non-linear behavior of the prediction; (4) Spline + Linear, which is a combination that first captures the non-linear behavior of the prediction and then the linear behavior of prediction; and (5) Linear + Spline + Linear, which is a combination that captures the nonlinear behavior in-between the linear behavior predictions at the beginning and at the end. For the combined cases, we determine the boundary value between different prediction regimes (i.e.,

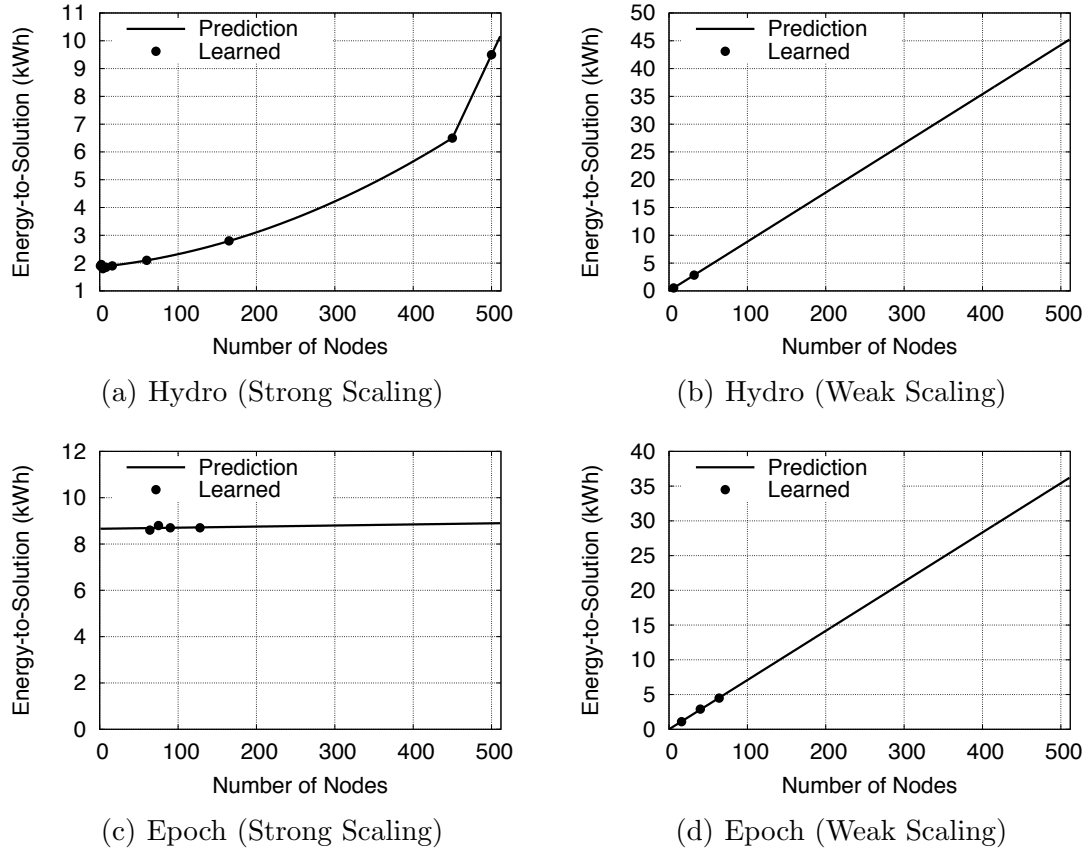


Figure 4.11: Node scaling model for different applications.

the points at which transition is made from linear to nonlinear or vice versa). After determining the appropriate prediction function and boundary values, we are able to predict the application behavior for unknown numbers of nodes. Then, we interpolate across known power-capping values to predict the application characteristics for unknown power-capping values.

To validate the EtS prediction model for different job sizes, we collected and used the values reported in [SWAB14] for two HPC applications: Hydro [LdVW⁺12] and EPOCH [A⁺14]. We first consider a strong-scaling scenario for the application Hydro and use the values from [SWAB14]. For this case, the data available were for node numbers 1, 2, 4, 8, 16, 165, 450, and 500. Fig. 4.11(a) shows the measured data points and predicted results for this scenario. We observe that the prediction model

correctly predicts a combination of nonlinear and linear predictor functions to be used for prediction, along with the appropriate boundary value (i.e., 450 nodes in this example). The prediction error is low. We make similar observations for Hydro under weak scaling, as shown in Fig. 4.11(b): the prediction error is reasonably low for this application.

Next, we consider a strong-scaling scenario for the application EPOCH. Fig. 4.11(c) presents the prediction for EPOCH application with strong scaling. We use the values from [SWAB14]. For this case, the data available were for node numbers 64, 75, 90, and 128. Fig. 4.11(d) presents the measured data points (for node numbers 16, 40, and 64) and predicted results for this scenario. As can be seen from the figures, the prediction model correctly predicts based on the measured data.

4.4.2.2 Determining Optimal Power Cap and Node Number

We now formulate an optimization problem to determine the optimal power cap and node number for the HPC system emergency demand response participation. The optimal number of nodes and optimal power cap determination problem during demand response periods can be formulated as follows:

$$\text{Minimize: } \sum_{j \in R} e_R(j, P_j, n_j) \quad (4.16)$$

subject to constraint (4.4),

where $e_R(j, P_j, n_j)$ denotes the remaining energy expected to be consumed if running job j at power cap P_j on n_j number of nodes, which can be calculated as follows:

$$e_R(j, P_j, n_j) = (1 - \alpha_j) \cdot n_j \cdot p(j, P_j) \cdot t(j, P_j). \quad (4.17)$$

We determine n_j and P_j for job j to optimize (4.16).

4.4.3 Performance Evaluation

In this subsection, we present experiments and results to show the effectiveness of our demand-response model exploiting the power-capping capability. We used two benchmarks:

- Demand-Response, which determines the optimal power-capping value based on the optimization problem and solution given in (4.14) during a demand response period. It chooses the maximum power-capping limit during the normal operating time.

- Non-Demand-Response, which always chooses the maximum power-capping limit to ensure best application performance. We choose this benchmark since it ensures the high-performance requirements of HPC applications. Such policy is also denoted as the default method for power allocation to processors [LSSS16].

To evaluate our design, we used a real-life workload trace from the parallel workload archive [FTK14]. The trace was collected at the San Diego Supercomputer Center (SDSC SP2), which contains 5,000 jobs. This trace has been widely-used in various studies [DSRI13, KMY15]. The trace includes information about job start time, job run time, job wait time, requested number of processors, and so on. The workload trace, however, does not contain any power-related information. For that, we ran real-life applications on the cluster to measure the power consumption. The details of the applications' power and execution time are given in Section 4.4.1.

Fig. 4.12 compares the two benchmarks with different demand response events ratio. We vary the demand response events ratio from 25% (i.e., a demand response event lasts 25% of the entire operation duration) to 100%. As can be seen in Fig. 4.12(a), Demand-Response achieves reduced per job average energy consumption compared with the Non-Demand-Response benchmark. The energy saving is

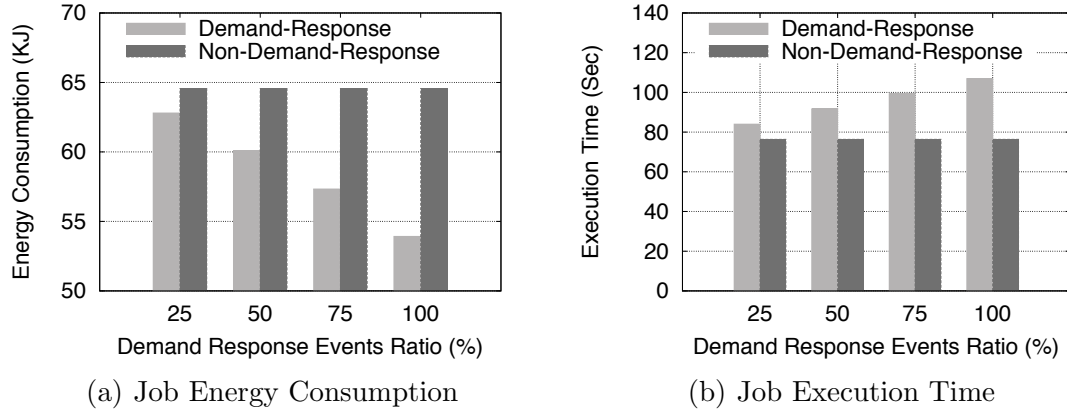


Figure 4.12: Benchmark comparison with power capping.

more pronounced as the demand response events ratio increases. Demand-Response incurs only a moderate increase in the per job average average execution time compared with that of the performance-mode. Since Demand-Response reduces the energy consumption during the critical demand response periods, the increase in application execution time is understandable.

In the rest of the subsection, we present an experiment to show the effectiveness of the demand-response model with both power-capping and node-scaling capabilities. We compare the following two benchmarks:

- Demand-Response-Scale, which determines optimal power-capping value and node allocation that reduces EtS. The benchmark solves (4.16) to determine the optimal resource allocation during the demand response period. The benchmark allocates all available nodes and chooses the maximum power-capping limit on all nodes during the normal operation time.
- Non-Demand-Response-Scale, which always chooses the maximum power-capping limit to ensure the best application performance. It also allocates the maximum number of nodes to the job. Note that, this allocation policy is also chosen in the literature as baseline case [SLK⁺13].

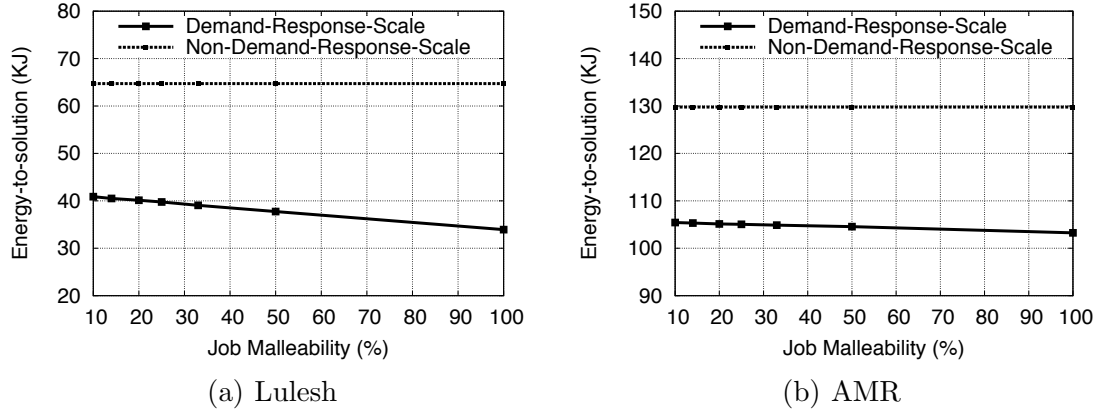


Figure 4.13: Benchmark comparison with power capping and node scaling.

We collected the execution time and energy consumption data from [LDKP15] for the following two HPC applications: AMR (an application for adaptive mesh refinement simulations) and LULESH (a shock hydrodynamic modeling application for unstructured meshes). The applications were executed on 4, 8, 12, and 16 nodes. The power-capping values were set to be between 31 W and 55 W with 3 W intervals. We collected the measurements from [LDKP15] and used the prediction model in Section 4.4.2 to predict the application characteristics (i.e., EtS) for unknown node numbers and power-capping values. We used the SDSC SP2 trace from the parallel workload archive [FTK14] for other job-related information (e.g., job start time, job wait time).

Fig. 4.13 compares the results. Fig. 4.13(a) shows the effect of running the jobs using the power and performance data of the Lulesh application. Fig. 4.13(b) shows the effect of using the power and performance data of AMR application. As evident from the figures, the Demand-Response-Scale benchmark always achieves smaller EtS than the Non-Demand-Response-Scale benchmark. This is because Demand-Response-Scale considers optimization of energy consumption during the demand response event through both power capping and node scaling, while Non-Demand-Response-Scale does not consider such optimization. In the figures, job

malleability denotes the percentage of jobs that are allowed to change the job size. We change the job malleability from 10% (in which case, only 10% of the jobs can be distributed on an optimal number of nodes) to 100%. As expected, with higher job malleability, more jobs can be flexible in choosing the job size for optimized performance, increasing the opportunity to reduce the energy consumption with our proposed algorithm.

4.5 Summary

In this part of the dissertation, we propose emergency demand-response-aware job scheduling and resource provisioning algorithm for HPC systems. The job scheduling algorithm operates between the power-constrained energy-conservation mode (using DVFS) and the performance-conservation mode, depending on whether the system is in a demand response period or not. We develop a scheduler simulator to evaluate the effectiveness of our approach. The simulator has been validated by comparing with existing simulators using real-life workload traces. We performed evaluation studies and results have demonstrated that our proposed emergency demand response method can achieve energy savings with only moderate impact to the application performance due to demand response. Next, we study HPC system demand response participation and propose an emergency demand-response model that leverages power-capping and node-scaling capabilities. We present power, performance, and energy prediction models for HPC applications with unknown power-capping values and job sizes. We validated our prediction models using real-life measurement of application characteristics (including both power and execution time) and compared our models with approaches in literature. We propose an HPC emergency demand-response model by selecting optimal power limit and job size.

Using real-life measurements and trace-based data, we examined the effectiveness of our proposed approach and compare it with existing approaches. Our model can effectively reduce the HPC system's energy consumption during critical demand response periods and by doing so enable emergency demand response participation from HPC systems.

CHAPTER 5

ECONOMIC DEMAND RESPONSE FOR HPC SYSTEMS

HPC systems can consume an enormous amount of electricity during their operation, and consequently incur significant energy cost. In this part of the dissertation, we propose an economic demand-response model to allow both HPC operators and HPC users to jointly reduce the energy cost. More specifically, we apply the contract theory originated from economics for studying the contractual arrangements among economic actors, and design a rewarding scheme to ensure participation of both HPC operators and HPC users in the demand response program.

5.1 Background

Energy cost is a major part of the overall cost-of-operation of HPC systems. With significant increase in the size of the supercomputers recently, the energy consumption has also increased. For example, the recently-built Summit supercomputer at Oak Ridge National Laboratory in United States has reached a peak power consumption of 15 MWs. Considering that 1 MW electricity costs approximately \$1 million, the Summit supercomputer would need to add a \$15 million annual recurring cost to the overall operation expense. Demand response is a scheme adopted by energy consumers to reduce energy use during high energy load periods at the request of the power grid.

