IMPROVING THE PERFORMANCE AND ENERGY EFFICIENCY FOR POWER-CONSTRAINED HIGH PERFORMANCE COMPUTING

A Dissertation
Submitted to
the Temple University Graduate Board

In Partial Fulfillment
of the Requirements for the Degree
DOCTOR OF PHILOSOPHY

---

by
Kun Tang
Diploma Date August 2017

Examining Committee Members:

Dr. Xubin He, Advisory Chair, TU Computer and Information Sciences
Dr. Krishna Kant, Examining Chair, TU Computer and Information Sciences
Dr. Bo Ji, TU Computer and Information Sciences
Dr. Weijun Xiao, Virginia Commonwealth University
Dr. Zhigen Zhao, External Reader, TU Statistical Science

ABSTRACT

The continuous growth in computing capability has expedited the scientific discovery and enabled scientific applications to simulate physical phenomena for increased problem sizes. However, as the computing capability escalates, power constraints are becoming a first-order concern for high performance computing (HPC) facilities. For example, the U.S. Department of Energy has set a power constraint of 20 MW to each exascale machine. How to achieve the target performance under power constraints remains to be an issue. Therefore, efficient operation of these facilities requires power constraints to be taken into account at all layers, which potentially impacts the performance and energy efficiency.

In order to improve the performance and energy efficiency for computing and storage resources under power constraints, I proposed the following three techniques. First, I developed a power-aware checkpointing model through exploring the interplay among power capping, temperature, reliability, performance, and energy efficiency. Applying the model leads to maximized performance and energy efficiency, and minimized data movements over storage systems. Second, I characterized the performance and energy efficiency of HPC workflows on heterogeneous processors. In addition, I also characterized how scientific simulation and analysis react to power capping differently and how they vary based on error resilience. Based on the characterization of HPC workflows, I developed a reliability-aware platform configuration model to determine the optimal platform configuration which includes power allocation and distribution, power capping levels, and computing scales for power-constrained HPC workflows. Third, I developed a proactive burst buffer draining scheme to minimize the I/O provisioning requirement of permanent storage systems while preserving the system

I/O performance. Facing power constraints, reducing the storage provisioning level directly decreases the power consumption of storage systems. Applying the proactive burst buffer draining scheme minimizes the storage provisioning level and power consumption without compromising the storage I/O performance.

I dedicate this dissertation to my family.

ACKNOWLEDGMENTS

First, I would like to thank my advisor Dr. Xubin (Ben) He for his support. I first met Dr. He in 2011 at NAS international conference. Two years later, I entered the Ph.D. program and he became my advisor. Dr. He has been a great advisor and friend. He has patiently and wisely guided me to conduct research in academia.

Next, I would like to thank my advisory committee members at Temple University, Dr. Xubin He, Dr. Krishna Kant, Dr. Bo Ji, and Dr. Weijun Xiao, for their generous advices and suggestions. I enjoyed and learned from the discussions with them over various academic topics. In addition, I would like to thank Dr. Zhigen Zhao at Temple University for serving as the external reader. I would also like to thank my former advisory committee members, Dr. Xubin He, Dr. Carl Elks, Dr. Wei Cheng, Dr. Qiqi Lu, and Dr. Weijun Xiao, at Virginia Commonwealth University.

I also would like to thank my mentors Dr. Sudharshan Vazhkudai, Dr. Devesh Tiwari, and Dr. Saurabh Gupta at the Oak Ridge National Laboratory for my two internships. I benefited significantly from the internships and learned a lot from them. I also appreciate the opportunity of accessing the U.S. fastest supercomputer Titan and solving the cutting edge HPC issues.

Then, I would like to thank my labmates and friends, Dr. Ping Huang, Dr. Tao Lu, Dr. Pradeep Subedi, Yuhua Guo, Morgan Stuart, Tong Liu, Peng Xu, and Yijie Huangfu. I enjoyed the meetings and discussions with them, which inspired me a lot for my study. I have collaborated with Dr. Huang, Dr. Lu and Morgan and together we composed several high quality research papers. I enjoyed the teamwork.

Last but not least, I want to thank my wife for her love and support. We first met at Virginia Commonwealth University. She supported me and encouraged me to accomplish every goal and pursue the PhD degree. My wife handles daily life chores so that I can devote myself to conduct research and pursuing the degree. She is my greatest cheerleader and her love turned my black and white life colorful.

Part of the research presented in this dissertation is sponsored by the U.S. National Science Foundation under grants CNS 1218960 and CNS 1320349. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the funding agency.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

The continuous growth in computing capability has expedited the scientific discovery and enabled scientific applications to simulate physical phenomena for increased problem sizes [1, 2]. However, as the computing scale becomes larger, the demands for power supply and heat dissipation are increasing rapidly. Power and cooling cost has increased by over 400% in the past two decades [3], which has become a nontrivial part of the budget. In recent years, power and cooling constraints are becoming a first-order concern for high-performance computing (HPC) facilities. The existing facilities can barely host the next-generation supercomputer. Therefore, efficient operation of these facilities requires power constraints to be taken into account at all layers, potentially impacting the performance and energy efficiency. However, how to achieve the target performance and energy efficiency under power constraints remains to be an issue. For example, the U.S. Department of Energy has set a power constraint of 20MW to each exascale machine. To achieve exascale under this power constraint requires significant innovation in circuits, architecture, power delivery, and cooling technologies [4].

In order to improve the performance and energy efficiency under power constraints in both processor layer and storage layer, I propose the following three techniques. As illustrated in Figure 1.1, the power-capping aware checkpointing model focuses on both computing and storage resources. The reliability-aware platform configuration model focuses on the computing resources. The proactive burst buffer draining scheme focuses on the storage resources.

Firstly, I proposed to study power constraints covering both processor layer and storage layer. Scientists typically employ checkpoint-restart mechanisms to

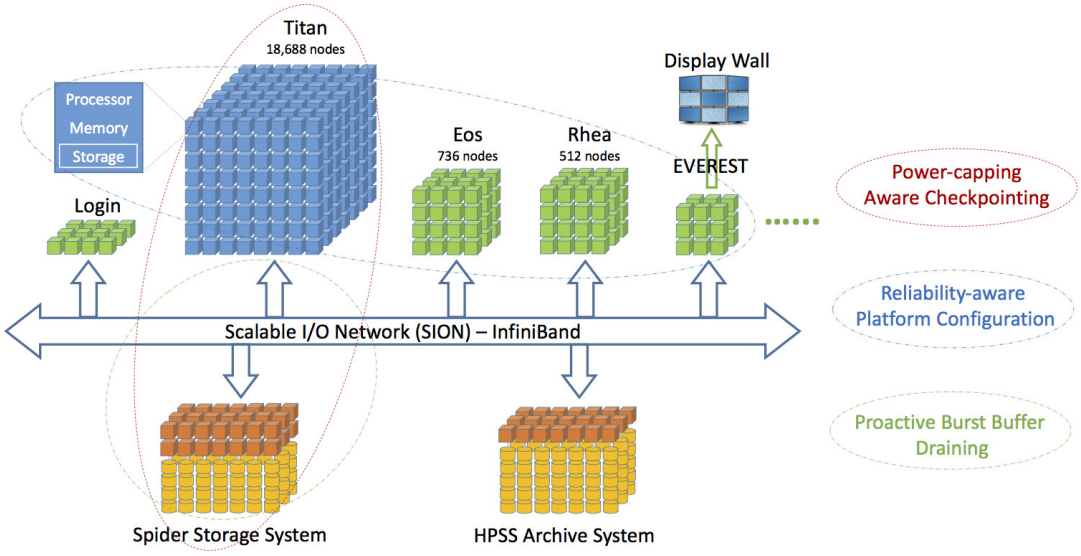


Figure 1.1.: An example of modern HPC platforms. Overall architecture of computing resources in Oak Ridge Leadership Computing Facility. The proposed techniques focus on different layers of the system, which are circled with dotted lines.

guarantee forward progress of the scientific simulations in case of failures [5–7]. The checkpointing process has its own trade-off in terms of performance and I/O overheads. It is important to checkpoint at the optimal checkpointing interval (OCI). However, it is not clear how power constraints affect the OCI decision. Therefore, I developed a power-capping aware checkpointing model to maximize performance and energy efficiency, and minimize data movements over storage systems. This power-capping aware model takes temperature, reliability, performance, and energy efficiency into account when making OCI decisions.

Secondly, I proposed to explore power constraints at the processor layer. In HPC workflows, data analytics is typically utilized to gain insights from scientific simulations. On one side, under power constraints, emerging heterogeneous processor architectures are gaining popularity since they can deliver higher performance per watt. I characterized the performance and energy efficiency of

HPC workflows on heterogeneous processors. Additionally, I also characterized the behaviors of scientific simulation and analysis under power capping and failure events. On the other side, enforcing power limits is emerging as a practical trend for power-constrained HPC facilities. However, it remains unclear how to choose the appropriate power limits for various HPC workflows and how to distribute the power limit of a workflow between simulation and analysis parts. In addition, given a power limit, it is unclear what the optimal scales and power capping levels are for various workflows, especially when taking reliability into account. In order to resolve these issues in power-constrained HPC, I developed a reliability-aware model to determine the aforementioned platform configurations for HPC workflows.

Thirdly, I proposed to investigate power constraints at the storage layer. HPC applications usually show bursty I/O behaviors [8]. In order to expedite the applications, permanent storage systems are usually provisioned to serve such I/O bursts. Approaching the era of exascale computing, non-volatile RAM is introduced as burst buffers to absorb the bursty bulk data. However, without judiciously draining the burst buffers, I/O bursts are passed down to the underlying storage systems [9] which still need to be provisioned to serve the I/O bursts. In order to minimize the I/O provisioning requirement and reduce the power consumption of the storage systems, I developed a proactive draining scheme to manage the draining process of distributed node-local burst buffers.

1.1 Power-capping Aware Checkpointing

As the computing scale becomes larger, the likelihood of failures also increases. Failures prevent scientific applications from making forward progress. To address this problem, scientists typically employ checkpoint-restart mechanisms to guarantee forward progress of scientific simulations in case of failures [5–7]. Checkpointing is a periodic process that writes the "required-to-recover" appli-

cation state to permanent storage systems. When a failure occurs, the application can restart from the latest checkpoint. Although checkpoint and restart mechanisms can keep scientific simulations moving forward, writing and reading application state incurs huge I/O overheads, which also impedes the scientific productivity [10, 11]. At exascale, such overheads are anticipated to increase further. In fact, it is estimated that in some cases applications may end up spending more than 50% of total execution time on checkpoint, restart, and lost work [12, 13].

The checkpointing process has its own trade-off in terms of performance and I/O overheads. A small checkpointing interval leads to high checkpointing I/O overheads while a large checkpointing interval results in more lost work if a failure occurs. It is important to perform checkpointing at the optimal checkpointing interval (OCI) – a problem well-studied for the last several decades. Young [14] proposed a first order approximation to the optimal checkpointing interval. Daly [15, 16] derived a high order estimation of the optimal checkpointing interval. There have been several other studies proposing finer refinements to these models, but none of the prior works has considered the interaction between power constraints and checkpointing intervals. Power consumption is becoming a first-order concern for HPC facilities. Therefore, efficient operation of these facilities requires power constraints to be taken into account at all layers. Power capping essentially limits the maximum allowable power consumption of a platform, potentially impacting temperature, reliability, performance, and energy efficiency. However, it is not clear how power capping affects OCI decisions. What are the involved temperature, reliability, performance, and energy trade-offs? Therefore, the goal of this work is to develop an understanding about the interaction between power capping and scientific applications relying on checkpoint-restart mechanisms. This work is based on real-system experiments, analytical models, and statistical techniques, and is driven by data obtained from the real large-scale computing facility, Oak Ridge Leadership Computing Facility [17].

The contributions of this work are as follows. First, I studied the impact of power capping on compute and checkpointing phases for a variety of scientific applications utilizing a widely-used checkpoint library (Berkeley Lab Checkpoint/Restart) [5]. I also demonstrated and quantified how power capping affects the system reliability due to change in processor temperature. Second, I developed a power-capping aware OCI model which derives OCI for both execution time and energy consumption. Third, I validated the power-capping aware model, and presented model and simulation driven results for a wide range of scenarios. I showed that the proposed model is significantly more accurate than previous power-capping unaware models. Compared to the previous models, the power-capping aware model results in significant time and energy savings for both petascale and exascale systems, with different checkpointing costs, wide range of power caps, and for different application and system characteristics. The evaluation also shows that the proposed model can save up to 18% of energy and execution time for a set of leadership applications from the Oak Ridge National Laboratory. It also reduces the amount of data movement by up to 57% for these large-scale applications. In addition, this work also reveals several interesting, and previously unknown, insights about how power capping affects temperature, reliability, performance, and energy efficiency of large-scale leadership applications[1] in the presence of system failures, checkpoint, and restart.

To be noted that, this work [18] was published in the 46th international conference on dependable systems and networks (DSN).

1.2 Reliability-aware Platform Configuration

HPC workflows comprise of large-scale scientific simulations and data analysis tasks, which produce and consume data respectively. Scientific simulations run iteratively and produce raw data periodically [8]. In order to gain scientific

[1]Leadership applications refer to the large-scale applications that run on supercomputers facilitated by the U.S. Department of Energy.

insights, a series of data analysis tasks are performed on the raw data. Using offline analysis, data go through the persistent storage systems within the workflow, which incurs enormous overheads. By comparison, online analysis (in-situ and in-transit) is gaining popularity in extreme-scale scientific analysis, because it usually reduces the volume of raw data by several orders of magnitude [19]. In in-situ and in-transit workflows, analysis runs simultaneously with scientific simulation in a pipelined manner.

In-situ, in-transit, and offline workflows have their own pros and cons. Extant studies have largely focused on the performance of HPC workflows instead of energy efficiency. However, in recent years, power and energy concerns are drawing more and more attention in HPC. Under power constraints, higher energy efficiency means better performance per unit power consumption. How to run HPC workflows in an energy-efficient manner remains unclear on the emerging heterogeneous processor architectures. There are multiple platform choices to run HPC workflows, such as high performance and power-consuming CPUs (brawny processors), low performance and power-consuming CPUs (wimpy processors), and coprocessors. In addition, it is also not clear how simulation and analysis parts of an HPC workflow behaves under power capping and failure events. In order to resolve the above uncertainties and guide the rethinking of workflow execution approaches, I performed comprehensive performance and energy efficiency evaluations for HPC workflows on emerging heterogeneous processor architectures. In addition, I also characterized the different behaviors of HPC workflows under power capping and failure events.

For power-constrained HPC workflows, enforcing power limits directly impacts the performance and potentially impacts the energy efficiency. However, it is not clear how to choose the appropriate power limits for various HPC workflows. Furthermore, exascale storage I/O requirement is projected to have an O(100) increase as compared with petascale computing [20–23], which requires more powerful data analysis. In HPC workflows, assigning too much power to

one part (either simulation or analysis) affects the other part negatively under power constraints. However, it remains unclear how to distribute the power limit between the simulation and analysis in a workflow. In addition, facing the power constraints, hardware over-provisioning also draws considerable interests [24]. Under a power limit, it is essential to choose the appropriate power capping levels and scales for simulation and analysis respectively. Both power capping and scaling affect the system reliability, which follow inverse variation under power constraints. For example, given a power limit, an increment in the power capping level leads to a decrement in scale. My previous study [18] reveals that a higher power capping level leads to a smaller MTBF (mean time between failures). In addition, a smaller scale is always accompanied by a larger MTBF. Therefore, it is essential to explore the trade-off between power capping and scaling in order to improve the system reliability which has significant impact on the overall application performance and energy efficiency. However, under power limits, it is unclear what the optimal power capping levels and scales are for HPC workflows.

To the best of my knowledge, no prior study has investigated the optimal platform configuration for power-constrained HPC workflows. In this work, I developed a reliability-aware model to determine the optimal platform configuration, which includes power allocation and distribution, power capping levels, and scales for power-constrained HPC workflows. This work is based on real-system measurements, statistical techniques, data obtained from leadership computing facilities, and analytical models.

The contribution of this work are as follows. First, I evaluated the performance and energy efficiency of HPC workflows on heterogeneous processor architectures. Second, I characterized how simulation and analysis parts of HPC workflows react differently to power capping and vary in terms of error resilience. Then, I developed a reliability-aware model to predict the optimal platform configuration for power-constrained HPC workflows, considering both execution time and energy consumption. Finally, I validated my model and presented model-

driven studies for a wide range of real-system scenarios. This study reveals several interesting and new insights about how platform configuration affects the execution time and energy consumption of HPC workflows under power constraints.

1.3 Proactive Burst Buffer Draining

In high-performance computing (HPC), scientific applications usually show periodic bursty I/O behaviors [8]. In the computation phase, scientific applications rarely perform storage I/O operations. Analysis files and checkpoint files are written intensively to the permanent storage system at the end of every computation segment. At machine level, various applications run simultaneously. The I/O phases of different applications overlap with each other, which leads to even higher I/O peaks. Permanent storage systems are typically provisioned to serve those I/O peaks, in order to minimize I/O overheads between consecutive computation segments. However, this leads to resource underutilization between consecutive I/O peaks. Taking the former supercomputer Intrepid at Argonne National Laboratory as an example, the aggregate I/O throughput is lower than one-third of the peak I/O bandwidth for 99% of the time [25] [26].