Various demand response programs (including both economic demand response and emergency demand response programs) are offered by energy service providers to encourage energy reduction from participants. Fig. 5.1 shows an example of the hourly energy reduction amount from various participants of PJM and location marginal pricing (LMP) value offered for participation in economic demand response program on a typical day [PJM17]. In this work, we particularly study

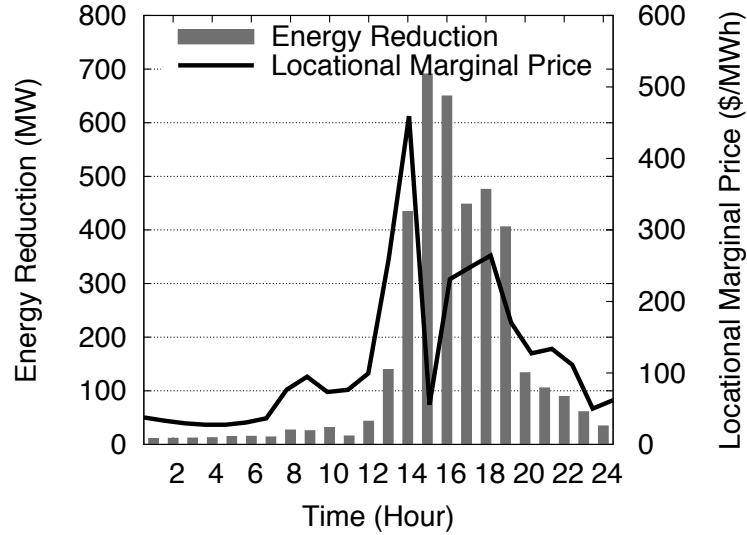


Figure 5.1: Energy reduction and electricity pricing from PJM economic demand response on July 18, 2013 [PJM17].

the possibility of HPC systems in economic demand response participation through *voluntary* energy reduction. By doing so, HPC systems can reduce their overall cost of operation. To achieve this goal, however, one main challenge is to incentivize the HPC users to participate in the demand response program. Since participation in demand response program can introduce delays in application execution, HPC users may be reluctant. Consequently, a proper rewarding mechanism is necessary to enable willing participation from HPC users in the demand response program.

In general, an incentive mechanism should reward users commensurate to their contributions: an HPC user with more contribution toward demand response should be allowed with more rewards compared to users with lower contribution, and vice versa. Yet, an HPC operator may not have the information of HPC users' willingness to participate. There is information asymmetry between the HPC operator and the HPC users. A proper rewarding mechanism must reward HPC users based on their willingness to participate. As such, we resort to the contract theory, a powerful tool from microeconomics, where an employer (in this case, the

HPC operator) offers contracts to the employees (the HPC users) based on the employees' preference and yet under information asymmetry. Contract theory is a well-established rewarding mechanism which has been used in different areas, such as spectrum trading in relay network [GWXZ11, ZSS⁺15], mobile cloud computing environment [KBC⁺11, DKS⁺14], and device-to-device communication in cellular network [SNHH14]. In summary, to motivate HPC users' energy reduction for HPC system demand response, we propose a *contract design* approach where an HPC operator offers a set of *contracts* (i.e., energy reduction goals and financial rewards) to HPC users who can voluntarily select either none or one of the contracts to accept to run the jobs. Our approach addresses the energy reduction problem in HPC systems through the contract-based demand response mechanism.

5.2 Related Work

We provide the background and discuss existing work related to our proposed contract-based economic demand response model for HPC systems. In particular, we describe existing approaches for reducing energy cost in HPC systems, introduce existing demand response models for data centers, and provide a brief overview of the contract theory and its applications.

5.2.1 Reducing Energy Cost

Utility companies charge their customers based on two types of costs: (1) energy charge, and (2) demand charge. For example, HPC facilities, such as the one in the Oak Ridge National Laboratory, pay two types of electricity charges from the utility company: energy charge and demand charge [PBG⁺16]. Energy charge is determined based on the total energy consumption of the system. Demand charge

is calculated based on the peak power usage of a customer during a billing cycle (usually a month). This charge is introduced by the utility company to compensate for the support in running the infrastructure at the maximum capacity. In this work, we consider both energy charge and demand charge. Our model takes into consideration the time-varying electricity price and tries to reduce the energy cost by focusing on the peak price periods.

There are different approaches to reducing energy cost for HPC systems. One can explore job scheduling to reduce the size of the electricity bill. For example, Yang et al. [YZW⁺13] proposed a job scheduling approach to exploit variable electricity price and power consumption profile of jobs. Each day is divided into two parts based on the electricity price: on-peak and off-peak. Jobs are classified based on their power profile (derived from past execution data). Low power-consuming jobs are preferably chosen to run during the on-peak time periods, while high power-consuming jobs are preferably run during the off-peak time periods. Zhou et al. [ZLTD13] also proposed an electricity pricing-aware job scheduling and power budgeting approach (based on on-peak and off-peak electricity pricing). However, their approach specifically focuses on the IBM Blue Gene/P systems, and may not be easily generalized to other systems. Murali et al. [MV15] proposed a method to exploit the spatio-temporal variation in electricity price and use it to dispatch workloads to geographically distributed supercomputers. Their scheduling algorithm is formulated as a minimum cost maximum flow problem. They use different prediction algorithms to calculate the response times of the jobs and electricity cost variations. Several strategies have been evaluated in [AS11] to exploit variations in electricity prices, carbon intensity and renewable energy. Through adapting the workload execution in accordance with the time variation in electricity price, one can demonstrate the potential in energy cost saving. Our proposed economic de-

mand response model also considers time-varying electricity price and workload for reducing the energy cost in HPC systems.

Reducing peak power usage has been studied extensively for data centers. For example, Liu et al. [LWC⁺13] proposed to use workload scheduling and backup power generation to perform peak shaving to enable data center demand response participation. Zhou et al. [ZYGL15] exploited two types of energy storage devices (ESDs), battery storage and thermal energy storage, to reduce the data center operation cost through shaving peak power consumption. Shi et al. [SXZW16] proposed to exploit ESDs exclusively to reduce peak power consumption in data center. Specific to HPC systems, some recent studies have focused on peak shaving. For example, Chiesi et al. [CVM⁺15] performed HPC job scheduling in heterogeneous CPU-GPU architectures to reduce the peak power under a predetermined budget. In this work, we consider only resource management schemes with frequency scaling capabilities, although methods such as using ESDs can be considered for HPC systems demand response.

5.2.2 Demand Response

Energy cost reduction through demand response is a well-studied topic in certain areas (such as data centers and smart buildings). As a large energy consumer, data centers have proven beneficial to the demand response programs [WLLMR14]. Data center demand response can exploit different workload scheduling (such as load shifting in time [LWC⁺13] and load balancing among geographical locations [WHLMR16]) and resource provisioning (such as frequency scaling [LLRL12]) to enable demand response participation.

It is important to note that data center demand response deals with specific types of workload (such as map-reduce applications or network transaction based applica-

tions), which are quite different from HPC applications. HPC applications possess unique power and performance characteristics. Unlike data center jobs, many HPC applications are less flexible with changing job size, and most are non-interactive (delay-tolerant) jobs submitted through a batch job scheduler that collectively schedules, manages, and monitors jobs running on the HPC systems. More importantly, HPC users usually have different expectations from data center users: achieving high performance is typically the main objective of running those jobs on HPC systems in the first place. With this mentality, energy saving is at best a secondary consideration. It is thus a challenge to translate energy saving concerns of an HPC operator into a judicious user decision making of running computational tasks with best energy conservation which can often lead to suboptimal runtime performance.

5.2.3 Contract Theory and Applications

A contract is the agreement between involved parties that specifies various actions that the parties are supposed to take at different times. There is the “principal” who offers the contracts; there are “agents” who are offered with the contracts and make decisions on whether to accept or refuse the contracts. Contract theory studies how the parties deal with the contracts in the presence of hidden or asymmetric information. The agents keep their private information (e.g., the *type* to which the agent belongs). The principal may be uninformed about agents’ private information, and yet needs to design the contracts in order to maximize agents’ participation as well as the principal’s payoff. Such problem is termed as the *adverse selection* problem or *hidden information* problem. This problem was studied by [Ake70]. A contract-based solution is to offer a menu of contract bundles. See, for example, [GL84].

Contract theory has been applied in many fields, including economics and communication systems (e.g., radio network, vehicular network, and cellular network).

In economics, for example, contract theory has contributed to formulating labor contract (employer offering wage contract to employees with unknown productivity), and financial contract (lender deciding investment with unknown profitability). In radio networks, there have been studies to enable cooperation among relay nodes to improve relay performance (e.g., [ZWXL15, HJB12]). Contract theory has helped in developing models to incentivize participation of relay nodes in cooperative relay. Nazari and Jamalipour [NJ14] proposed a power allocation and price assignment algorithm to maximize utility of multi-hop wireless networks. Their algorithm is based on a contract-auction mechanism in the presence of asymmetric information. Recent studies [GCWL13, WNC⁺14] have also applied contract theory to coordinate a large number of electric vehicles (EVs) to appropriately charge or discharge as ancillary services to the power grid, based on information asymmetry between the EVs and the power grid (e.g., different EVs have different preferences toward charging/discharging at different times). To encourage device-to-device (D2D) communication in cellular networks, various contract-based approaches have been proposed to increase network capacity [ZSS⁺15, MLY⁺16]. In them, information asymmetry exists since the base station may not know user’s willingness to contribute to the D2D communication.

5.3 Model

In this section, we describe the system model along with different notations used. We consider a discrete-time model and divide the entire time period into T time slots of equal length (e.g., 15 minutes or one hour). We denote the time slots by $t = 1, \dots, T$ and duration of each time slot by t_s . To simplify the notations, we exclude the time notations from the model. We decide and allocate resources at the

Table 5.1: List of Key Notations

Notation	Description
Θ	Set of job types, $\Theta = \{\theta_1, \theta_2, \dots, \theta_i, \dots, \theta_n\}$
θ_i	Flexibility of job type i
Δe_i	Energy reduction by job type i
Δt_i	Percentage increase in execution time for job type i
r_i	Reward awarded to job type i
n_i	Number of processors required for job type i
f_i	Frequency allocated to processors for job type i
ϕ	Electricity price

beginning of each time period. The key notations and their meanings are given in Table 5.1, while description of these terms and some other notations are given in the text.

5.3.1 An Example

We start with a hypothetical example to demonstrate how our proposed contract-based model would work. We assume for simplicity there are only three types of jobs: $\{\theta_1, \theta_2, \theta_3\} = \{0.2, 0.4, 0.8\}$. The θ values, which can any positive numbers, are preselected to represent the flexibility of the job types to participate in the demand response. A lower value indicates that the job type is more flexible and the HPC operator can have more flexibility (or cause less inconvenience to the users) to "stretch" the runtime of the jobs belonging to this type in order to save the energy. A higher value, on the contrary, would mean that the job type is more rigid; the HPC operator has less flexibility (or cause more inconvenience to users) to "stretch" the runtime of the jobs to save energy. When job submission, the user will select a job type for the submitted job depending on the urgency of the job's completion. More rigid job types generate less opportunities for the HPC operators to change the execution of the jobs to save energy. To incentivize the participation, a user can

expect more reward given for jobs with more flexibility. As an example, we consider three jobs currently in the system, denoted by job#1, job#2, job#3. Assume we assign job#1 to type θ_1 , job#2 to type θ_2 , and job#3 to type θ_3 . That is, job#1 is the most flexible job among the three, and job#3 is the least flexible job.

At each time slot, the HPC operator considers all jobs in the submission queue (three in this example). However, she does not know the types of these jobs. This information is considered private to the users. (Giving this information to the HPC operator would be disadvantageous to the users as the HPC operator may use this information to exploit the users and trick them into suboptimal contracts with less rewards). The HPC operator knows the current electricity price, say, \$20/kWh, and with this information, she can design a contract bundle for each job type. A contract bundle contains two elements: the monetary reward, and the expected increase in percentage of time to run the jobs belonging to this type. In our example, the HPC operator would provide three contract bundles, one for each job type: (\$2550, 30), (\$1200, 15), (\$200, 5). Details on how this is done will be discussed in the sections to follow.

Upon receiving the three contracts, the users need to determine for each job which of three contracts to accept, or reject them all. To do that, the user needs to calculate a utility for each job. A job must have positive utility to accept a contract; in addition, a job will choose the contract type that can maximize the utility. There are several ways to define the utility. For our model, we define a job's utility with respect to a contract as the difference between the reward and the inconvenience cost, which is a function of the percentage increase in execution time. For the example, we use $\theta_i c \Delta t_i$ as the inconvenience cost, where we set $c = 10$ and Δt_i is the percentage increase in execution time for job type i (Eq. 5.4).

As such, the utility of job#1 for job type 1 contract (\$2550, 30) can be calculated as $2550 - 0.2 \cdot (30)^2 \cdot 10 = 750$. Similarly, the utility of job#1 for job type 2 contract (\$1200, 15) is $1200 - 0.2 \cdot (15)^2 \cdot 10 = 750$, and the utility of job#1 for job type 3 contract (\$200, 5) is $200 - 0.2 \cdot (5)^2 \cdot 10 = 150$. Therefore, the user submitting job#1 may select the contract for job type 1 as an optimal decision. Using the same argument, the utilities of job#2 for the three contracts are -1050, 300, and 100, respectively; and the utilities of job#3 are -4650, -600, and 0, respectively. The users submitting job#2 and job#3 will select the contract for job type 2 and 3, respectively. This is not a coincidence. As we show later, contract theory guarantees that optimally the users choose the contracts designed for the job types to which their jobs belong.

At the system side, the HPC operator can calculate the total energy savings from the three jobs if the users choose to enter the respective contracts for the rewards and therefore willingly extend the execution time of the jobs with the specified percentage. Energy saving is achieved by the HPC operator lowering the CPU frequency of the processors running the jobs to a predetermined value (determined by an energy optimization model). The total amount of energy saved can be translated and distributed among the users as monetary rewards stipulated in the contracts. After providing the rewards to the users, the operator shall still maintain a positive amount for its own demand response participation. As a result, it creates a win-win situation for both HPC users and the HPC operator, as well as the energy supplier for reduced energy consumption.

5.3.2 HPC System Model

We consider an HPC system with one HPC operator and set of Θ jobs types. We assume that jobs are distributed to one of the job types denoted by $\Theta =$

$\{\theta_1, \theta_2, \dots, \theta_i, \dots, \theta_n\}$. The operator will offer a menu of contracts with the contract bundle intended for job of type i as $(r_i, \Delta t_i)$, where r_i is the reward to the job of type i and Δt_i is the application performance change for jobs belonging to type i due to demand response participation. Note that the operator needs to design one contract item for each job type in order to obtain the optimal result [BD05].

We now formulate the job power consumption and execution time characteristics. To determine the average power consumption, we use a similar model as proposed in [WTCM16]. We assume that a job of type i runs at frequency f_i on the requested processors. We estimate the processor's power consumption at frequency f_j for job of type i using the following third-order polynomial function:

$$p(f_i) = a_i + b_i \cdot f_i + c_i \cdot f_i^2 + d_i \cdot f_i^3 \quad (5.1)$$

where a_i , b_i , c_i , and d_i are constants determined from empirical analysis of average power relation with different frequency values. Here, a_i represents the static power consumption while running the jobs. We assume that HPC operator can determine the parameters based on historical data and update the parameters for each job type with more learned data related to power consumption behavior.

During each time slot, the HPC system operator determines the optimal configuration (i.e., frequency allocation) for the incoming jobs. While there are many different energy-saving techniques available, we consider frequency allocation as a possible method in this work. To reduce power consumption with particular focus on demand response, the HPC operator allocates optimized speeds to processors for new jobs. We assume that changed server frequency of servers allocated to job of type i is denoted by f'_i . The average power consumption by job of type i with such changed speed is denoted by:

$$p(f'_i) = a_i + b_i \cdot f'_i + c_i \cdot f'^2_i + d_i \cdot f'^3_i$$

The following constraint ensures the upper and lower limits for changed frequency allocation for job of type i :

$$f_{min} \leq f'_i \leq f_{max} \quad (5.2)$$

where, f_{min} and f_{max} denote the lower and upper limit of frequency allocation to processors, respectively.