The ever-increasing computing capability exerts escalating I/O demands on the storage systems. In order to boost I/O performance, burst buffers are introduced to absorb the bulk data and expedite I/O requests [9]. For example, supercomputer Cori at National Energy Research Scientific Computing Center is equipped with an off-node shared burst buffer [27]. In addition, future supercomputer Summit at Oak Ridge National Laboratory will have non-volatile RAM serving as node-local burst buffers upon delivery [28]. A burst buffer is a layer of high-performance storage, which sits between main memory and permanent storage systems. HPC applications dump data rapidly to burst buffers which asynchronously drain data to the underlying permanent storage system.

Burst buffers can absorb I/O bursts, thus providing opportunities to relax the provisioning requirement on I/O bandwidth of the underlying permanent storage system. For example, with the introduction of high-performance burst buffers, the permanent storage system of the next-generation supercomputer Summit [28] is provisioned to have the same bandwidth (1 TB/s) as the storage system of supercomputer Titan [29]. Unfortunately, if not judiciously designed, the burst buffer draining process still causes similar I/O bursts (as on burst buffers) on the permanent storage systems [9]. In this situation, the permanent storage system still needs to be provisioned for I/O performance rather than storage capacity. Forcing the reduction in the I/O provisioning of the permanent storage system causes the following issues. First, the storage system suffers from severe I/O contention and long tail latency at I/O peaks. This directly affects the read and restart performance of scientific applications, data analysis tasks, data visualization tasks, and interactive user activities on the storage system. Secondly, the burst buffer draining speed is also impacted around the I/O peaks, which causes data to accumulate in the burst buffers. If the burst buffer capacity is exhausted, application I/O performance will get impacted. Finally, it is hard to determine the minimal I/O provisioning requirement of the permanent storage system.

In order to relax the I/O provisioning requirement and resolve the aforementioned issues associated with bursty I/O behaviors, I propose a proactive burst buffer draining scheme to regulate the bursty I/O behaviors and relax the I/O provisioning requirement. Based on the proactive burst buffer draining scheme, I propose a storage I/O provisioning model to predict the I/O provisioning requirement for the permanent storage system. I evaluate the proactive burst buffer draining scheme and show that it flattens I/O peaks and preserves the I/O performance for other clients sharing the storage system. I also validate the I/O provisioning model based on real-system traces and simulation, and perform sensitivity studies on model parameters. Furthermore, I demonstrate that applying the proactive burst buffer draining scheme to leadership applications achieves

near-maximal reduction in storage I/O provisioning requirement. Leadership applications refer to large-scale applications running on the supercomputers facilitated by the U.S. Department of Energy.

CHAPTER 2
LITERATURE REVIEW

In this chapter, I review the related literatures covering various aspects which include power capping, checkpointing, HPC workflows, power-aware job scheduling, processor heterogeneity, hardware over-provisioning, burst buffers, and I/O scheduling.

2.1 Power Capping Techniques

There are many ways to achieve power capping effects. Power capping is generally achieved through either Dynamic Voltage and Frequency Scaling (DVFS) [30], or throttling by idle cycle insertion [31]. Based on DVFS, Lefurgy et al. [30] performed power capping through the introduction of a feedback controller. Recent works have also explored applying DVFS to I/O intensive tasks such as garbage collection [32]. Gandhi et al. [31] proposed to achieve power capping through flipping between highest performance state and low-power idle state. This method could achieve a finer granularity in power capping and a higher processor frequency. Under power capping, processor manufacturing variability causes performance loss. Inadomi et al. [33] and Gholkar et al. [34] proposed to address this issue through applying non-uniform power capping. The power capping tool used in this dissertation is Intel Power Governor, which utilizes the throttling technique [35] [36].

2.2 Checkpointing Techniques

Recognizing the importance of checkpointing, researchers have long-investigated methods to reduce the checkpointing overheads. Some studies have focused on reducing checkpoint data size itself, for example via designing incremental checkpointing schemes [37]. Some researchers have proposed diskless checkpointing through redundancy, such as replication [38]. Compared with the disk-based checkpointing techniques, diskless checkpointing consumes excessive computing resources, including processor, memory, network, etc. Finally, one of the most widely used ways to reduce checkpointing overheads has been to derive optimal checkpointing interval. Young [14] proposed a first order approximation to the optimum checkpointing interval based on the assumption that system failures follow a Poisson process. Based on the first order model, Daly [15, 16] proposed a high order estimation of the optimum checkpointing interval. The high order model can predict the OCI more accurately when mean time between failures (MTBF) becomes smaller. Recently, some works have taken failure characteristics into account to tune optimal checkpointing interval [12], and make checkpointing more energy-efficient [39]. However, none of these works investigate the impact of power capping on checkpointing for large-scale HPC applications.

2.3 HPC Workflows and Power-aware Scheduling

The continuous growth in computing capability exacerbates the I/O bottleneck issues in HPC. Online analysis draws considerable interest in extreme-scale scientific analysis, since it usually reduces the volume of scientific raw data significantly. Li et al. [40] and Zhang et al. [41] proposed to run both simulation and analysis workloads using separate cores on a chip. Besides, Bennett et al. [19] proposed to combine in-situ and in-transit processing to achieve the trade-off between performance and data movement cost.

Existing research on power-aware scheduling mainly focus on energy savings of scientific simulations [42] [43]. Bailey et al. [44] proposed a predictive model to estimate the performance and power for scientific simulations. It is alternative to the regression analysis on power capping impact, which the platform configuration model bases on. Langer et al. [45] evaluated the performance and energy trade-off for scientific simulations under power constraints. In exascale computing, the O(100) increase in storage I/O requirement [21] gives prominence to the online analysis of HPC workflows. This dissertation targets HPC workflows and investigates the impacts of platform configurations on performance, energy, and energy efficiency under power constraints.

2.4 Processor Heterogeneity and Hardware Over-provisioning

Heterogeneous processor design has recently become popular in HPC, e.g., utilizing coprocessors to boost performance and improve energy efficiency. An emerging trend is to utilize low-power smaller cores (aka wimpy cores) to reduce the high power consumption and improve the energy efficiency of HPC workloads. For example, the Mont-Blanc project in the Barcelona supercomputing center houses an energy-efficient high performance ARM-based cluster. Ou et al. [46] evaluated the energy and cost-efficiency of ARM clusters. The conclusion of their study is that ARM clusters are advantageous in lightweight computation, but they are relatively inefficient in executing compute-bound applications. Since ARM processors are mainly designed and optimized for mobile devices, it is expected that running server workloads will be inefficient. Rajovic et al. [47] [48] illustrated that the performance gap between commodity processors and mobile processors is slowly narrowing. SoCs running at a larger scale could provide sufficient high performance with low energy consumption. Therefore, they concluded that SoCs are almost ready for HPC. However, this also requires the programs to have good scalability. Blem et al. [49] investigated the impact of ISA on per-

formance and energy efficiency of brawny cores and wimpy cores, and concluded that ISA does not affect either, i.e., performance and energy consumption differences between ARM and x86 platforms are due to microarchitecture-based optimizations.

Hardware over-provisioning draws considerable research interests in power-constrained HPC. The common goals shared by many researchers are maximizing the power utilization and maximizing the throughput (or minimizing the turnaround time) [50] [51]. These works are complementary to the platform configuration model proposed in this dissertation. Patki et al. [24] examined the feasibility of hardware over-provisioning in power-constrained HPC. They showed that over-provisioning leads to an average speedup of more than 50% comparing with the worst-case provisioning. However, the authors did not propose a model to derive the optimal configurations. Researchers have also investigated the power allocation schemes between CPU and memory for power-constrained scientific simulations [52] [53] [54]. These schemes are also complementary to the platform configuration model which focuses on the power allocation between different components of HPC workflows.

2.5 Burst Buffers and I/O Scheduling

I/O bottleneck issues have been studied intensively in HPC systems. Xie et al. [55] characterized the output bottlenecks in supercomputers. In order to improve overall I/O performance, Song et al. [56] and Dorier et al. [57] proposed to reduce I/O interference through cross-application I/O coordination. In order to preserve I/O bandwidth for clients sharing the storage system, researchers proposed various I/O isolation techniques [58–60]. However, if the I/O performance demand escalates beyond the capability of the storage system, I/O coordination and I/O isolation techniques become less efficient. In this case, Narayanan et

al. [61] [62] proposed to offload I/O to idle resources in order to better serve I/O peaks in storage centers.