In the same fashion we estimate the execution time for job of type i at frequency f_i using the second-order polynomial function:

$$t(f_i) = \alpha_i + \beta_i \cdot f_i + \gamma_i \cdot f_i^2 \quad (5.3)$$

where, α_i , β_i , and γ_i are regression coefficients determined from polynomial fitting function using empirical data. After changing the frequency to f'_i , the execution time for job of type i can be represented as follows:

$$t(f'_i) = \alpha_i + \beta_i \cdot f'_i + \gamma_i \cdot f'^2_i$$

We assume that by default the jobs are run at highest frequency to achieve the best performance. Energy consumption for job of type i at maximum frequency can be represented as following: $e_i = n_i \cdot p(f_{max}) \cdot t_s$, where t_s denotes duration of time period (e.g., 15 minutes or 1 hour). We consider $t_s = 1$ hour for this work since various values (such as the electricity price) are reported hourly. Energy consumption for job of type i at changed frequency f'_i can be represented as following: $e'_i = n_i \cdot p(f'_i) \cdot t_s$. Therefore, energy reduction by job of type i through frequency scaling can be represented as $\Delta e_i = e_i - e'_i$. Moreover, the change in execution time due to changing the frequency to f'_i is denoted by $t(f_{max}) - t(f'_i)$. We denote the percentage change in execution time by $\Delta t = 100 \cdot [t(f_{max}) - t(f'_i)]/t(f_{max})$.

HPC users need to be compensated for changes in their application execution performance, so that they are willing to participate in demand response program for

energy reduction. Now we formulate the HPC jobs' inconvenience cost (or implicit cost [MHL⁺13]) due to demand response participation. We denote job of type i 's flexibility to participate in demand response by θ_i . A higher value of θ_i would denote that the job of such type is less flexible towards participation in the demand response program (e.g., due to deadline for project completion). On the other hand, some jobs would be more flexible and correspondingly have lower value of θ_i . Such information may be private to the user, which produces an information asymmetry between the job types and operator. We denote the job of type i 's incurred inconvenience cost due to demand response participation as a general function of execution time change, denoted by the following:

$$v(\theta_i, \Delta t_i) = \theta_i \cdot c(\Delta t_i) \quad (5.4)$$

where Δt_i denotes the percentage execution time change due to demand response participation for job of type i . Without loss of generality, we assume that $c(\Delta t_i) = c_0 \cdot (\Delta t_i)^2$, where c_0 is the coefficient to convert the quantity into monetary value. Here, other functions can be incorporated as well. Representing inconvenience as a quadratic cost function is common in economics and has also been considered in other applications [ZWXL15, KC15]. HPC operator may not have complete information of how jobs are distributed to different types. Since there is information asymmetry between the operator and the users, we resort to contract theory to design an incentive mechanism to ensure users' willing participation in the demand response program.

We define the utility for jobs of type i as following:

$$u_i = r_i - v(\theta_i, \Delta t_i), \forall i \in \{1, 2 \dots n\} \quad (5.5)$$

The objective of the HPC operator is to determine the optimal resource allocation to incoming jobs and corresponding rewards such that HPC system's profit from

demand response participation is maximized. From the HPC operator’s perspective, the utility function in terms of type- i job can be defined as:

$$u_{op}^i = m_i \cdot (\phi \cdot \gamma \cdot \Delta e_i - r_i)$$

where m_i denotes the number of jobs belonging to job type i and ϕ denotes electricity price. For the user’s participation to be of benefit to the HPC operator, the following needs to be satisfied: $u_{op}^i \geq 0$. Note that we focus on the server power consumption, while the power consumption of non-IT parts of the HPC system such as cooling and power supply system is captured using the factor of power usage effectiveness (PUE), denoted by γ . The total utility for HPC operator can be represented as following:

$$u_{op} = \sum_{i=1}^n u_{op}^i. \tag{5.6}$$

The HPC operator may not have complete information on the job types (e.g., what jobs are assigned to what type). However, the operator may maintain statistical information on the distribution of jobs (i.e., the probability of a job belonging to a certain type i). Based on historical data, the HPC operator can approximate the optimal choices for the job types and provide rewards to the HPC users accordingly. Although such information may be quite imprecise initially, the HPC operator can gradually improve the estimate through “online learning” [GCWL13, SL14]. In this way, the operator can learn and update the job types based on whether the offered contracts are accepted or not.

5.4 Formulation and Algorithm

In this section, we formulate the HPC system demand response participation problem and outline the algorithm. We first define two constraints that are essential for designing a *direct revelation contract*¹.

Definition 5.4.1 (Individual Rationality) *Individual rationality (IR) constraint or participation constraint ensures that participants in contract mechanism achieve non-negative pay-off or utility. This ensures that an HPC user has an incentive to participate in the demand response program. Mathematically, for job type i , IR constraint can be described as following:*

$$u_i = r_i - \theta_i \cdot c(\Delta t_i) \geq 0 \quad (5.7)$$

The reward received must compensate participation through energy reduction to motivate user's participation. However, if $u_i < 0$ the user will not participate in demand response program.

Definition 5.4.2 (Incentive Compatibility) *Incentive compatibility (IC) constraint represents the fact that utility is maximized when participant chooses own contract type over the other contract types. It reflects that, although a participant of type i can choose a contract intended for another type (which we denote by i'), the utility is maximized when the user chooses its own type i . Mathematically, for type i , IC constraint can be described as following:*

$$r_i - \theta_i \cdot c(\Delta t_i) \geq r_{i'} - \theta_i \cdot c(\Delta t_{i'}). \quad (5.8)$$

¹Direct revelation principle ensures that the agents are willing to reveal their true identity to the mechanism designer

Now, the optimization objective along with the constraints can be represented as following:

$$\begin{aligned} & \max_{(\Delta e, r)} \sum_{i=1}^n m_i \cdot [\phi \cdot \gamma \cdot \Delta e_i - r_i] \\ & \text{s.t.} \quad \text{Eqs. (5.2), (5.7), and (5.8)} \end{aligned} \tag{5.9}$$

where ϕ denotes electricity price and m_i denotes number of jobs of job type i . The HPC operator determines the optimal energy reduction amount Δe , corresponding reward r and performance change Δt for each job type and shares the contract bundle $(r, \Delta t)$ to the users. Note that, the operator may not have information about the job distribution information to different types. However, the operator may assume job distribution information to different types (e.g., normal distribution) as considered in literature [NJ14, KC15]. The operator then can maximize the expected utility to derive the optimal energy reduction amount and reward for each job type. The problem is intractable to solve due to large number of IR and IC constraints. However, we can resort to constraint reductions to reduce the number of constraints and make the problem solvable, which we present in the next section.

5.4.1 Feasibility and Optimality of Solutions

In this subsection, we provide contract feasibility and optimality for discrete-type problem formulation in (5.9). The following two conditions are necessary conditions to ensure feasibility and optimality of designed contracts:

Conditions for feasibility. We assume the contract set is $C = \{(r_i, \Delta t_i)\}$ with the following relation: $\theta_1 < \theta_2 < \dots < \theta_n$. Then the contract is feasible if and only if:

$$r_n - \theta_n \cdot c(\Delta t_n) \geq 0 \quad (5.10)$$

$$r_{i-1} - \theta_{i-1} \cdot c(\Delta t_{i-1}) \geq r_i - \theta_{i-1} \cdot c(\Delta t_i),$$

$$\forall i \in \{2, \dots, n-1\} \quad (5.11)$$

Conditions for optimality. For the optimal solution, the individual rationality condition for the highest type and adjacent ICs are binding, and all other conditions can be ignored. Let a feasible contract list be given by $(r, \Delta t)$. Then, we can state the following:

$$r_n^* = \theta_n \cdot c(\Delta t_n) \quad (5.12)$$

$$r_{i-1}^* = r_i^* + \theta_{i-1} \cdot (c(\Delta t_{i-1})) - c(\Delta t_i),$$

$$\forall i \in \{2, \dots, n-1\} \quad (5.13)$$

Based on the above conditions, we can state the following theorem and reduced optimization problem.

Theorem 5.4.3 (Contract Feasibility) *The designed reduced contract-based formulation ensures IR and IC constraints and hence the formulation is feasible.*

Proof. We first consider a case with two types of HPC jobs. Next, we extend and generalize to multiple types of HPC jobs.

A Two-Type Scenario.

Suppose there are two types of HPC jobs so that $\Theta = \{\theta_1, \theta_2\}$, with $\theta_1 < \theta_2$. As before, the reward for lowering usages is denoted by r_j and the cost of doing so is given by $\theta_i c(\Delta t_j)$, where j may or may not equal i (the HPC operator may choose the contract intended for the other type).

This yields the following incentive compatibility (IC) constraints:

$$IC_1 : r_1 - \theta_1 c(\Delta t_1) \geq r_2 - \theta_1 c(\Delta t_2), \quad (5.14)$$

and

$$IC_2 : r_2 - \theta_2 c(\Delta t_2) \geq r_1 - \theta_2 c(\Delta t_1). \quad (5.15)$$

The participation constraints (or individual rationality constraints) are given by:

$$IR_1 : r_1 - \theta_1 c(\Delta t_1) \geq 0,$$

and

$$IR_2 : r_2 - \theta_2 c(\Delta t_2) \geq 0.$$

Since $\theta_1 < \theta_2$ combining the Eqs. 5.14 and 5.15 yields

$$r_1 - \theta_1 c(\Delta t_1) \geq r_2 - \theta_1 c(\Delta t_2) > r_2 - \theta_2 c(\Delta t_2) \geq r_1 - \theta_2 c(\Delta t_1),$$

which implies

$$r_1 - \theta_1 c(\Delta t_1) > r_2 - \theta_2 c(\Delta t_2) \geq 0. \quad (5.16)$$

So θ_1 's IR must be satisfied strictly.

Now, adding the ICs yields

$$r_1 - \theta_1 c(\Delta t_1) + r_2 - \theta_2 c(\Delta t_2) \geq r_2 - \theta_1 c(\Delta t_2) + r_1 - \theta_2 c(\Delta t_1),$$

which implies

$$-\theta_1 c(\Delta t_1) - \theta_2 c(\Delta t_2) \geq -\theta_1 c(\Delta t_2) - \theta_2 c(\Delta t_1),$$

or

$$[\theta_2 - \theta_1]c(\Delta t_1) \geq [\theta_2 - \theta_1]c(\Delta t_2),$$

which implies

$$c(\Delta t_1) \geq c(\Delta t_2).$$

Since $c(\cdot)$ is increasing, we have

$$\Delta t_1 \geq \Delta t_2.$$

The intuition here is that the lower cost job θ_1 will reduce usage by a larger amount, hence incur larger increase in execution time. Note that type θ_1 can act like (or choose the contract intended for) θ_2 , but we do not need to worry about θ_2 acting like θ_1 .²

These imply that we are concerned with only the following two constraints:

$$IC_1 : r_1 - \theta_1 c(\Delta t_1) \geq r_2 - \theta_1 c(\Delta t_2),$$

and

$$IR_2 : r_2 - \theta_2 c(\Delta t_2) \geq 0.$$

Both of these constraints must bind. If not, the HPC operator could reduce r_i and receive a higher payoff. So we have

$$IC_1 : r_1 - \theta_1 c(\Delta t_1) = r_2 - \theta_1 c(\Delta t_2),$$

and

$$IR_2 : r_2 - \theta_2 c(\Delta t_2) = 0.$$

We can solve the reduced optimization problem through standard optimization solving algorithm.

Extension to n-type Case.

Next we can consider extending this two-type case to a setting with $\Theta = \{\theta_1, \theta_2, \dots, \theta_n\}$. The cost is given by $\theta_i c(\Delta t_i)$, with c increasing. So, thinking in terms of the continuum of types case of Section 5.4.2, we have the analogue of this having a strictly positive cross partial with respect to θ_i and Δt_i . Thus, this satisfies the Spence-Mirrlees property so that satisfying the “local” constraints implies the global constraints are satisfied. More precisely, we can define:

$$U_i = r_i - \theta_i c(\Delta t_i).$$

²We can verify this after finding the optimal menu of contracts. This follows a standard procedure. See, for example [LM09].

So we have $U_{\Delta t_i} = -\theta_i c'(\Delta t_i)$ and $U_{r_i} = 1$. Thus, we have

$$\frac{U_{\Delta t_i}}{U_{r_i}} = -\theta_i c'(\Delta t_i),$$

which implies

$$\frac{\partial}{\partial \theta_i} \frac{U_{\Delta t_i}}{U_{r_i}} = -c'(\Delta t_i) < 0.$$

This implies that the Spence-Mirrlees property is satisfied and we have a monotonicity property. Thus, we need only to consider local constraints or the constraints associated with adjacent types.³ Therefore, the reduced optimization problem is feasible and satisfies the IR and IC constraints. \square

After reduction of number of IR and IC constraints, the overall optimization problem can be formulated as following:

$$\begin{aligned} \max_{(\Delta e, r)} m_i \cdot \left[\sum_{i=1}^n \phi \cdot \gamma \cdot \Delta e_i - r_i \right] \\ \text{s.t. } r_n - \theta_n \cdot c(\Delta t_n) = 0 \end{aligned} \quad (5.17)$$

$$r_{i-1} - \theta_{i-1} \cdot c(\Delta t_{i-1}) = r_i - \theta_{i-1} \cdot c(\Delta t_i) \quad (5.18)$$

$$f_{min} \leq f'_i \leq f_{max}$$

The HPC operator can solve the above reduced optimization problem using standard optimization solving methods, such as the sequential least squares programming algorithm using the Han-Powell quasi-Newton method [Pow78]. Operator will then distribute the contract bundles to the users for their selection. Alternatively, the problem can also be solved taking Lagrangian multipliers to the equality constraints and then solving the Lagrangian formulation, as also considered in the literature [ZJYH16, ZSS⁺15].

³Ref. [LM09] contains a good discussion of these ideas, and the setting here is similar to theirs with the reward being analogous to the agent's transfer and $\theta_i c(\Delta t_i)$ being analogous to the quantity the agent produces.

5.4.2 Contract Design with Continuum Type

In this subsection, we present contract design formulation of continuous types and present theoretical analysis. We assume that the types are continuum (i.e., types are denoted by continuous function $[\theta', \theta'']$, where θ' denotes initial value of type and θ'' denotes ending value of type). Under asymmetrically incomplete information, we assume that $\{f(\theta)\}_{\theta \in \Theta}$ denotes probability distribution function (PDF) of types and $\{F(\theta)\}_{\theta \in \Theta}$ denotes the cumulative distribution function (CDF). HPC system operator's objective in such continuum of types is given by:

$$\max_{(\Delta e, r)} \int_{\theta'}^{\theta''} \left[\sum_{\theta \in \Theta} \phi \cdot \gamma \cdot \Delta e(\theta) - r(\theta) \right] |f(\theta) d\theta \quad (5.19)$$

$$\text{s. t. } r(\theta) - \theta \cdot c(\Delta t(\theta)) \geq 0, \forall \theta \quad (5.20)$$

$$r(\theta) - \theta \cdot c(\Delta t(\theta)) \geq r(\bar{\theta}) - \theta \cdot c(\Delta t(\bar{\theta})), \forall \theta, \bar{\theta} \quad (5.21)$$

Note that, with a continuum types, the number of IC constraints becomes intractable due to double continuum of IC constraints.