As I/O performance demand keeps escalating, utilizing I/O offloading techniques gets constrained by the amount of idle resources. Therefore, in order to absorb the bursty bulk data, burst buffers are introduced at different layers of HPC systems. Regarding the on-node aspect, Moody et al. [6] and Bautista-Gomez et al. [7] proposed to use node-local storage to buffer HPC checkpointing data. Recently, Oak Ridge National Laboratory announced the introduction of node-local storage in the next-generation supercomputer Summit [28], which serves as burst buffers to absorb the bursty data at node-level. Concerning the off-node aspect, Abbasi et al. [63] proposed scalable data staging services for HPC applications, which allows more efficient data extraction and manipulation. Al-Kiswany et al. [64] proposed to use some of the compute nodes allocated to the application as an intermediate storage system. After that, Costa et al. [65] [66] proposed a predictive method to accelerate the exploration of the optimal provisioning configuration for the intermediate storage system. Besides, Liu et al. [9] first evaluated the design and benefits of burst buffers in HPC systems. Thapaliya et al. [67] proposed to mitigate I/O interference in a shared burst buffer through investigating I/O scheduling techniques. Wang et al. [68] developed a distributed burst buffer file system.

There are few works focused on the interaction between burst buffers and the underlying storage system. Kougkas et al. [69] proposed a buffer-based I/O coordination strategy to prevent I/O interference on the storage system. The proposed strategy leverages burst buffers to defer conflicting I/O accesses to the storage system. Wang et al. [70] proposed to flush the buffered data in sequential order and coordinate burst buffer draining processes to relieve I/O contention. However, these techniques cannot prevent the burst buffers from passing the I/O peaks on to the underlying storage system. Under this circumstance, reducing the I/O provisioning of the storage system leads to severe I/O contention issues. In

this dissertation, I proposed a proactive burst buffer draining scheme to regulate the bursty I/O behaviors and resolve I/O contention issues on the storage system.

CHAPTER 3
POWER-CAPPING AWARE CHECKPOINTING

In this chapter, I introduce the power-capping aware checkpointing technique in the following aspects. Section 3.1 describes the experimental and modeling methodologies. Section 3.2 discusses how power capping affects application performance and energy efficiency. Besides, the effects of power capping on checkpointing performance and power consumption are also studied in this section. Section 3.3 shows how power capping affects processor temperature, revisits the Arrhenius Equation [71], and verifies the empirical value of activation energy using the temperature and failure logs of supercomputer Titan [72] at Oak Ridge National Laboratory. Section 3.4 establishes the functional relationship between power capping and the optimal checkpointing interval. In section 3.5, I validate and evaluate the power-capping aware OCI model.

3.1 Background and Methodology

This work is primarily modeled after and based on the U.S. fastest supercomputer, Titan. Titan consists of 18,688 compute nodes (CPU and GPU) and more than 700 TB memory capacity, delivering a theoretical peak performance of 27 petaflops. I have also included various system design points to show the relevance and impact of the insights, and proposed models for the future exascale systems.

System failure data are utilized to validate the model and drive simulation studies, which were collected from the Oak Ridge Leadership Computing Facility. These data represent Titan's failure log for over two years since production. I evaluate the impact of the proposed model on different leadership applications.

In Table 3.1, I show the checkpoint size and execution time for such applications based on traditional hourly checkpoints.

Table 3.1.: Characteristics of leadership applications [12].

Application Name	**Scientific Domain**	**Checkpoint Data Size**	**Execution Time**
CHIMERA	Astrophysics	160 TB	360 Hours
GTC	Fusion	20 TB	120 Hours
GYRO	Fusion	50 GB	120 Hours
POP	Climate	26 GB	480 Hours
S3D	Combustion	5 TB	240 Hours
VULCUN/2D	Astrophysics	0.83 GB	720 Hours

In order to study the impact of power capping on the optimal checkpointing interval, I combine experiments, simulations, and model analyses. First, I obtain performance and temperature data under power capping on small-scale machines. Second, I develop the OCI model based on the regression analysis. Then, I validate the OCI model through simulations, and evaluate the model. Finally, I show a case study for the model at large scale based on leadership application runs.

It should be noted that performing power capping experiments directly on supercomputer Titan is currently not supported. To overcome this limitation, I choose two Xeon platforms to drive my study and gain insights. The two Xeon platforms are E5-2670 and E5-2630 respectively. E5-2670 platform has 8 cores, each clocked at 2.6 GHz with 115 watts Thermal Design Power (TDP) and 166.4 flops of double floating point peak performance. By comparison, E5-2630 platform has only 6 cores each clocked at 2.3 GHz with 95 watts TDP and 110.4 flops of double floating point peak performance. Both platforms have 64 GB of DRAM and run Linux 2.6.32 kernel with GCC 4.4.6 compiler installed. In addition, since these platforms cannot run large-scale applications that depend on

Cray linux environment and platform specific libraries, I run a wide variety of scientific applications for understanding power capping effects, taken from Rodinia benchmark suite and NPB benchmark suite (Table 4.2). These benchmarks cover a variety of science domains and were characterized using TAU profiling tool [73] and PAPI counters [74] to ensure that they represent a wide range of architectural characteristics.

A widely used system-level checkpointing library, BLCR [5], is utilized to perform system-level checkpointing on scientific applications. Performance is measured in terms of execution time. Each processor runs at its full capacity and utilizes all available cores. Librapl [24] and Intel Power Governor [35] are utilized to profile CPU power consumption and cap the package power consumption respectively. Linux-monitoring sensors (lm_sensors) are used to measure CPU temperature. Since this work is based on realistic assumptions and the model is bounded by the scope of parameters, I discuss the threats to validity and provide sensitivity studies in section 3.5.

Table 3.2.: Benchmark domain and problem size

Rodinia	Problem Size	**NPB**	Problem Size
LUD (LU Decomposition)	4M	pseudo applications	
LavaMD	4K	LU, SP & BT	B
CFD (CFD Solver)	97K	kernels	
		FT, MG, IS, EP & CG	B

3.2 Power Capping Effects on Performance

The first step toward obtaining the optimal checkpointing interval under power-capping is to understand how power-capping affects: the execution time of simulation (computation time), the checkpointing time, and system reliabil-

ity. In this section, I focus on first two goals, i.e., how power capping affects the execution time of application computation phase and checkpointing phase.

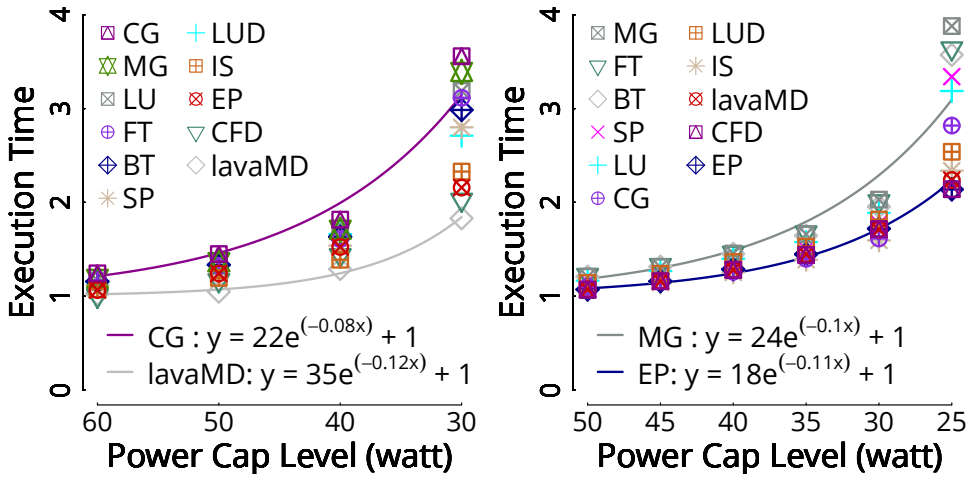
First, I present results that help to understand how different power capping levels affect the execution time of application computation phase. Fig. 3.1 shows the normalized execution time for a set of scientific benchmarks from linear algebra, computational fluid dynamics, and molecular dynamics domains (Table 4.2), on two different platforms (Section 3.1). As shown in the figure, execution time increases non-linearly across all the benchmarks on both platforms. This indicates that power capping affects the computation time significantly, although the degree of effect may vary across benchmarks and platforms. To be noted that, the average power consumption for the benchmarks on Xeon E5-2670 platform ranges from 63 watts to 78 watts, and the minimum package power consumption that Intel power governor can enforce is approximately 23 watts on this platform. This implies that the range for effective power capping levels should be between 23 watts and 63 watts to observe the effect on performance. Therefore, I choose power capping levels of 60, 50, 40, and 30 watts for Xeon E5-2670 platform. Similarly for Xeon E5-2630 platform, I choose power caps 50, 45, 40, 35, 30, and 25 watts taking average power consumption of the benchmarks into consideration.