Theorem 5.4.4 (Reduction of Continuum Types) *The intractable continuum IC constraints can be reduced and solved.*

Proof. We reduce the number of IC constraints based on the Single-Crossing property (or Spence-Mirrless condition) [Mir71], where IC constraints are reduced by corresponding first order condition (FOC) [Mir71]. The incentive constraint Eq. 5.8 for continuum case can be replaced by following two conditions: $\Delta t(\theta)$ and $r(\theta)$ are incentive compatible iff

$$-c'(\Delta t(\theta)) \cdot \frac{d\Delta t(\theta)}{d\theta} \leq 0 \quad (5.22)$$

$$\frac{d(r(\theta))}{d\theta} \leq 0. \quad (5.23)$$

Eq. 5.22 is referred to as *local adjacency* constraint and follows from the FOC and second order condition for payoff maximization by type θ . Eq. 5.23 is called the monotonicity constraint.

This is analogous to the conditions in [LM09] and the Spence-Mirrlees condition applies, as in the binary case above, and the local constraints imply the global constraints.⁴ Further, for settings in which the monotone hazard ratio property is not satisfied everywhere along the distribution $f(\theta)$, we can resort to *Myerson's ironing* mechanism to design contract for the HPC users [Mye81]. \square

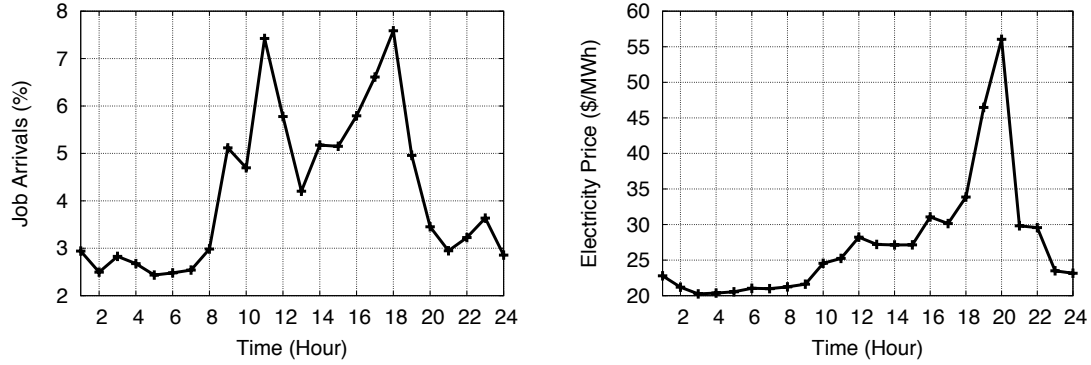
5.5 Experiments

In this section, we present a trace-based simulation study to validate our proposed approach and show effectiveness. First, we present the data set we used for benchmarking. Then, we present how our proposed approach performs for the trace data and show that basic properties of contract theory is preserved. We then compare our proposed approach with a system that does not consider demand response participation. Finally, we present a sensitivity study.

5.5.1 Data Set

We considered an HPC system which includes one HPC operator and six job types (denoted as Type#1, Type#2, Type#3, Type#4, Type#5 and Type#6). The job types' flexibilities are denoted as $\{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6\} = \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$. The PUE for the system is considered to be 1.3. We collected real-life HPC workload trace from the parallel workload archive [FTK14]. The workload traces contain runtime information collected at the University of Luxemburg Gaia HPC cluster

⁴See also [Hol16].



(a) Percent of job arrivals during different hours (b) Electricity price data on a day of October 2017 at PJM

Figure 5.2: Job arrivals and electricity price data.

during the time period from May 2014 through August 2014. The trace contains 51,987 jobs. The trace includes information about job start time, job run time, job wait time, requested number of processors, and so on. We processed the data and collected number of job arrivals during each hour to determine percentage job arrivals during a day. Fig. 5.2(a) presents the percentage of job arrivals during each hour of a day. We collected electricity price data for a 24-hour period on a day of October 2017 from PJM Interconnect. The data is shown in Fig. 5.2(b). We consider the cost for job execution on the system to be 0.023\$/hour, which is a pricing used in Amazon EC2 cluster [Ama17]. Different HPC clusters have such pricing scheme for their users, while other clusters are envisioned to deploy such scheme in their systems [MHL⁺13].

The workload trace we collected does not contain any power usage information. Therefore, we collected and used power-related information from literature for this study. We use the power data at different frequencies for six applications from an existing study [ABB⁺14]. The details of the applications and related power data are described in Section 4.3.1. Measurements were collected when running these applications with different CPU frequencies, ranging from 1.2 GHz to 2.4 GHz at

0.2 GHz intervals and 2.7 GHz. That is, the minimum and maximum frequency are 1.2 GHz and 2.7 GHz, respectively. The peak power of the processors was set to 240 W (determined from the power consumption of the six HPC applications when running at the maximum frequency).

5.5.2 Energy Reduction and Utility

We first demonstrate how different HPC users participate in demand response program and their rewards for such participation. Fig. 5.3 presents results for energy reduction and users' and operator's utilities. Fig. 5.3(a) shows energy reduction amount by all jobs of different types. As evident from the figure, different jobs participate in energy reduction at varying level to enable demand response participation. Fig. 5.3(b) shows rewards awarded to all the jobs of various job types. During high electricity price periods, the users contribute more to energy reduction and therefore are rewarded more. Fig. 5.3(c) shows the inconvenience cost for each job in different job types. As can be seen in the figure, lower-type jobs (i.e., with lower θ_i values) have higher inconvenience value than higher-type jobs, a requirement for designing feasible contracts. Fig. 5.3(d) shows utility of each job for different job types. As can be seen in the figure, different job types have non-negative utility (Type#1 jobs have 0 utility). Fig. 5.3(e) shows the operator's utility throughout the time period. As evident from the figure, the operator will have positive utility at different time period. Therefore, participation in demand response program will be beneficial to the operator, after rewarding the users for their participations in the program.

Fig. 5.4 shows the jobs types' utilities when the same job type is offered different contract options. In the figure, we show the utilities of each job type, when selecting all the contracts offered by the operator. As clearly evident from the figure, each

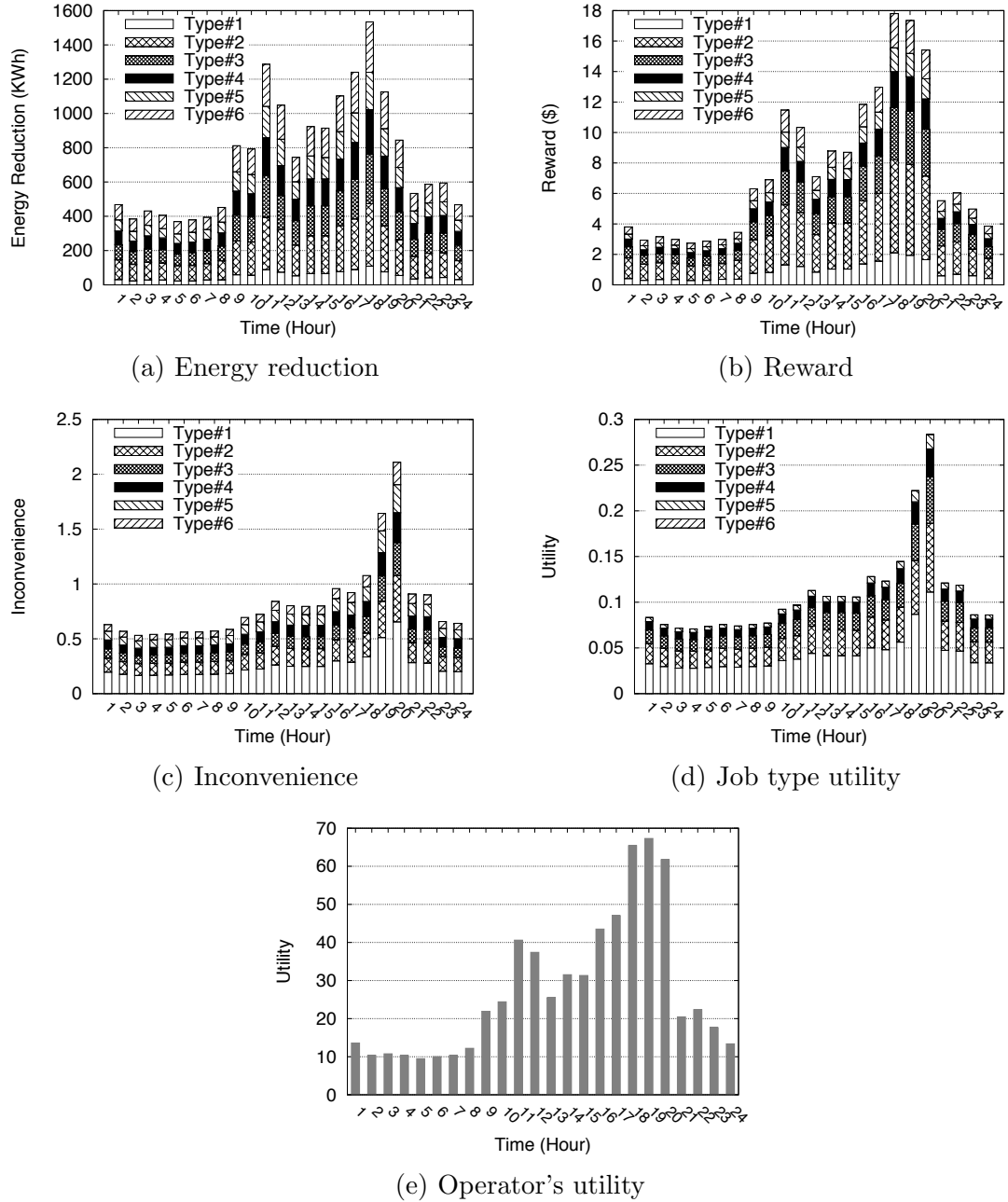


Figure 5.3: Energy reduction and utility.

type of user will achieve maximum utility when the user selects the contract that is intended to its own type. Hence, if operator design the contracts using our designed mechanism, the hidden types of the users will be revealed to the operator. Since

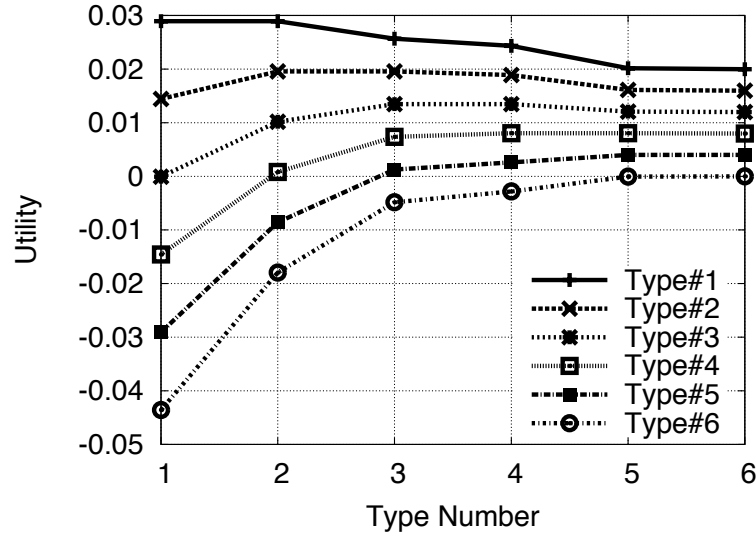


Figure 5.4: Incentive compatibility constraint.

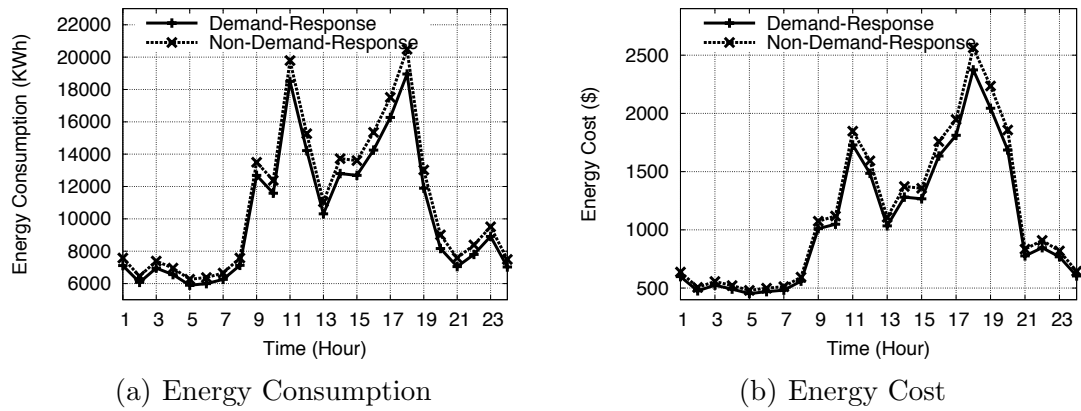


Figure 5.5: Benchmark comparison.

each type of users would be willing to accept the contract designed for own type, incentive compatibility constraint is satisfied using our designed mechanism.

5.5.3 Benchmark Comparison

To evaluate the performance of our resource provisioning algorithm for demand response, we compare it with another policy that does not consider demand response. We denote our policy for resource allocation as *Demand-Response* and the other

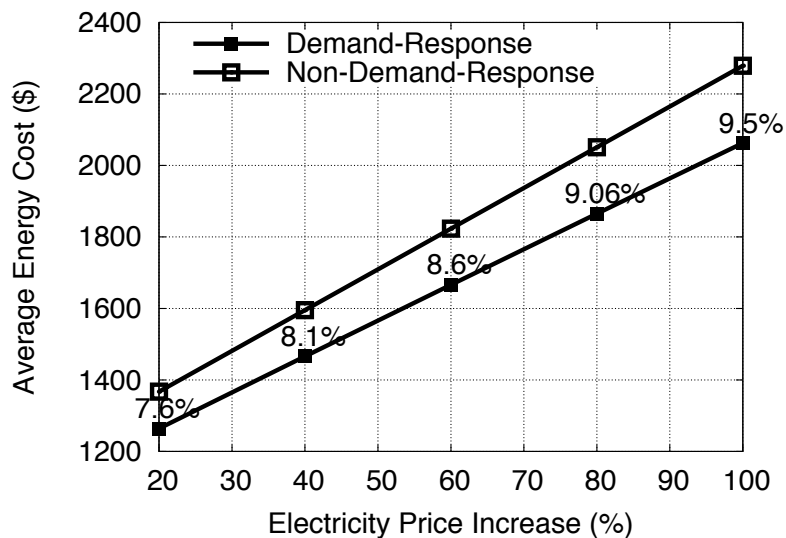


Figure 5.6: Change in electricity price.

policy which does not consider demand response participation as *Non-Demand-Response*. Fig. 5.5 compares the two benchmarks in terms of energy reduction and energy cost during different time periods. Fig. 5.5(a) compares two benchmarks for energy consumption. As evident from the figure, Demand-Response achieves lower energy consumption throughout the time period. The energy consumption reduction is more pronounced during the peak electricity period (for example, energy consumption is reduced as much as 10% during hour#20 when the electricity price is highest for the day). Fig. 5.5(b) compares the energy cost for both the benchmarks. Demand-Response incurs lower energy cost throughout the time period compared to Non-Demand-Response.