To take this effect into consideration toward obtaining optimal checkpointing interval, this trend should be captured mathematically. The normalized execution time under power capping for a given benchmark can be fitted using an exponential function. The R-squared values of regression functions are above 0.97 for all the benchmarks on both platforms indicating statistically sound fit. Since the benchmarks are affected differently by power capping in terms of execution time, the parameters or regression coefficients are different for each application. The exponential regression functions can be generalized as Equation 4.1.

$$\overline{T_{comp}}(P_i)/T_{comp} = A \times e^{B \times Pi} + 1 \tag{3.1}$$

T_{comp} represents computation time without power capping, and $\overline{T_{comp}}(P_i)$ denotes the computation time under power cap P_i. e is Euler's number.

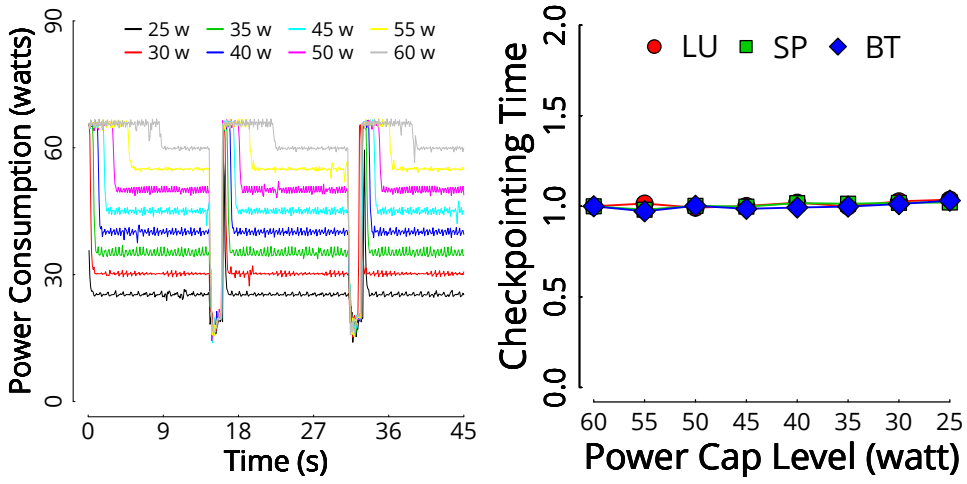
The upper bound and lower bound regression functions for both platforms are shown in Fig. 3.1. From these results, we can observe that both applications and platforms impact the coefficients in fitted exponential functions. The impact of these coefficients is studied in section 3.5, focusing on how these coefficients affect the optimal checkpointing interval and total execution time under different power capping scenarios.



(a) Xeon E5-2670 (b) Xeon E5-2630

Figure 3.1.: Effect of power capping on compute phase of benchmarks on different platforms.

Next, I present results that help to understand how different power capping levels affect the execution time of checkpointing phase. Checkpointing is an I/O intensive operation in contrast to computation intensive scientific simulation applications. Therefore, one can reasonably expect to observe distinct power capping effects on checkpointing phase than on compute phase. BLCR is used to perform checkpointing on three benchmarks (LU, SP, and BT). I find similar results for other benchmarks and platforms, therefore I only present representative



(a) Power Consumption (b) Checkpointing Time

Figure 3.2.: Effect of power capping on checkpointing phase of benchmarks: BLCR checkpointing library is used and Checkpointing time is normalized by the checkpointing time without any power capping.

results that capture trends for all the benchmarks. Fig. 3.2 shows the computing and checkpointing power consumption and execution time on Xeon E5-2630 platform under different power caps. Notice the two dips in the Fig. 3.2 (a), which correspond to two checkpoint phases. Checkpointing power consumption under all power caps are similar (approximately 21.4 watts). The effect of power capping on checkpointing time can be captured by the Eq. 3.2.

$$\overline{\beta}(P_i) = \beta \tag{3.2}$$

β represents time needed to take a checkpoint without power capping, and $\overline{\beta}(P_i)$ denotes time needed to take a checkpoint under power cap P_i. As expected, the effect of power capping on duration of checkpointing phase is minimal because checkpointing is an I/O intensive operation and throttling CPU performance to save instantaneous power does not affect I/O performance significantly, similar to observed by previous studies too.

Finding 1 *Power capping affects the execution time of compute phase significantly across all benchmarks on different platform. Furthermore, this effect can be captured by an exponential function fitting.*

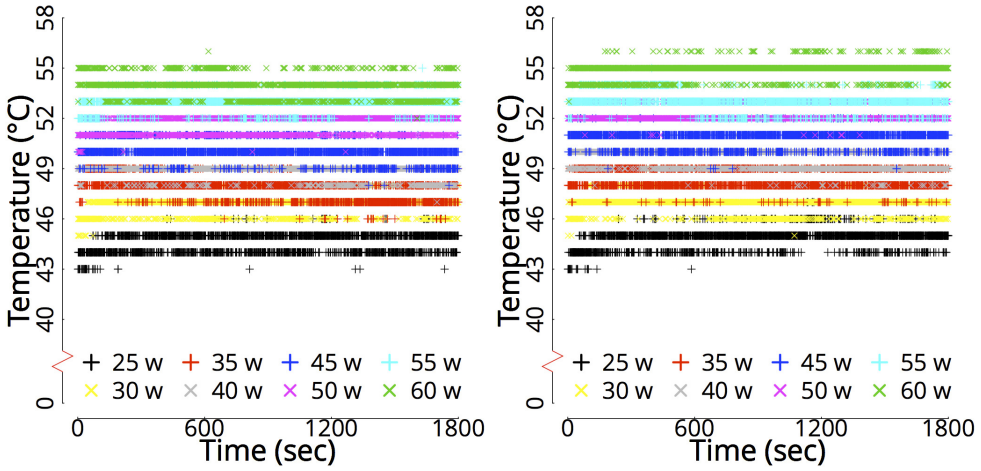
3.3 Power Capping Effects on Reliability

As discussed earlier, the next step toward obtaining the optimal checkpointing interval under power-capping is to understand how power-capping affects system reliability. However, deriving this relationship is more challenging. I show that this process consists of two steps. First, I show that power-capping directly affects the temperature of the system. Second, I show how temperature affects the system reliability.

3.3.1 Power Capping Effects on Temperature

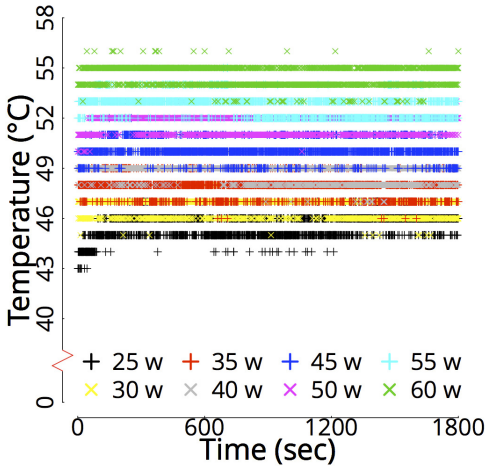
In this section, I show that different power-capping levels result in different steady-state temperature. First, I perform power capping and temperature measurement on the Xeon E5-2630 platform and E5-2670 platform. I run each benchmark for 1800 seconds under ten different power caps. Steady power consumption and temperature are calculated using the average of last 30 seconds. Fig. 3.3 shows the representative trend for two benchmarks on the Xeon E5-2630 platform. Both benchmarks show almost same behavior in terms of temperature profile under different power caps. I obtained similar results for other benchmarks and platforms, indicating that power-capping has a direct impact on the temperature and is largely independent of benchmark characteristics, unlike the effect of power-capping on execution time of compute phase.