Fig. 5.6 shows effect of increase in hourly electricity price. We change the original electricity price from 20% to 100% and compare the two benchmarks. Average energy cost in the figure denotes the hourly average energy cost for the entire day. As evident from the figure, Demand-Response incurs lower energy cost compared to Non-Demand-Response for different electricity price values. Moreover, the per-

centage savings in energy cost for Demand-Response compared to Non-Demand-Response increases with higher electricity price. This is because Demand-Response captures the variation in time-varying electricity price, and therefore is able to reduce the energy cost more significantly when the electricity price is higher.

5.6 Summary

In this chapter, we present an economic demand participation model for HPC systems. Based on contract theory, we design a rewarding mechanism for HPC users to enable their willing participation in demand response program, so that HPC operators and HPC users can jointly reduce the energy cost. Our proposed demand response model benefits all the involved participant: HPC system can reduce energy cost, HPC users can earn reward from such participation and energy service provider can achieve grid stability through overall energy reduction. We performed analytical study to show that our proposed model can satisfy the necessary properties of contract theory. We also present simulation study to show that our model preserves the necessary properties of contract theory and is effective than other existing approach.

CHAPTER 6

CONCLUSIONS

In this chapter, we present a brief summary of this dissertation and provide some future directions the research could be taken.

6.1 Summary

This dissertation addresses massive energy consumption of HPC system through presenting detailed power and performance demand response participation models.

1. In the first part of the dissertation, we present how to accurately model HPC applications' performance running on large-scale HPC systems containing hundreds of thousands of nodes. More specifically, we do the following:
 - We present interconnection network models for performance prediction of large-scale scientific applications in high-performance architectures. Specifically, we present interconnect models for Cray's Gemini 3-D torus, IBM's Blue Gene/Q 5-D torus, Cray's Aries dragonfly, and Infiniband's fat-tree network. Our interconnection network models have been fully integrated with an implementation of the MPI model, which mimics all common MPI commands, including various send and receive functions, as well as collective operations. The MPI model can achieve packet-level accuracy at the target platforms.
 - We present extensive validation studies of our MPI and interconnect models, including a trace-based study using data obtained from executing real-life computational physics code on an existing high-performance

computing platform. We also present a performance study of a parallel computational physics application to show that our model can accurately predict the parallel behavior of large-scale applications. All the results show that our models can provide sufficient accuracy. Moreover, our study of the parallel performance of integrated models on large-scale HPC platforms show good parallel scaling performance.

2. In the next part of the dissertation, we present power and performance models to enable HPC system's demand response participation. To this end, we develop emergency demand-response models for HPC systems.

- We explore the opportunity of HPC emergency demand response by proposing a new HPC job scheduling and resource provisioning model. More specifically, the proposed model applies power-bound energy-conservation job scheduling and resource allocation (using DVFS) during the critical demand-response events, while maintaining the traditional performance-optimized job scheduling during the normal period.
- We implemented the proposed method in a simulator and compare it with the traditional scheduling approach. Using trace-driven simulation, we demonstrate that HPC demand response is a viable approach toward power stability and energy savings with only marginal increase in the jobs' execution time.
- We propose an emergency demand-response model exploiting both power capping of HPC systems and node scaling of HPC applications. We present power and performance prediction models for HPC systems with only power capping, upon which we propose our demand-response model. We validated the models with real-life measurements of application char-

acteristics. Next, we present models to predict energy-to-solution for HPC applications with different numbers of nodes and power-capping values, and we validate the models. Finally, we demonstrate the effectiveness of our proposed emergency demand-response model using real-life measurements and trace data.

3. Finally, we address HPC system’s significant energy cost reduction through proposing an economic demand response participation model. Specifically, we:

- Propose an economic demand response model to allow both HPC operators and HPC users to jointly reduce the energy cost. We apply the contract theory to prepare a rewarding scheme to be offered by the HPC operator to the HPC users to encourage users’ willing participation in economic demand response.
- Performed both analytical and simulation studies of our proposed approach. We show that our contract-based demand response model is both feasible (in ensuring both individual rationality and incentive compatibility) and optimal. Through trace-based simulation, we demonstrate that our model preserves the necessary properties of contract theory and is effective compared to other existing approach.

6.2 Future Directions

In the first part of the dissertation, we discuss how to perform rapid performance prediction modeling in large-scale HPC systems. A possible future extension to this work would be to include more recent interconnect topologies (e.g., Slim Fly), and

integrate the interconnect model with detailed system models, including processors, cache, and file systems. Another possible future work is to translate all our interconnect models to Simian Lua. Such translation would help us to study the parallel performance of our integrated models on large-scale platforms and also to compare with performance of current Python-based interconnect model implementation.

In the second part of the dissertation, we present job scheduling and resource allocation algorithms to reduce energy consumption in HPC systems during demand response periods. There are many opportunities to extend our proposed demand-response model for HPC systems.

1. Our job scheduler uses linear regression models for predicting the job’s power consumption and execution time. Regression models require empirical measurements of each application’s power consumption and execution time with different CPU frequencies. In practice, this may not be readily available *a priori*, especially for unknown applications, at the job’s submission. More advanced models may be applied in this case. For example, inference and learning techniques (such as artificial neural networks) may be employed to help in these scenarios [IdSSM05, LBdS⁺07].
2. For all types of power and performance models, prediction errors may occur. A future work would be to investigate the effect of prediction errors on the job scheduling and resource provisioning, and also to consider more adaptive methods in power and performance prediction models.
3. Our demand response job scheduler considers only processor-level frequency scaling. Some current HPC systems can support only machine-level DVFS. In future systems, frequency scaling may also be available more commonly at individual cores. We have not yet investigated these options. Moreover,

frequency scaling and power capping are not the only control knobs for energy saving. More adaptable power-aware scheduling policies and resource provisioning algorithms can be considered in addition to frequency scaling and power capping.

4. There are techniques that consider power consumption as a function of computation and communication at the system level and/or at the sub-job level. For example, Adagio [RLDS⁺09] considers variations in the energy consumption during computation and communication phases of the applications. Also, SERA-IO [GFS12] replaces the traditional I/O middleware with an energy-conscious mechanism that combines with DVFS for power-performance scheduling. These techniques can be considered in our future studies to allow more effective demand response.

In the final part of the dissertation, we address HPC system’s economic demand response participation and present a contract-based demand-response model. In our study, we assume that the HPC operator have complete information on the job types (e.g., which jobs belong to what type). In practice, the operator may not directly have such information. Updating the statistical information on the jobs’ distribution to different types (e.g., through “online learning”), can be an interesting future work. We performed a brief theoretical study of the continuum types of contracts. However, a more rigorous study of continuum types of contracts, along with implementation of such types in practice, can be another future direction to explore.

BIBLIOGRAPHY

- [A⁺14] T Arber et al. Epoch: Extendable pic open collaboration. <http://www.ccpp.ac.uk/codes.html>, 2014.
- [ABB⁺14] Axel Auweter, Arndt Bode, Matthias Brehm, Luigi Brochard, Nicolay Hammer, Herbert Huber, Raj Panda, Francois Thomas, and Torsten Wilde. A case study of energy aware scheduling on SuperMUC. In *International Supercomputing Conference*, pages 394–409. Springer, 2014.
- [ABD⁺15] R. E. Alcouffe, R. S. Baker, J. D. Dahl, E. D. Fichtl, S. A. Turner, R. C. Ward, and R. J. Zerr. Partisn: a time-dependent, parallel neutral particle transport code system. LANL, LA-UR-08-07258, last revised, December 2015, 2015.
- [ABF⁺10] Laksono Adhianto, Sinchan Banerjee, Mike Fagan, Mark Krentel, Gabriel Marin, John Mellor-Crummey, and Nathan R Tallent. Hpc-toolkit: Tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience*, 22(6):685–701, 2010.
- [AFKR12] Bob Alverson, Edwin Froese, Larry Kaplan, and Duncan Roweth. Cray xc series network. <http://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf>, 2012.
- [AFLV08] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, pages 63–74. ACM, 2008.
- [AJB⁺15] Bilge Acun, Nikhil Jain, Abhinav Bhatele, Misbah Mubarak, Christopher D Carothers, and Laxmikant V Kale. Preliminary evaluation of a parallel trace replay tool for HPC network simulations. In *Euro-Par 2015: Parallel Processing Workshops*, pages 417–429, 2015.
- [Ake70] George Akerlof. The market for lemons. *Quarterly Journal of Economics*, 89:488–500, 1970.
- [AKPJ09] Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraj K Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 33–42. IEEE, 2009.

- [ALE02] Todd Austin, Eric Larson, and Dan Ernst. SimpleScalar: An infrastructure for computer system modeling. *Computer*, 35(2):59–67, 2002.
- [ALEZ16] K. Ahmed, J. Liu, S. Eidenbenz, and J. Zerr. Scalable interconnection network models for rapid performance prediction of HPC applications. In *2016 IEEE 18th International Conference on High Performance Computing and Communications (HPCC)*, pages 1069–1078, Dec 2016.
- [ALSJ13] Jung Ho Ahn, Sheng Li, O Seongil, and Norman P Jouppi. Mcsima+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling. In *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, pages 74–85, 2013.
- [Alv12] Robert Alverson. Cray high speed networking. In *Proceedings of the 20th Annual Symposium on High-Performance Interconnects (HOTI)*, 2012.
- [Ama17] Amazon. Amazon EC2 reserved instances pricing. <https://aws.amazon.com/ec2/pricing/reserved-instances/pricing/>, 2017.
- [AOL⁺16] Kishwar Ahmed, Mohammad Obaida, Jason Liu, Stephan Eidenbenz, Nandakishore Santhi, and Guillaume Chapuis. An integrated interconnection network model for large-scale performance prediction. In *Proceedings of the 2016 annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation*, pages 177–187. ACM, 2016.
- [APM⁺12] Pablo Abad, Pablo Prieto, Lucia G Menezo, Adrián Colaso, Valentin Puente, and Jose-Angel Gregorio. Topaz: An open-source interconnection network simulator for chip multiprocessors and supercomputers. In *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, pages 99–106. IEEE, 2012.
- [AR13] Ehsan K Ardestani and Jose Renau. Esesc: A fast multicore simulator using time-based sampling. In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pages 448–459. IEEE, 2013.
- [Arc17] Arch Linux. CPUs frequency scaling. https://wiki.archlinux.org/index.php/CPU_frequency_scaling, 2017.

- [ARK10] Robert Alverson, Duncan Roweth, and Larry Kaplan. The Gemini system interconnect. In *2010 18th IEEE Symposium on High Performance Interconnects*, pages 83–87. IEEE, 2010.
- [AS11] David Aikema and Rob Simmonds. Electrical cost savings and clean energy usage potential for hpc workloads. In *Sustainable Systems and Technology (ISSST), 2011 IEEE International Symposium on*, pages 1–6. IEEE, 2011.
- [ASC17] ASCR. ExaCT: Exascale Simulation of Combustion in Turbulence. <https://crd.lbl.gov/projects/combustion-codesign/>, 2017.
- [AW14] Brian Austin and Nicholas J Wright. Measurement and interpretation of microbenchmark and application energy use on the Cray XC30. In *Proceedings of the 2nd International Workshop on Energy Efficient Supercomputing*, pages 51–59. IEEE Press, 2014.
- [BBB⁺91] David H Bailey, Eric Barszcz, John T Barton, David S Browning, Robert L Carter, Leonardo Dagum, Rod A Fatoohi, Paul O Frederickson, Thomas A Lasinski, Rob S Schreiber, et al. The NAS parallel benchmarks summary and preliminary results. In *Supercomputing*, 1991.
- [BBB⁺11] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.
- [BCPC16] Enes Bilgin, Michael C Caramanis, Ioannis Ch Paschalidis, and Christos G Cassandras. Provision of regulation service by smart buildings. *IEEE Transactions on Smart Grid*, 7(3):1683–1693, 2016.
- [BCY13] Reuben D Budiardja, Lonnie Crosby, and Haihang You. Effect of rank placement on cray xc30 communication cost. In *The Cray User Group Meeting*, 2013.
- [BD05] Patrick Bolton and Mathias Dewatripont. *Contract theory*. MIT press, 2005.
- [BDH⁺06] Nathan L Binkert, Ronald G Dreslinski, Lisa R Hsu, Kevin T Lim, Ali G Saidi, and Steven K Reinhardt. The m5 simulator: Modeling networked systems. *IEEE Micro*, 26(4):52–60, 2006.

- [BE11] Swen Böhm and Christian Engelmann. xsim: The extreme-scale simulator. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 280–286, 2011.
- [BGA⁺15] Natalie Bates, Girish Ghatikar, Ghaleb Abdulla, Gregory A Koenig, Sridutt Bhalachandra, Mehdi Sheikhalishahi, Tapasya Patki, Barry Rountree, and Stephen Poole. Electrical grid and supercomputing centers: an investigative analysis of emerging opportunities and challenges. *Informatik-Spektrum*, 38(2):111–127, 2015.
- [BGK⁺16] Tekin Bicer, Doga Gürsoy, Rajkumar Kettimuthu, Francesco De Carlo, and Ian T Foster. Optimization of tomographic reconstruction workflows on geographically distributed resources. *Journal of Synchrotron Radiation*, 23(4), 2016.
- [BHC⁺16] Wenlei Bao, Changwan Hong, Sudheer Chunduri, Sriram Krishnamoorthy, Louis-Noël Pouchet, Fabrice Rastello, and P Sadayappan. Static and dynamic frequency scaling on multicore CPUs. *ACM Transactions on Architecture and Code Optimization (TACO)*, 13(4):51, 2016.
- [BJL⁺15] Abhinav Bhatele, Nikhil Jain, Yarden Livnat, Valerio Pascucci, and Peer-Timo Bremer. Simulating and visualizing traffic on the dragonfly network. https://cug.org/proceedings/cug2016_proceedings/includes/files/pap155.pdf, 2015.
- [Bog14] Bartosz Bogdaski. *Optimized Routing for Fat-tree Topologies*. PhD thesis, University of Oslo, Oslo, Norway, 2014.
- [Can69] L. E. Cannon. *A Cellular Computer to Implement the Kalman Filter Algorithm*. PhD thesis, Montana State University, Bozeman, MT, 1969.
- [CBP02] Christopher D Carothers, David Bauer, and Shawn Pearce. Ross: A high-performance, low-memory, modular time warp system. *Journal of Parallel and Distributed Computing*, 62(11):1648–1669, 2002.
- [CBS⁺12] Niladrish Chatterjee, Rajeev Balasubramonian, Manjunath Shevgoor, S Pugsley, A Udipi, Ali Shafiee, Kshitij Sudan, Manu Awasthi, and Zeshan Chishti. USIMM: The utah simulated memory module. *University of Utah, Tech. Rep*, 2012.