To take this effect into consideration toward obtaining optimal checkpointing interval, I attempt to capture this trend mathematically. I find that temperature under power capping can be fitted using a linear function. The R-squared values of regression functions are above 0.99 for different platforms indicating a statisti-



(a) LU Temperature

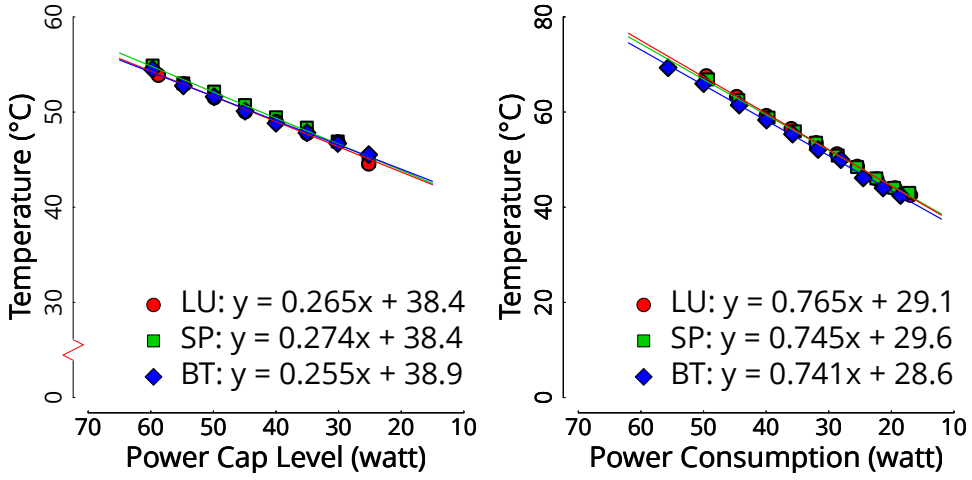
(b) SP Temperature



(c) BT Temperature

Figure 3.3.: Effect of power capping on the processor temperature (Xeon E5-2630 platform).

cally sound fit. I find the regression coefficients to be different across platforms, but same for different benchmarks on the same platform. Fig. 3.4 (a) shows this for Xeon E5-2630 platform, I find similar results for other benchmark and platform combinations (e.g., Xeon E52670 and E52603). The linear regression functions can be expressed more generically as Eq. 3.3.



(a) Xeon E5-2630 (b) Core i7-2600

Figure 3.4.: Temperature variation with power capping level, and fitted regression functions on Xeon E5-2630 platform (a), and Core i7-2600 platform (b).

$$TEMP(P_i) = C \times Pi + D \tag{3.3}$$

$TEMP(P_i)$ denotes temperature under power cap P_i.

I recognize that the relationship between power cap and processor temperature can also depend on the cooling infrastructure. Heat is dissipated in the form of both heat exchange and radiation, which grows faster as processor temperature increases. When processor heat generation equals to heat dissipation, processor temperature becomes steady. A powerful cooling infrastructure can maintain a low steady processor temperature such as in servers. Therefore, in addition to Xeon server platforms, I also a chose a desktop-like processor (Core i7-2600) to be able to compare and contrast the findings with two different types of cooling infrastructures (Fig. 3.4(a) and (b)). Repeating the exactly same experiments for core i7-2600 is not feasible because power capping is not supported on desktop processors. Therefore, different clock rates are utilized to generate distinct power consumption and temperature pairs. Steady power consumption

and temperature data of Core i7-2600 are shown in Fig. 3.4(b). These data points can still be fitted with linear functions as shown in the figure. The R-squared values of regression functions are higher than 0.99 for all the applications. As PC cooling infrastructure has less capacity as compared to server cooling infrastructure, Core i7-2600 temperature increases much faster than Xeon E5-2630 when power consumption is increased. This result also illustrates that the regression coefficients are dependent on the platform and cooling infrastructure.

Finding 2 *Power capping level directly impacts the temperature of the processor. This relationship can be captured by a linear function, and is largely independent of the application characteristics for the different platform and benchmarks pairs I tested. However, the regression coefficients are platform-specific, potentially indicating dependent on the cooling infrastructure itself.*

3.3.2 Temperature Effects on Reliability

As discussed in the previous section, power capping and system reliability are connected by understanding how power capping affects temperature and then how temperature impacts the MTBF of the system. Previous works have shown the evidence that temperature can affect the overall system reliability [75–79].

In this section, I establish how temperature affects the MTBF of the system. For this purpose, I take advantage of the Arrhenius Equation [71], which has been shown to fit computing systems [80] defining mean time between failure (MTBF) dependence on temperature.

$$\overline{MTBF}(P_i) = MTBF_{base}/F_A(TEMP(P_i)) \tag{3.4}$$

$F_A(x)$ is the acceleration factor under a given temperature x, as defined in Equation 3.5.

$$F_A(x) = e^{\frac{E_a}{k} \times (1/TEMP_{base} - 1/x)} \tag{3.5}$$

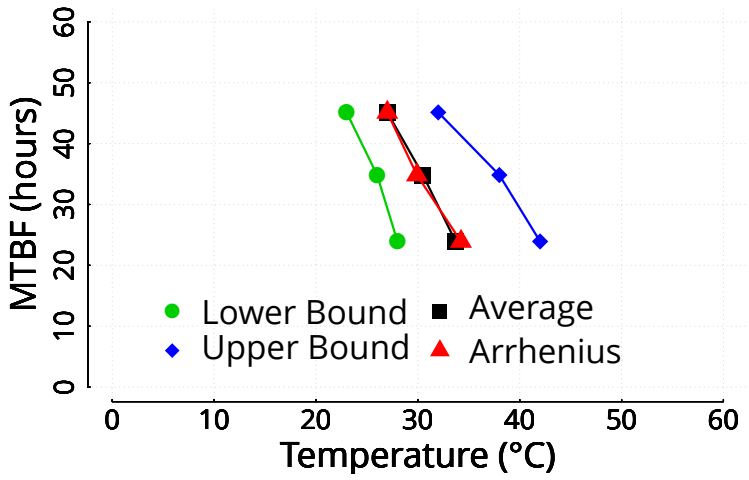


Figure 3.5.: Temperature and MTBF data of top, middle, and bottom cages on Titan. MTBF is calculated according to the total number of failures across cabinets, while temperature is calculated based on each individual cabinet.

k is Boltzmann constant which equals to 8.617×10^{-5} eV/°K. E_a represents activation energy. Using supercomputer Titan's log data, it demonstrates that this relationship holds true in a large-scale HPC computing facility. Fig. 3.5 shows the temperature and MTBF data for different cage layers across the cabinets of Titan. Each server cabinet in Titan consists of three cage layers, i.e., top, middle, and bottom cages. Cold air flows from bottom cage to top cage, which creates a gradient in ambient temperature. The temperature increases from bottom cage to the top cage, and hence lower cages tend to have shorter MTBF. I set $MTBF_{base}$ and $TEMP_{base}$ as MTBF and average temperature of bottom cages. Then, I calculate temperature for middle and top cages based on MTBF of corresponding cage level using Equation 4.18. As shown in Fig. 3.5, temperature data for middle and top cages closely match the field data obtained on Titan, for the empirical value of activation energy, $E_a = 0.7eV$. The fig-

ure also shows the variance in the temperature data to demonstrate that it falls within the range and has similar trend.

This mathematical relationship can be used to model the system's reliability behavior and its impact on the optimal checkpointing interval. However, it is important to recognize the potential limitation and scope of this approach. Power-capping alone may not be responsible for temperature of different computing components. The inefficiencies in power/cooling infrastructure may cause temperature variability, in addition to what may be caused by the power-capping alone. This approach doesn't directly and explicitly model such variance caused by the power/cooling infrastructure itself. The focus of this work is to understand the impact of power-capping on checkpointing decisions, although other environmental conditions may also contribute toward this decision. It should also be noted that, power/cooling infrastructure can not completely mitigate the temperature's impact on system MTBF without dynamically changing the cooling infrastructure load. However, current HPC facilities often do not react dynamically to load-changes in order to adjust cooling resources. They are typically designed for a fixed load and therefore, power capping effect on the temperature will exist in such systems. Therefore, it is important to explicitly model and understand the power-capping's effect on checkpointing decisions, performance and energy consumption. Finally, this work does not model the effect of variance in temperature on failures [79] since the presence of such effects in the Titan supercomputer's failure and temperature logs was not statistically significant.

Finding 3 *The system MTBF decreases with increase in temperature. The effect of temperature on the system MTBF can be modeled by Arrhenius Equation.The field data obtained on Titan validates this relationship.*

Table 3.3.: Symbols and definitions

Symbols	Definitions	Symbols	Definitions
P_i	power cap	$\overline{T_{total}}(P_i)$	total execution time
α	checkpointing interval	$\overline{T_{comp}}(P_i)$	total computation time
β, $\overline{\beta}(P_i)$	time to take a checkpoint	$\overline{T_{chkp}}(P_i)$	total time in taking checkpoints
γ	time to restart from a failure	$\overline{T_{waste}}(P_i)$	total wasted time
ϵ	fraction of lost work	$\overline{E_{total}}(P_i)$	total energy consumption
$TEMP_{base}$	baseline temperature	$\overline{E_{comp}}(P_i)$	total computation energy
$MTTF_{base}$	baseline MTTF under $TEMP_{base}$	$\overline{E_{chkp}}(P_i)$	total energy in taking checkpoints
$TEMP(P_i)$	temperature under power cap P_i	$\overline{E_{waste}}(P_i)$	total wasted energy
$F_A(x)$	acceleration factor under temp. x	$\overline{T_{waste}^{comp}}(P_i)$	total wasted computation time
P_{comp}	computing power consumption	$\overline{T_{waste}^{chkp}}(P_i)$	total wasted checkpoint time
P_{chkp}	checkpointing power consumption	$\overline{T_{restart}}(P_i)$	total time in restarting

* Symbols with overlines have the same meanings as the ones without overlines, except that they are under power capping level P_i.

3.4 Power Capping Effects on the OCI

In section 3.4.1, first, I revisit how the first order model calculates the OCI [14], as shown in Equations 3.6 to 3.11. Then I introduce the power capping aware OCI model based on the first order model in section 3.4.2. In section 3.4.3, I revisit the high order model [16] and develop the high order power-aware OCI model using the same approach in first order model. Table 5.1 lists all the parameters used in the models.

3.4.1 First Order Model

According to the first order model, when considering checkpoint and restart, the total execution time is composed of successful computation time, successful checkpoint time and wasted time caused by failures, as shown in Eq. 3.6.

$$T_{total} = T_{comp} + T_{chkp} + T_{waste} \tag{3.6}$$

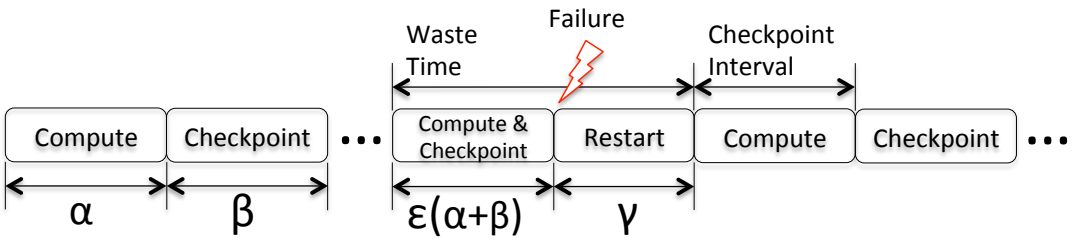


Figure 3.6.: An example of computation, checkpoint, failure, and restart. Computation is divided into multiple segments and a checkpoint is taken at the end of each segment.

An example to illustrate computing, checkpoint, failure, and restart is given in Fig. 3.6. Checkpointing interval is denoted as α. Time to take a single checkpoint is denoted as β. ϵ represents the fraction of lost computation and checkpoint. γ is the time to restart from a failure. Total amount of time spent in checkpointing can be expressed as Eq. 3.7.

$$T_{chkp} = (\frac{T_{comp}}{\alpha} - 1) \times \beta \tag{3.7}$$

Time wasted due to failures consists of lost computation (T^{comp}_{waste}), lost checkpoint (T^{chkp}_{waste}), and time spent in restart process ($T_{restart}$), which can be expressed as Eq. 3.8.

$$\begin{aligned} T_{waste} &= T^{comp}_{waste} + T^{chkp}_{waste} + T_{restart} \\ T^{comp}_{waste} &= \frac{T_{comp}}{\alpha} \times (e^{\frac{\alpha+\beta}{MTBF}} - 1)(\epsilon \times \alpha) \\ T^{chkp}_{waste} &= \frac{T_{comp}}{\alpha} \times (e^{\frac{\alpha+\beta}{MTBF}} - 1)(\epsilon \times \beta) \\ T_{restart} &= \frac{T_{comp}}{\alpha} \times (e^{\frac{\alpha+\beta}{MTBF}} - 1)(\gamma) \end{aligned} \tag{3.8}$$

Based on Eq. 3.6, total energy consumption of the first order model is shown in Eq. 3.9.

$$\begin{aligned} E_{total} =& P_{comp} \times T_{comp} + P_{chkp} \times T_{chkp} \\ &+ P_{comp} \times T^{comp}_{waste} + P_{chkp} \times (T^{chkp}_{waste} + T_{restart}) \end{aligned} \tag{3.9}$$

Replacing T_{chkp} and T_{waste} in Eq. 3.6 and Eq. 3.9, we can get the expressions of total time (T_{total}) and energy consumption (E_{total}) in the first order model.

The OCI optimized for execution time (α_t) is achieved when $\frac{d}{d\alpha}(T_{total}) = 0$. Similarly, the OCI optimized for energy consumption (α_e) is achieved when $\frac{d}{d\alpha}(E_{total}) = 0$. When $\alpha + \beta \ll MTBF$, expression $e^{\frac{\alpha+\beta}{MTBF}} - 1$ can be approximated as $\frac{\alpha+\beta}{MTBF}$. Solving the differential equations, we can get the expressions for the OCI optimized for execution time (α_t^-) and energy consumption (α_e^-) for the first order model, which are shown in Eq. 3.10 and Eq. 3.11 respectively. Note the superscript "−" signifies that OCI is from a power-unaware model.

$$\alpha_t^- = \sqrt{\beta^2 + \frac{\beta \times \gamma}{\epsilon} + \frac{MTBF \times \beta}{\epsilon}} \tag{3.10}$$

$$\alpha_e^- = \sqrt{\frac{P_{chkp}}{P_{comp}} \times (\beta^2 + \frac{\beta \times \gamma}{\epsilon} + \frac{MTBF \times \beta}{\epsilon})} \tag{3.11}$$

Finding 4 *Even without applying power-capping, the OCI optimized for energy can be smaller than the OCI optimized for performance. The difference between these OCIs gets larger as the ratio of power consumption during checkpointing to power consumption during computing becomes smaller.*

3.4.2 First Order Power-aware Model

Taking the first order model described here as a baseline, I propose a power-aware OCI model. The goal is to express T_{total} and E_{total} as functions of power cap P_i. Similar to Equation 3.6 and 3.9, we can write the following equations for execution time and energy consumption under a given power cap P_i (i.e., $\overline{T_{total}}(P_i)$, $\overline{E_{total}}(P_i)$).

$$\overline{T_{total}}(P_i) = \overline{T_{comp}}(P_i) + \overline{T_{chkp}}(P_i) + \overline{T_{waste}}(P_i) \tag{3.12}$$

$$\begin{aligned}\overline{E_{total}}(P_i) =& P_i \times \overline{T_{comp}}(P_i) + P_{chkp} \times \overline{T_{chkp}}(P_i) \\ &+ P_i \times \overline{T_{waste}^{comp}}(P_i) \\ &+ P_{chkp} \times (\overline{T_{waste}^{chkp}}(P_i) + \overline{T_{restart}}(P_i))\end{aligned} \tag{3.13}$$

Equations 3.7 and 3.8 are also applicable here if we replace T_{comp} by $\overline{T_{comp}}(P_i)$ and $MTBF$ by $\overline{MTBF}(P_i)$. According to Eq. 4.1, T_{comp} can be expressed as functions of P_i (i.e., $\overline{T_{comp}}(P_i)$). Also, based on Eq. 3.3, Eq. 4.18 and Eq. 3.5, MTBF can also be expressed as a function of P_i (i.e., $\overline{MTBF}(P_i)$). Note that β is dominated by writing checkpoints to storage system, and γ is dominated by reading checkpoints from storage system. Since I find in the experiments that time to write checkpoints does not vary significantly with power capping, as shown in Eq. 3.2, it is reasonable to assume that time to read checkpoints is also independent of power capping. Therefore, I assume γ to be independent of power capping. After performing these substitutions we can obtain detailed expressions for $\overline{T_{total}}(P_i)$ and $\overline{E_{total}}(P_i)$.

The OCI optimized for execution time (α_t^+) and optimized for energy consumption (α_e^+) are achieved when $\frac{d}{d\alpha}(\overline{T_{total}}(P_i)) = 0$ and $\frac{d}{d\alpha}(\overline{E_{total}}(P_i)) = 0$ respectively. Note that superscript "+" signifies that the OCI includes the power capping aware model. When $\alpha + \beta \ll MTBF$, solving the differential equations, we can get the functional relationship between power and OCI, as shown in Eq. 3.14 and 3.15.

$$\alpha_t^+ = \sqrt{\beta^2 + \frac{2 \times \beta \times \gamma}{\epsilon} + \frac{MTTF_{base} \times \beta}{F_A(C \times P_i + D) \times \epsilon}} \tag{3.14}$$

$$\alpha_e^+ = \sqrt{\frac{P_{chkp}}{P_i} \times [\beta^2 + \frac{2 \times \beta \times \gamma}{\epsilon} + \frac{MTTF_{base} \times \beta}{F_A(C \times P_i + D) \times \epsilon}]} \tag{3.15}$$

To be noted that, power capping aware OCI has no dependence on the regression coefficients A and B, which show how power capping affects compute phase performance.