- [CEH⁺11] Dong Chen, Noel A Eisley, Philip Heidelberger, Robert M Senger, Yutaka Sugawara, Sameer Kumar, Valentina Salapura, David L Satterfield, Burkhard Steinmacher-Burow, and Jeffrey J Parker. The ibm blue gene/q interconnection network and message unit. In *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–10. IEEE, 2011.
- [CEH⁺12] Dong Chen, Noel Eisley, Philip Heidelberger, Robert Senger, Yutaka Sugawara, Sameer Kumar, Valentina Salapura, David Satterfield, Burkhard Steinmacher-Burow, and Jeffrey Parker. The ibm blue gene/q interconnection fabric. *IEEE Micro*, 32(1):32–43, 2012.
- [CESP15] G. Chapuis, S. Eidenbenz, N. Santhi, and E. J. Park. Simian integrated framework for parallel discrete event simulation on GPUs. In *2015 Winter Simulation Conference (WSC)*, pages 1127–1138, Dec 2015.
- [CGL⁺14] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917, 2014.
- [CHK16] Thang Cao, Yuan He, and Masaaki Kondo. Demand-aware power management for power-constrained HPC systems. In *Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on*, pages 21–31. IEEE, 2016.
- [CLL⁺11] Jason Cope, Ning Liu, Sam Lang, Phil Carns, Chris Carothers, and Robert Ross. Codes: Enabling co-design of multilayer exascale storage architectures. In *Proceedings of the Workshop on Emerging Supercomputing Technologies*, pages 303–312, 2011.
- [CNEP16] Guillaume Chapuis, David Nicholaeff, Stephan Eidenbenz, and Robert S Pavel. Predicting performance of smoothed particle hydrodynamics codes at large scales. In *Winter Simulation Conference (WSC), 2016*, pages 1825–1835. IEEE, 2016.
- [Cou16] Rachel Courtland. China inches toward the exascale[news]. *IEEE Spectrum*, 53(8):14–15, 2016.
- [CVB15] Cristóbal Camarero, Enrique Vallejo, and Ramón Beivide. Topological characterization of hamming and dragonfly networks and its im-

- plications on routing. *ACM Transactions on Architecture and Code Optimization (TACO)*, 11(4):39, 2015.
- [CVM⁺15] Matteo Chiesi, Luca Vanzolini, Claudio Mucci, Eleonora Franchi Scarselli, and Roberto Guerrieri. Power-aware job scheduling on heterogeneous multicore architectures. *IEEE Transactions on Parallel and Distributed Systems*, 26(3):868–877, 2015.
- [DBM⁺11] Jack Dongarra, Pete Beckman, Terry Moore, Patrick Aerts, Giovanni Aloisio, Jean-Claude Andre, David Barkai, Jean-Yves Berthou, Taisuke Boku, Bertrand Braunschweig, et al. The international exascale software project roadmap. *International Journal of High Performance Computing Applications*, 25(1):3–60, 2011.
- [Dep14] Department of Energy. Department of energy awards \$425 million for next generation supercomputing technologies, 2014.
- [DGML13] Mohammed El Mehdi Diouri, Olivier Glück, J-C Mignot, and Laurent Lefèvre. Energy estimation for mpi broadcasting algorithms in large scale hpc systems. In *Proceedings of the 20th European MPI Users’ Group Meeting*, pages 111–116, 2013.
- [DKS⁺14] Lingjie Duan, Takeshi Kubo, Kohei Sugiyama, Jianwei Huang, Teruyuki Hasegawa, and Jean Walrand. Motivating smartphone collaboration in data acquisition and distributed computing. *IEEE Transactions on Mobile Computing*, 2014.
- [DSRI13] Kefeng Deng, Junqiang Song, Kaijun Ren, and Alexandru Iosup. Exploring portfolio scheduling for long-term execution of scientific workloads in IaaS clouds. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 55. ACM, 2013.
- [DVMJC13] Karel De Vogeleer, Gerard Memmi, Pierre Jouvelot, and Fabien Coelho. The energy/frequency convexity rule: Modeling and experimental validation on mobile devices. In *International Conference on Parallel Processing and Applied Mathematics*, pages 793–803. Springer Berlin Heidelberg, 2013.
- [ECLV12] Maja Etinski, Julita Corbalan, Jesus Labarta, and Mateo Valero. Parallel job scheduling for power constrained HPC systems. *Parallel Computing*, 38(12):615–630, 2012.

- [EL10] Christian Engelmann and Frank Lauer. Facilitating co-design for extreme-scale systems through lightweight simulation. In *Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), 2010 IEEE International Conference on*, pages 1–8, 2010.
- [Ene14] EnerNOC. IEA workshop: Demand response. https://www.iea.org/media/workshops/2014/esapworkshopii/Jeff_Renaud.pdf, 2014.
- [Eng14] Christian Engelmann. Scaling to a million cores and beyond: Using light-weight simulation to understand the challenges ahead on the road to exascale. *Future Generation Computer Systems*, 30:59–65, 2014.
- [FBCM14] Mark R Fahey, Reuben Budiardja, Lonnie Crosby, and Stephen McNally. Deploying darter-a cray xc30 system. In *International Supercomputing Conference*, pages 430–439. Springer, 2014.
- [FBR⁺12] Greg Faanes, Abdulla Bataineh, Duncan Roweth, Edwin Froese, Bob Alverson, Tim Johnson, Joe Kopnick, Mike Higgins, James Reinhard, et al. Cray cascade: a scalable hpc system based on a dragonfly network. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 103. IEEE Computer Society Press, 2012.
- [FDK⁺11] Christian Feichtinger, Stefan Donath, Harald Köstler, Jan Götz, and Ulrich Rude. Walberla: HPC software design for computational engineering simulations. *Journal of Computational Science*, 2(2):105–112, 2011.
- [Fed16] Federal Energy Regulatory Commission. Assessment of demand response and advanced metering. <https://www.ferc.gov/legal/staff-reports/2016/DR-AM-Report2016.pdf>, 2016.
- [FP16] Peter Fox-Penner. Why Apple is getting into the energy business. <https://hbr.org/2016/11/why-apple-is-getting-into-the-energy-business>, 2016.
- [FPK⁺05] Vincent W Freeh, Feng Pan, Nandini Kappiah, David K Lowenthal, and Robert Springer. Exploring the energy-time tradeoff in MPI programs on a power-scalable cluster. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 10–pp. IEEE, 2005.

- [FTK14] Dror G. Feitelson, Dan Tsafir, and David Krakov. Experience with using the parallel workloads archive. *Journal of Parallel and Distributed Computing*, 74(10):2967 – 2982, 2014.
- [GAO14] QIANWEN GAO. *Investigation of power capping techniques for better computing energy efficiency*. PhD thesis, Politecnico di Milano, 2014.
- [GBB⁺09] Paolo Giannozzi, Stefano Baroni, Nicola Bonini, Matteo Calandra, Roberto Car, Carlo Cavazzoni, Davide Ceresoli, Guido L Chiarotti, Matteo Cococcioni, Ismaila Dabo, et al. Quantum espresso: a modular and open-source software project for quantum simulations of materials. *Journal of physics: Condensed matter*, 21(39):395502, 2009.
- [GCWL13] Yang Gao, Yan Chen, Chih-Yu Wang, and KJ Ray Liu. A contract-based approach for ancillary services in v2g networks: Optimality and learning. In *INFOCOM, 2013 Proceedings IEEE*, pages 1151–1159. IEEE, 2013.
- [GFFC07] Rong Ge, Xizhou Feng, Wu-chun Feng, and Kirk W Cameron. CPU MISER: A performance-directed, run-time system for power-aware clusters. In *Parallel Processing, 2007. ICPP 2007. International Conference on*, pages 18–18. IEEE, 2007.
- [GFS12] R. Ge, X. Feng, and X. H. Sun. SERA-IO: Integrating energy consciousness into parallel i/o middleware. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2012)*, pages 204–211, May 2012.
- [GGMP12] Girish Ghatikar, Venkata Ganti, Nance Matson, and Mary Ann Piette. Demand response opportunities and enabling technologies for data centers: Findings from field studies. Technical report, Lawrence Berkeley National Lab, 2012.
- [GGRT15] Yiannis Georgiou, David Glesser, Krzysztof Rzdca, and Denis Trystram. A scheduler-level incentive mechanism for energy efficiency in HPC. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pages 617–626. IEEE, 2015.
- [GL84] Roger Guesnerie and Jean-Jacques Laffont. A complete solution to a class of principal-agent problems with an application to the control of a self-managed firm. *Journal of Public Economics*, 25:329–369, 1984.

- [GR17] Al Geist and Daniel A Reed. A survey of high-performance computing scaling challenges. *The International Journal of High Performance Computing Applications*, 31(1):104–113, 2017.
- [GRP⁺13] Marc Gamell, Ivan Rodero, Manish Parashar, Janine C Bennett, Hemant Kolla, Jacqueline Chen, Peer-Timo Bremer, Aaditya G Landge, Attila Gyulassy, Patrick McCormick, et al. Exploring power behaviors and trade-offs of in-situ data analytics. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 77, 2013.
- [GWXZ11] Lin Gao, Xinbing Wang, Youyun Xu, and Qian Zhang. Spectrum trading in cognitive radio networks: A contract-theoretic modeling approach. *IEEE Journal on Selected Areas in Communications*, 2011.
- [HBH14] David G Holmberg, Steven T Bushby, and David B Hardin. Facility smart grid interface and a demand response conceptual model. Technical Report NIST Technical Note 1832, NIST, 2014.
- [HJB12] Ziaul Hasan, Abbas Jamalipour, and Vijay K Bhargava. Cooperative communication and relay selection under asymmetric information. In *Wireless Communications and Networking Conference (WCNC), 2012 IEEE*, pages 2373–2378. IEEE, 2012.
- [Hol16] Richard Holden. Contract Theory Notes. <http://research.economics.unsw.edu.au/richardholden/assets/2060-rh-2016-v2.pdf>, 2016.
- [How09] Paul G Howard. Six-core amd opteron processor istanbul. *White Paper, Microway Inc*, 2009.
- [HRT⁺12] Ming-yu Hsieh, Rolf Riesen, Kevin Thompson, William Song, and Arun Rodrigues. Sst: A scalable parallel framework for architecture-level performance, power, area and thermal simulation. *The Computer Journal*, 55(2):181–191, 2012.
- [IDR16] IDRIS. Turing, IBM Blue Gene/Q: Hardware configuration. <http://www.idris.fr/eng/turing/hw-turing-eng.html>, 2016.
- [IdSSM05] Engin Ipek, Bronis R. de Supinski, Martin Schulz, and Sally A. McKee. An approach to performance prediction for parallel applications. *Euro-Par 2005 Parallel Processing*, pages 627–628, 2005.

- [Ill12] Illinois Institute of Technology. CQsim. <http://bluesky.cs.iit.edu/cqsim/>, 2012.
- [Int14] Intel Corporation. Intel 64 and ia-32 architectures software developers manual. *Combined Volumes: 1, 2A, 2B, 2C, 3A, 3B and 3C*, 2014.
- [JAC⁺12] Curtis L Janssen, Helgi Adalsteinsson, Scott Cranford, Joseph P Kenny, Ali Pinar, David A Evensky, and Jackson Mayo. A simulator for large-scale parallel computer architectures. *Technology Integration Advancements in Distributed Systems and Computing*, 179, 2012.
- [Jan10] Jeff Janzen. The micron system-power calculator. <https://www.micron.com/support/tools-and-utilities/power-calc>, 2010.
- [JBW⁺16] Nikhil Jain, Abhinav Bhatele, Sam White, Todd Gamblin, and Laxmikant V. Kale. Evaluating HPC networks via simulation of parallel workloads. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '16*, pages 14:1–14:12, Piscataway, NJ, USA, 2016. IEEE Press.
- [JE11] Ian S Jones and Christian Engelmann. Simulation of large-scale hpc architectures. In *Parallel Processing Workshops (ICPPW), 2011 40th International Conference on*, pages 447–456, 2011.
- [JYE12] Min Kyu Jeong, Doe Hyun Yoon, and Mattan Erez. Drsim: A platform for flexible DRAM system research. <http://lph.ece.utexas.edu/public/DrSim>, 2012.
- [KBC⁺11] Rico Knapper, Benjamin Blau, Tobias Conte, Anca Sailer, Andrzej Kochut, and Ajay Mohindra. Efficient contracting in cloud service markets with asymmetric information—a screening approach. In *Commerce and Enterprise Computing (CEC)*, 2011.
- [KBD⁺08] Andreas Knüpfer, Holger Brunst, Jens Doleschal, Matthias Jurenz, Matthias Lieber, Holger Mickler, Matthias S Müller, and Wolfgang E Nagel. The vampir performance analysis tool-set. In *Tools for High Performance Computing*, pages 139–155. Springer, 2008.
- [KBK⁺13] Ian Karlin, Abhinav Bhatele, Jeff Keasler, Bradford L Chamberlain, Johanne Cohen, Zachary DeVito, Rakibul Haque, Daniel Laney, Edward Luke, Felix Wang, et al. Exploring traditional and emerging

- parallel programming models using a proxy application. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 919–932. IEEE, 2013.
- [KBVH14] Darren J Kerbyson, Kevin J Barker, Abhinav Vishnu, and Adolfo Hoisie. A performance comparison of current HPC systems: Blue Gene/Q, Cray XE6 and InfiniBand systems. *Future Generation Computer Systems*, 30:291–304, 2014.
- [KC15] Angeliki V Kordali and Panayotis G Cottis. A contract-based spectrum trading scheme for cognitive radio networks enabling hybrid access. *IEEE Access*, 3:1531–1540, 2015.
- [KDSA08] John Kim, Wiliam J Dally, Steve Scott, and Dennis Abts. Technology-driven, highly-scalable dragonfly topology. In *ACM SIGARCH Computer Architecture News*, pages 77–88. IEEE Computer Society, 2008.
- [KK93] Laxmikant V Kale and Sanjeev Krishnan. Charm++: A portable concurrent object oriented system based on c++. In *ACM Sigplan Notices*, pages 91–108, 1993.
- [KLPS09] Andrew B Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. ORION 2.0: A fast and accurate noc power and area model for early-stage design space exploration. In *Proceedings of the conference on Design, Automation and Test in Europe*, pages 423–428, 2009.
- [KMY15] Kamran Kianfar, Ghasem Moslehi, and R Yahyapour. A novel meta-heuristic algorithm and utility function for qos based scheduling in user-centric grid systems. *The Journal of Supercomputing*, 71(3):1143–1162, 2015.
- [KPC⁺08] Martin Käser, Josep de al Puente, Cristóbal Castro, Verena Hermann, and Michael Dumbser. Seismic wave field modelling using high performance computing. In *SEG Technical Program Expanded Abstracts 2008*, pages 2884–2888. Society of Exploration Geophysicists, 2008.
- [KRY12] Chad D Kersey, Arun Rodrigues, and Sudhakar Yalamanchili. A universal parallel front-end for execution driven microarchitecture simulation. In *Proceedings of the 2012 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, pages 25–32, 2012.

- [KT11] Volodymyr Kindratenko and Pedro Trancoso. Trends in high-performance computing. *Computing in Science & Engineering*, 13(3):92–95, 2011.
- [KYM15] Yoongu Kim, Weikun Yang, and Onur Mutlu. Ramulator: A fast and extensible dram simulator. *IEEE Computer Architecture Letters*, 2015.
- [Lab15] Sandia National Laboratories. Structural simulation toolkit (sst) dumpi trace library. <https://github.com/sstsimulator/sst-dumpi>, Accessed December 2015.
- [LAS⁺09] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 469–480, 2009.
- [Law11] Lawrence Livermore National Lab. ExMatEx: Extreme Materials at Extreme Scale. <https://codesign.llnl.gov/exmatex.php>, 2011.
- [Law15] Lawrence Livermore National Lab. Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH). <https://codesign.llnl.gov/lulesh.php>, Accessed December 2015.
- [LBD⁺06] Piotr R Luszczek, David H Bailey, Jack J Dongarra, Jeremy Kepner, Robert F Lucas, Rolf Rabenseifner, and Daisuke Takahashi. The hpc challenge (hpcc) benchmark suite. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 213. Citeseer, 2006.
- [LBdS⁺07] Benjamin C Lee, David M Brooks, Bronis R de Supinski, Martin Schulz, Karan Singh, and Sally A McKee. Methods of inference and learning for performance modeling of parallel applications. In *Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 249–258. ACM, 2007.
- [LC11] Ning Liu and Christopher D Carothers. Modeling billion-node torus networks using massively parallel discrete-event simulation. In *Proceedings of the 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation*, pages 1–8, 2011.

- [LCC⁺12] Ning Liu, Christopher Carothers, Jason Cope, Philip Carns, and Robert Ross. Model and simulation of exascale communication networks. *Journal of Simulation*, 6(4):227–236, 2012.
- [LCD⁺16] Yanpei Liu, Guilherme Cox, Qingyuan Deng, Stark C Draper, and Ricardo Bianchini. FastCap: An efficient and fair algorithm for power capping in many-core systems. In *ISPASS*, 2016.
- [LCH04] Xuan-Yi Lin, Yeh-Ching Chung, and Tai-Yi Huang. A multiple lid routing scheme for fat-tree-based infiniband networks. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 11. IEEE, 2004.
- [LCM⁺05] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *Acm sigplan notices*, volume 40, pages 190–200, 2005.
- [LDKP15] Akhil Langer, Harshit Dokania, Laxmikant V Kalé, and Udatta S Palekar. Analyzing energy-time tradeoff in power overprovisioned hpc data centers. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, pages 849–854. IEEE, 2015.
- [LdVW⁺12] Pierre-François Lavallée, Guillaume Colin de Verdiere, Philippe Wautelet, Dimitri Lecas, and Jean-Michel Dupays. Porting and optimizing HYDRO to new platforms and programming paradigms-lessons learnt. *Technical report, PRACE*, 2012.
- [LHSJ15] Ning Liu, Adnan Haider, Xian-He Sun, and Dong Jin. Fattreesim: Modeling large-scale fat-tree networks for hpc systems and data centers using parallel and discrete event simulation. In *Proceedings of the 3rd ACM Conference on SIGSIM-Principles of Advanced Discrete Simulation*, pages 199–210, 2015.
- [LLRL12] Jie Li, Zuyi Li, Kui Ren, and Xue Liu. Towards optimal electric demand management for internet data centers. *IEEE Transactions on Smart Grid*, 3(1):183–192, 2012.
- [LM06] Jian Li and Jose F Martinez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *High-Performance*

Computer Architecture, 2006. The Twelfth International Symposium on, pages 77–87. IEEE, 2006.

- [LM09] Jean-Jacques Laffont and David Martimort. *The theory of incentives: the principal-agent model*. Princeton university press, 2009.
- [LSSS16] Gary Lawson, Vaibhav Sundriyal, Masha Sosonkina, and Yuzhong Shen. Runtime power limiting of parallel applications on Intel Xeon Phi processors. In *Proceedings of the 4th International Workshop on Energy Efficient Supercomputing*, 2016.
- [LW15] Feng Liu and Jon B Weissman. Elastic job bundling: An adaptive resource request strategy for large-scale parallel applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 33. ACM, 2015.
- [LWC⁺13] Zhenhua Liu, Adam Wierman, Yuan Chen, Benjamin Razon, and Niangjun Chen. Data center demand response: Avoiding the coincident peak via workload shifting and local generation. *Performance Evaluation*, 2013.
- [Man14] Katrina Managan. Demand response: A market overview. Technical report, Institute for Building Efficiency, 2014.
- [McA17] James McAnany. 2016 demand response operations markets activity report: April 2017. <http://www.pjm.com/~media/markets-ops/dsr/2016-demand-response-activity-report.ashx>, 2017.
- [McC02] John D McCalpin. Stream benchmark. URL: <http://www.cs.virginia.edu/stream/stream2>, 2002.
- [MCE⁺02] Peter S Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hallberg, Johan Hogberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, 2002.
- [MCRC12] Misbah Mubarak, Christopher D Carothers, Robert Ross, and Philip Carns. Modeling a million-node dragonfly network using massively parallel discrete-event simulation. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion.*, pages 366–376. IEEE, 2012.

- [MCRC14] Misbah Mubarak, Christopher D Carothers, Robert B Ross, and Philip Carns. Using massively parallel simulation for mpi collective communication modeling in extreme-scale networks. In A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, editors, *Proceedings of the 2014 Winter Simulation Conference*, pages 3107–3118, Piscataway, New Jersey, 2014. Institute of Electrical and Electronics Engineers, Inc.
- [ME13] Hedin Marianne and Woods Eric. Market data: Demand response. residential, commercial, and industrial demand response participation and sites, load curtailment, and spending: Global and regional market sizing and forecasts. <https://www.pjm.com/-/media/markets-ops/dsr/2017-demand-response-activity-report.ashx>, 2013.
- [Mel16] Mellanox Technologies. Deploying hpc cluster with mellanox infiniband interconnect solutions, 2016.
- [MHL⁺13] Aniruddha Marathe, Rachel Harris, David K Lowenthal, Bronis R De Supinski, Barry Rountree, Martin Schulz, and Xin Yuan. A comparative study of high-performance computing on the cloud. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 239–250. ACM, 2013.
- [Mir71] James A Mirrlees. An exploration in the theory of optimum income taxation. *The review of economic studies*, pages 175–208, 1971.
- [Mis15] Mission Critical Power. Equinix in R&D phase of demand response experiments. <https://missioncriticalpower.uk/equinix-tests-demand-response/>, 2015.
- [MKK⁺10] Jason E Miller, Harshad Kasture, George Kurian, Charles Gruenwald III, Nathan Beckmann, Christopher Celio, Jonathan Eastep, and Anant Agarwal. Graphite: A distributed parallel simulator for multicores. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–12. IEEE, 2010.
- [MLY⁺16] Chuan Ma, Yuqing Li, Hui Yu, Xiaoying Gan, Xinbing Wang, Yong Ren, and Jun Jim Xu. Cooperative spectrum sharing in d2d-enabled cellular networks. *IEEE Transactions on Communications*, 64(10):4394–4408, 2016.

- [MSB⁺05] Milo MK Martin, Daniel J Sorin, Bradford M Beckmann, Michael R Marty, Min Xu, Alaa R Alameldeen, Kevin E Moore, Mark D Hill, and David A Wood. Multifacet’s general execution-driven multiprocessor simulator (gems) toolset. *ACM SIGARCH Computer Architecture News*, 33(4):92–99, 2005.
- [MV15] Prakash Murali and Sathish Vadhiyar. Metascheduling of HPC jobs in day-ahead electricity markets. In *2015 IEEE 22nd International Conference on High Performance Computing (HiPC)*, pages 386–395. IEEE, 2015.
- [Mye81] Roger B Myerson. Optimal auction design. *Mathematics of operations research*, 6(1):58–73, 1981.
- [NER15a] NERSC. Design forward characterization of DOE mini-apps. <http://portal.nersc.gov/project/CAL/doe-miniapps.htm>, 2015.
- [NER15b] NERSC. NERSC. Interconnect. <https://www.nersc.gov/users/computational-systems/retired-systems/hopper/configuration/interconnect/>, 2015.
- [Nic93] David M Nicol. The cost of conservative synchronization in parallel discrete event simulations. *Journal of the ACM (JACM)*, 40(2):304–333, 1993.
- [NJ14] Bahareh Nazari and Abbas Jamalipour. A contract-auction mechanism for multi-relay cooperative wireless networks. In *Vehicular Technology Conference (VTC Spring), 2014 IEEE 79th*, pages 1–5. IEEE, 2014.
- [Nvi15] Nvidia Corporation. Nvml api reference manual. https://docs.nvidia.com/deploy/pdf/NVML_API_Reference_Guide.pdf, 2015.
- [Oak14] Oak Ridge National Laboratory, Argonne National Laboratory, Lawrence Livermore National Laboratory. CORAL benchmark codes. <https://asc.llnl.gov/CORAL-benchmarks/>, 2014.
- [Off13] Office of Science Co-design Center. CESAR: Center for Exascale Simulation of Advanced Reactors. <https://cesar.mcs.anl.gov/>, 2013.
- [ORS⁺10] Catherine Mills Olschanowsky, Tajana Rosing, Allan Snaveley, Laura Carrington, Mustafa M Tikir, and Michael Laurenzano. Fine-grained

- energy consumption characterization and modeling. In *High Performance Computing Modernization Program Users Group Conference (HPCMP-UGC), 2010 DoD*, pages 487–497. IEEE, 2010.
- [PACG11] Avadh Patel, Furat Afram, Shunfei Chen, and Kanad Ghose. Marss: A full system simulator for multicore x86 cpus. In *Proceedings of the 48th Design Automation Conference*, pages 1050–1055. ACM, 2011.
- [PBG⁺16] Tapasya Patki, Natalie Bates, Girish Ghatikar, Anders Clausen, Sonja Klingert, Ghaleb Abdulla, and Mehdi Sheikhalishahi. Supercomputing centers and electricity service providers: a geographically distributed perspective on demand management in Europe and the United States. In *International Conference on High Performance Computing*, pages 243–260. Springer, 2016.
- [PCD⁺13] Antonio J Peña, Ralf G Correa Carvalho, James Dinan, Pavan Balaji, Rajeev Thakur, and William Gropp. Analysis of topology-dependent MPI performance on Gemini networks. In *Proceedings of the 20th European MPI Users’ Group Meeting*, pages 61–66. ACM, 2013.
- [Per10] Kalyan S Perumalla. $\mu\pi$: A scalable and transparent system for simulating mpi programs. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, page 62, 2010.
- [PJM14] PJM Interconnect. Demand response and why its important. <https://www.pjm.com/~media/markets-ops/dsr/end-use-customer-fact-sheet.ashx>, 2014.
- [PJM17] PJM Interconnection. Demand response strategy. <http://www.pjm.com/~media/library/reports-notice/demand-response/20170628-pjm-demand-response-strategy.ashx>, 2017.
- [PLR⁺13] Tapasya Patki, David K Lowenthal, Barry Rountree, Martin Schulz, and Bronis R De Supinski. Exploring hardware overprovisioning in power-constrained, high performance computing. In *SC*, 2013.
- [PLR⁺16] T. Patki, D. K. Lowenthal, B. L. Rountree, M. Schulz, and B. R. d. Supinski. Economic viability of hardware overprovisioning in power-constrained high performance computing. In *2016 4th International Workshop on Energy Efficient Supercomputing (E2SC)*, pages 8–15, Nov 2016.

- [PLS⁺15] Tapasya Patki, David K Lowenthal, Anjana Sasidharan, Matthias Maiterth, Barry L Rountree, Martin Schulz, and Bronis R de Supinski. Practical resource management in power-constrained, high performance computing. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pages 121–132. ACM, 2015.
- [Pow78] Michael JD Powell. A fast algorithm for nonlinearly constrained optimization calculations. In *Numerical analysis*, pages 144–157. Springer, 1978.
- [PP14] Kalyan S Perumalla and Alfred J Park. Simulating billion-task parallel programs. In *Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2014), International Symposium on*, pages 585–592, 2014.
- [PRA97] Vijay S Pai, Parthasarathy Ranganathan, and Sarita V Adve. Rsim: An execution-driven simulator for ilp-based shared-memory multiprocessors and uniprocessors. In *Proceedings of the Third Workshop on Computer Architecture Education*, volume 178. Citeseer, 1997.
- [PS06] Venkatesh Pallipadi and Alexey Starikovskiy. The ondemand governor. In *Proceedings of the Linux Symposium*, pages 215–230, 2006.
- [PSL10] Parallel Systems Lab. Python scheduler simulator. <https://code.google.com/archive/p/pyss/>, 2010.
- [PV97] Fabrizio Petrini and Marco Vanneschi. k-ary n-trees: High performance networks for massively parallel architectures. In *Parallel Processing Symposium, 1997. Proceedings., 11th International*, pages 87–93. IEEE, 1997.
- [PVB⁺13] Kevin Pedretti, Courtenay Vaughan, Richard Barrett, Karen Devine, and K Scott Hemmert. Using the Cray Gemini performance counters. *Proc Cray User Group (CUG)*, 2013.
- [RADS⁺12] Barry Rountree, Dong H Ahn, Bronis R De Supinski, David K Lowenthal, and Martin Schulz. Beyond dvfs: A first look at performance under a hardware-enforced power bound. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 947–953. IEEE, 2012.

- [RCBJ11] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. Dramsim2: A cycle accurate memory system simulator. *Computer Architecture Letters*, 10(1):16–19, 2011.
- [RHB⁺11] Arun F Rodrigues, K Scott Hemmert, Brian W Barrett, Chad Kersey, Ron Oldfield, Marlo Weston, R Risen, Jeanine Cook, Paul Rosenfeld, E CooperBalls, et al. The structural simulation toolkit. *ACM SIGMETRICS Performance Evaluation Review*, 38(4):37–42, 2011.
- [RLDS⁺09] Barry Rountree, David K Lownenthal, Bronis R De Supinski, Martin Schulz, Vincent W Freeh, and Tyler Bletsch. Adagio: making dvs practical for complex HPC applications. In *Proceedings of the 23rd international conference on Supercomputing*, pages 460–469. ACM, 2009.
- [SBK13] Shuaiwen Leon Song, Kevin Barker, and Darren Kerbyson. Unified performance and power modeling of scientific workloads. In *Proceedings of the 1st International Workshop on Energy Efficient Supercomputing*, page 4, 2013.
- [SBM09] Karan Singh, Major Bhadauria, and Sally A McKee. Real time power estimation and thread scheduling via performance counters. *ACM SIGARCH Computer Architecture News*, 37(2):46–55, 2009.
- [SCC⁺12] Cagri Sahin, Furkan Cayci, James Clause, Fouad Kiamilev, Lori Pollock, and Kristina Winbladh. Towards power reduction through improved software design. In *Energytech*, 2012.
- [SEL15] Nandakishore Santhi, Stephan Eidenbenz, and Jason Liu. The simian concept: parallel discrete event simulation with interpreted languages and just-in-time compilation. In *Proceedings of the 2015 Winter Simulation Conference*, pages 3013–3024. IEEE Press, 2015.
- [SK13] Daniel Sanchez and Christos Kozyrakis. Zsim: Fast and accurate microarchitectural simulation of thousand-core systems. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 475–486, 2013.
- [SKSS02] Srividya Srinivasan, Rajkumar Kettimuthu, Vijay Subramani, and P Sadayappan. Characterization of backfilling strategies for parallel job scheduling. In *Parallel Processing Workshops, 2002. Proceedings. International Conference on*, pages 514–519. IEEE, 2002.

- [SL14] Shang-Pin Sheng and Mingyan Liu. Profit incentive in trading nonexclusive access on a secondary spectrum market through contract design. *IEEE/ACM Transactions on Networking*, 22(4):1190–1203, 2014.
- [SLGK14] Osman Sarood, Akhil Langer, Abhishek Gupta, and Laxmikant Kale. Maximizing throughput of overprovisioned HPC data centers under a strict power budget. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 807–818. IEEE Press, 2014.
- [SLK⁺13] Osman Sarood, Akhil Langer, Laxmikant Kalé, Barry Rountree, and Bronis De Supinski. Optimizing power allocation to cpu and memory subsystems in overprovisioned hpc systems. In *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*, pages 1–8. IEEE, 2013.
- [SM06] Sameer S Shende and Allen D Malony. The tau parallel performance system. *International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.
- [SNHH14] Lingyang Song, Dusit Niyato, Zhu Han, and Ekram Hossain. Game-theoretic resource allocation methods for device-to-device communication. *IEEE Wireless Communications*, 2014.
- [Spr05] Volker Springel. The cosmological simulation code gadget-2. *Monthly notices of the royal astronomical society*, 364(4):1105–1134, 2005.
- [SV12] Kyle L Spafford and Jeffrey S Vetter. Aspen: A domain specific language for performance modeling. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 84, 2012.
- [SWA⁺15] Hayk Shoukourian, Torsten Wilde, Axel Auweter, Arndt Bode, and Daniele Tafani. Predicting energy consumption relevant indicators of strong scaling HPC applications for different compute resource configurations. In *Proceedings of the Symposium on High Performance Computing*, pages 115–126. Society for Computer Simulation International, 2015.
- [SWAB14] Hayk Shoukourian, Torsten Wilde, Axel Auweter, and Arndt Bode. Predicting the energy and power consumption of strong and weak

- scaling HPC applications. *Supercomputing frontiers and innovations*, 1(2):20–41, 2014.
- [SXZW16] Yuanyuan Shi, Bolun Xu, Baosen Zhang, and Di Wang. Leveraging energy storage to optimize data center electricity cost in emerging power markets. In *Proceedings of the Seventh International Conference on Future Energy Systems*, page 18. ACM, 2016.
- [TEL95] Dean M Tullsen, Susan J Eggers, and Henry M Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *ACM SIGARCH Computer Architecture News*, pages 392–403. ACM, 1995.
- [Tex17] Texas Advanced Computing Center. Stampede user guide. <https://portal.tacc.utexas.edu/user-guides/stampede>, 2017.
- [TH14] Nathan R Tallent and Adolfo Hoisie. Palm: Easing the burden of analytical performance modeling. In *Proceedings of the 28th ACM international conference on Supercomputing*, pages 221–230, 2014.
- [Tha00] D Thaler. Multipath issues in unicast and multicast next-hop selection. In *RFC 2991, Nov. 2000 [Online]*. Available: <http://tools.ietf.org/html/rfc2991>. Citeseer, 2000.
- [The13] The Energy Collective. Demand response in the US electricity market. <http://theenergycollective.com/rasika-athawale/195536/demand-response-us-electricity-market>, 2013.
- [TLDB09] W. Tang, Z. Lan, N. Desai, and D. Buettner. Fault-aware, utility-based job scheduling on Blue Gene/P systems. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10, Aug 2009.
- [TOP16] TOP500.org. Top 500 list. <https://www.top500.org/lists/2016/11/>, 2016.
- [TRLD12] Wei Tang, Dongxu Ren, Zhiling Lan, and Narayan Desai. Adaptive metric-aware job scheduling for production supercomputers. In *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, pages 107–115. IEEE, 2012.
- [Tsa13] Brian Tsay. The Tianhe-2 supercomputer less than meets the eye. *SITC Bulletin Analysis*, 2013.

- [TTG⁺16] Kun Tang, Devesh Tiwari, Saurabh Gupta, Ping Huang, Qiqi Lu, Christian Engelmann, and Xubin He. Power-capping aware checkpointing: On the interplay among power-capping, temperature, reliability, performance, and energy. In *Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on*, pages 311–322. IEEE, 2016.
- [Val82] Leslie G. Valiant. A scheme for fast parallel communication. *SIAM journal on computing*, 11(2):350–361, 1982.
- [VB81] Leslie G Valiant and Gordon J Brebner. Universal schemes for parallel communication. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 263–277. ACM, 1981.
- [VDP12] Abhinav Vishnu, Jeff Daily, and Bruce Palmer. Designing scalable PGAS communication subsystems on Cray Gemini interconnect. In *High Performance Computing (HiPC), 2012 19th International Conference on*, pages 1–10. IEEE, 2012.
- [VSC12] Brian Van Straalen and Phil Collela. Resiliency and codesign. In *DOE Exascale Research Conference*, 2012.
- [WBB⁺14] Jun Wang, Jesse Beu, Rishiraj Bheda, Tayana Conte, Zhenjiang Dong, Chad Kersey, Michelle Rasquinha, George Riley, Wanjuan Song, He Xiao, et al. Manifold: A parallel simulation framework for multicore systems. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, pages 106–115, 2014.
- [WCM⁺13] Xingfu Wu, Hung-Ching Chang, Shirley Moore, Valerie Taylor, Chun-Yi Su, Dan Terpstra, Charles Lively, Kirk Cameron, and Chee Wai Lee. Mummi: multiple metrics modeling infrastructure for exploring performance and power modeling. In *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery*, page 36, 2013.
- [WGT⁺05] David Wang, Brinda Ganesh, Nuengwong Tuaycharoen, Kathleen Baynes, Aamer Jaleel, and Bruce Jacob. Dramsim: A memory system simulator. *ACM SIGARCH Computer Architecture News*, 33(4):100–107, 2005.

- [WHLMR16] Hao Wang, Jianwei Huang, Xiaojun Lin, and Hamed Mohsenian-Rad. Proactive demand response for data centers: A win-win solution. *IEEE Transactions on Smart Grid*, 7(3):1584–1596, 2016.
- [WJ96] Steven JE Wilton and Norman P Jouppi. Cacti: An enhanced cache access and cycle time model. *Solid-State Circuits, IEEE Journal of*, 31(5):677–688, 1996.
- [WLLMR14] Adam Wierman, Zhenhua Liu, Iris Liu, and Hamed Mohsenian-Rad. Opportunities and challenges for data center demand response. In *Green Computing Conference (IGCC), 2014 International*, pages 1–10. IEEE, 2014.
- [WLPA11] Andrew Waterman, Yunsup Lee, David A Patterson, and Krste Asanovic. The risc-v instruction set manual, volume i: Base user-level isa. *EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2011-62*, 2011.
- [WNC⁺14] Yubo Wang, Hamidreza Nazaripouya, Chi-Cheng Chu, Rajit Gadh, and Hemanshu R Pota. Vehicle-to-grid automatic load sharing with driver preference in micro-grids. In *Innovative Smart Grid Technologies Conference Europe (ISGT-Europe), 2014 IEEE PES*, pages 1–6. IEEE, 2014.
- [WNC⁺15] Nathan Wichmann, Cindy Nuss, Pierre Carrier, Ryan Olson, Sarah Anderson, Mike Davis, Randal Baker, Erik W Draeger, Stefan Domino, Anthony Agelastos, et al. Performance on trinity (a cray xc40) with acceptance-applications and benchmarks. *Memory*, 2:4, 2015.
- [WSB⁺11] NJ Wright, H Shan, F Blagojevic, H Wasserman, T Drummond, J Shalf, K Fuerlinger, K Yelick, S Ethier, M Wagner, et al. The NERSC-Cray center of excellence: Performance optimization for the multicore era. *CUG Proceedings*, 2011.
- [WSS⁺12] Hao Wang, Vijay Sathish, Ripudaman Singh, Michael J Schulte, and Nam Sung Kim. Workload and power budget partitioning for single-chip heterogeneous processors. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pages 401–410. ACM, 2012.

- [WTCM16] Xingfu Wu, Valerie Taylor, Jeanine Cook, and Philip Mucci. Using performance-power modeling to improve energy efficiency of HPC applications. *IEEE Computer*, 49(10):20–29, 2016.
- [WWF⁺06] Thomas F Wenisch, Roland E Wunderlich, Michael Ferdman, Anastasia Ailamaki, Babak Falsafi, and James C Hoe. Simflex: Statistical sampling of computer system simulation. *Micro, IEEE*, 26(4):18–31, 2006.
- [Yos15] Kazutomo Yoshii. Python script collection for COOLR. <https://github.com/coolr-hpc/pycoolr>, 2015.
- [YZW⁺13] Xu Yang, Zhou Zhou, Sean Wallace, Zhiling Lan, Wei Tang, Susan Coghlan, and Michael E Papka. Integrating dynamic pricing of electricity into energy aware scheduling for HPC systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 60. ACM, 2013.
- [ZB13] R. J. Zerr and R. S. Baker. Snap: Sn (discrete ordinates) application proxy, version 1.01: user’s manual. LANL, <https://github.com/losalamos/SNAP>, Accessed May 23, 2016, 2013.
- [ZJYH16] Biling Zhang, Chunxiao Jiang, Jung-Lang Yu, and Zhu Han. A contract game for direct energy trading in smart grid. *IEEE Transactions on Smart Grid*, 2016.
- [ZKK04] Gengbin Zheng, Gunavardhan Kakulapati, and Laxmikant V Kalé. Bigsim: A parallel simulator for performance prediction of extremely large parallel machines. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 78, 2004.
- [ZLTD13] Zhou Zhou, Zhiling Lan, Wei Tang, and Narayan Desai. Reducing energy costs for ibm blue gene/p via power-aware job scheduling. In *Workshop on Job Scheduling Strategies for Parallel Processing*, 2013.
- [ZSS⁺15] Yanru Zhang, Lingyang Song, Walid Saad, Zaher Dawy, and Zhu Han. Contract-based incentive mechanisms for device-to-device communications in cellular networks. *IEEE Journal on Selected Areas in Communications*, 2015.

- [ZWXL15] Nan Zhao, Minghu Wu, Wei Xiong, and Cong Liu. Optimal contract design for cooperative relay incentive mechanism under moral hazard. *Journal of Electrical and Computer Engineering*, 2015:56, 2015.
- [ZYGL15] Haihang Zhou, Jianguo Yao, Haibing Guan, and Xue Liu. Comprehensive understanding of operation cost reduction using energy storage for idcs. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 2623–2631. IEEE, 2015.
- [ZYL⁺15] Zhou Zhou, Xu Yang, Zhiling Lan, Paul Rich, Wei Tang, Vitali Morozov, and Narayan Desai. Improving batch scheduling on blue gene/q by relaxing 5d torus network allocation constraints. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 439–448. IEEE, 2015.

VITA

KISHWAR AHMED

- 2009 B.Sc., Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka, Bangladesh
- 2017 M.Sc., Computer Science
Florida International University
Miami, Florida
- 2018 Doctoral Candidate, Computer Science
Florida International University
Miami, Florida

PUBLICATIONS

1. Kishwar Ahmed, Jason Liu, and Kazutomo Yoshii. Enabling Demand Response for HPC Systems Through Power Capping and Node Scaling. Submitted to IEEE International Conference on High Performance Computing and Communications (HPCC), 2018.
2. Kishwar Ahmed, Jesse Bull, and Jason Liu. Contract-Based Demand Response Model for HPC Systems. Submitted to International Conference on Parallel Processing (ICPP), 2018.
3. Kishwar Ahmed, Jason Liu, Abdel-Hameed Badawy, and Stephan Eidenbenz. A Brief History of HPC Simulation and Future Challenges. In 2017 Winter Simulation Conference (WSC), pages 419-430. IEEE, 2017.
4. Kishwar Ahmed, Jason Liu, and Xingfu Wu. An Energy Efficient Demand-Response Model for High Performance Computing Systems. In 2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pages 175-186. IEEE, 2017.
5. Kishwar Ahmed, Jason Liu, Stephan Eidenbenz, and Joe Zerr. Scalable interconnection network models for rapid performance prediction of HPC applications. In High Performance Computing and Communications (HPCC), 2016 IEEE 18th International Conference on, pages 1069-1078. IEEE, 2016.
6. Kishwar Ahmed, Mohammad Obaida, Jason Liu, Stephan Eidenbenz, Nandakishore Santhi, and Guillaume Chapuis. An integrated interconnection network model for large-scale performance prediction. In Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS), pages 177-187. ACM, 2016.

7. Mohammad A. Islam, Kishwar Ahmed, Hong Xu, Nguyen Tran, Gang Quan, and Shaolei Ren. Exploiting Spatio-Temporal Diversity for Water Saving in Geo-Distributed Data Centers. *IEEE Transactions on Cloud Computing*, 2016.
8. Kishwar Ahmed, Mohammad A. Islam, and Shaolei Ren. A Contract Design Approach for Colocation Data Center Demand Response. In *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on*, pages 635-640. IEEE, 2015.
9. Kishwar Ahmed, Shaolei Ren, Yuxiong He, and Athanasios V. Vasilakos. Online Resource Management for Carbon-Neutral Cloud Computing. In *Handbook on Data Centers*, pages 607-630. Springer, New York, NY, 2015.
10. Kishwar Ahmed, Mohammad A. Islam, Shaolei Ren, and Gang Quan. Can data center become water self-sufficient?. In *6th Workshop on Power-Aware Computing and Systems (HotPower)*. USENIX Association, 2014.
11. Mohammad A. Islam, Kishwar Ahmed, Shaolei Ren, and Gang Quan. Exploiting Temporal Diversity of Water Efficiency to Make Data Center Less “Thirsty”. In *11th International Conference on Autonomic Computing (ICAC)*, pages 145-152. USENIX Association, 2014.
12. Samia Tasnim, Mohammad Aatur Rahman Chowdhury, Kishwar Ahmed, Niki Pissinou, and S. Sitharama Iyengar. Location aware code offloading on mobile cloud with QoS constraint. In *Consumer Communications and Networking Conference (CCNC), 2014 IEEE 11th*, pages 74-79. IEEE, 2014.
13. Kishwar Ahmed, Shaolei Ren, Vance Turnewitsch, and Athanasios V. Vasilakos. Credibility optimization and power control for secure mobile crowdsourcing. In *Communication, Control, and Computing (Allerton), 2013 51st Annual Allerton Conference on*, pages 1501-1508. IEEE, 2013.