Finding 5 *Power capping aware OCI is not determined by how the compute phase performance is affected by power capping. That is, two applications with varying sensitivity toward power capping on their compute phase performance, will have the same OCI if all else is the same.*

3.4.3 High Order Models

The high order model refines T_{waste} in the first order model in the following three steps. First, high order model introduces the fraction of lost work over a time interval $\phi(\Delta t)$ to replace ϵ, which is shown in Eq. 3.16, similar to [16]. Second, high order model defines the number of failures as $\frac{T_{total}}{MTBF}$ to consider multiple failures in a computing segment. Finally, high order model considers failures during restart processes. Refined T_{waste}^{comp}, T_{waste}^{chkp}, and $T_{restart}$ are shown in Eq. 3.17 and Eq. 3.18.

$$\phi(\Delta t) = \frac{MTBF}{\Delta t} + \frac{1}{1 - e^{\Delta t/MTBF}} \tag{3.16}$$

$$\begin{aligned} T_{waste}^{comp} =& \frac{T_{total}}{MTBF} \times [\phi(\alpha+\beta) \times \alpha \times e^{-\frac{\alpha+\beta+\gamma}{MTBF}} \\ &+ \phi(\alpha+\beta+\gamma) \times \alpha \times (1 - e^{-\frac{\alpha+\beta+\gamma}{MTBF}})] \\ T_{waste}^{chkp} =& \frac{T_{total}}{MTBF} \times [\phi(\alpha+\beta) \times \beta \times e^{-\frac{\alpha+\beta+\gamma}{MTBF}} \\ &+ \phi(\alpha+\beta+\gamma) \times \beta \times (1 - e^{-\frac{\alpha+\beta+\gamma}{MTBF}})] \end{aligned} \tag{3.17}$$

$$\begin{aligned} T_{restart} =& \frac{T_{total}}{MTBF} \times [\gamma \times e^{-\frac{\alpha+\beta+\gamma}{MTBF}} \\ &+ \phi(\alpha+\beta+\gamma) \times \gamma \times (1 - e^{-\frac{\alpha+\beta+\gamma}{MTBF}})] \end{aligned} \tag{3.18}$$

Replacing T_{waste} in the first order model, the expression of T_{total} for the high order model is shown in Eq. 3.19.

$$\begin{aligned} T_{total} =& MTBF \times (\frac{T_{comp}}{\alpha} - \frac{\beta}{\alpha+\beta}) \\ &\times e^{\frac{\gamma}{MTBF}} \times (e^{\frac{\alpha+\beta}{MTBF}} - 1) \end{aligned} \tag{3.19}$$

High order model can also be extended to power-aware OCI model using same approach in Section 3.4.2. Since the high order model only refines T_{waste}, we can derive $\overline{T_{total}}(P_i)$ and $\overline{E_{total}}(P_i)$ from $\overline{T_{waste}}(P_i)$. Similar to Eq. 3.17, Eq. 3.18, and Eq. 3.19, $\overline{T_{waste}^{comp}}(P_i)$, $\overline{T_{waste}^{chkp}}(P_i)$, and $\overline{T_{restart}}(P_i)$ can be expressed if T_{total}, $MTBF$, and ϕ is replaced by $\overline{T_{total}}(P_i)$, $\overline{MTBF}(P_i)$ and $\overline{\phi}(\Delta t)$. Similar to Section 3.4.2,

$\overline{MTBF}(P_i)$ can be obtained using Eq. 3.3, 4.18 and 3.5. Expression for $\overline{T_{total}}(P_i)$ is shown in Eq. 3.20 and Eq. $\overline{\phi}(\Delta t)$ is defined in Eq. 3.21.

$$\begin{aligned}\overline{T_{total}}(P_i) =& \overline{MTBF}(P_i) \times (\frac{\overline{T_{comp}}(P_i)}{\alpha} - \frac{\beta}{\alpha+\beta}) \\ & \times e^{\frac{\gamma}{\overline{MTBF}(P_i)}} \times (e^{\frac{\alpha+\beta}{\overline{MTBF}(P_i)}} - 1)\end{aligned} \tag{3.20}$$

$$\overline{\phi}(\Delta t) = \frac{\overline{MTBF}(P_i)}{\Delta t} + \frac{1}{1 - e^{\Delta t/\overline{MTBF}(P_i)}} \tag{3.21}$$

Based on Eq. 3.13, we can also get the expression of $\overline{E_{total}}(P_i)$ for power capping aware high order model. The OCI optimized for execution time (α_t^+) and optimized for energy consumption (α_e^+) are achieved when $\frac{d}{d\alpha}(\overline{T_{total}}(P_i)) = 0$ and $\frac{d}{d\alpha}(\overline{E_{total}}(P_i)) = 0$ respectively. The analytical solution of the OCI can be found in [15] [16]. However, the analytical solutions are approximation to or estimation of OCIs based on certain conditions. In order to accurately predict OCIs under all conditions, I use the numeric solver "vpasolve" in MATLAB to calculate OCIs for both first order power-aware model and high order power-aware model.

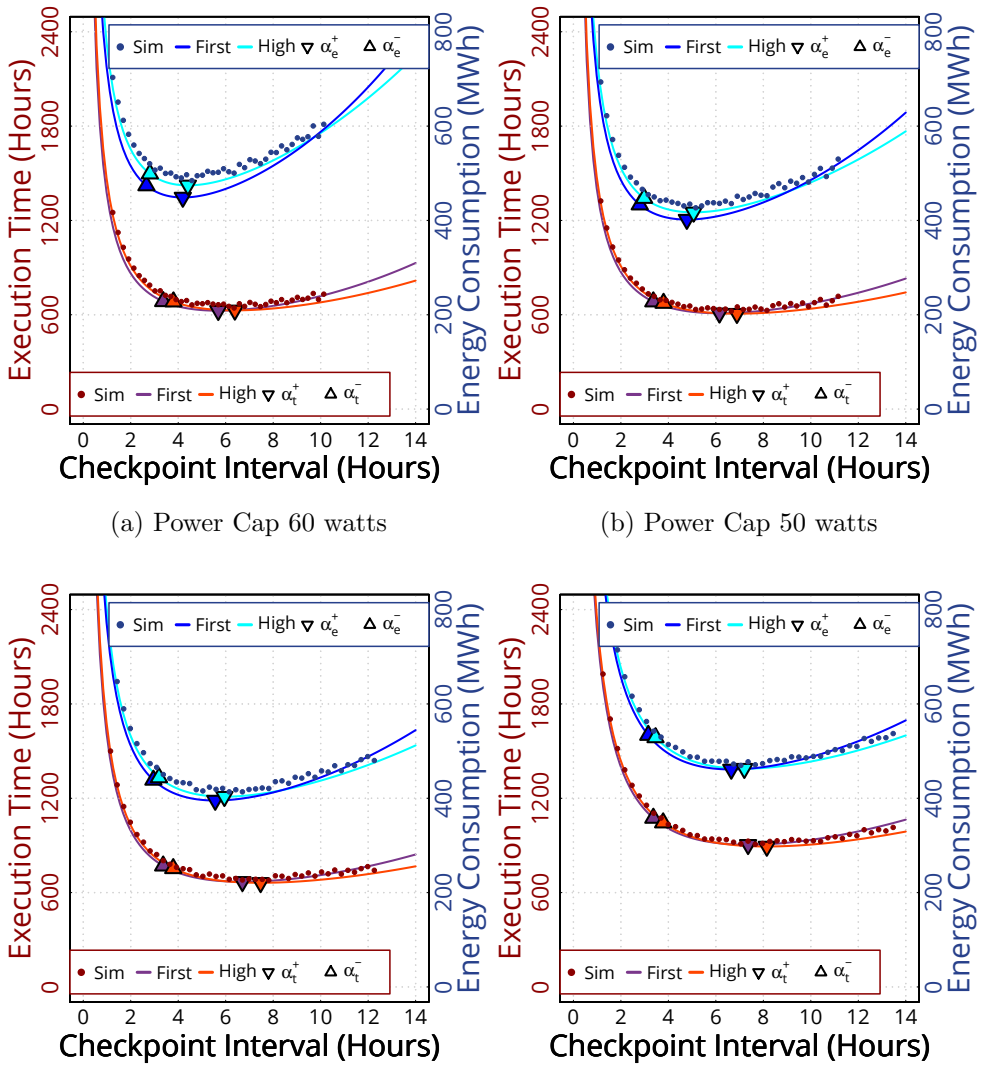
3.5 Model Validation and Model-driven Study

In this section, I perform simulations using an event-driven simulator and validate the model against simulation results. Then, I conduct a model-driven study to compare the power-aware OCI models with the prior power-capping unaware OCI models (i.e., first order and high order models).

To validate the power-capping aware OCI model, I use a simulation based approach to compare against. I utilized an event-driven simulator developed in [12] to simulate the compute phase, checkpointing phase, and failure events. The simulator generates random failures which follow a Poisson process, and intervals between failures follow an exponential distribution. The execution time, checkpointing time, and MTBF are adjusted in the simulation based on the input power capping level in accordance to relationships derived in previous sections.

The findings are bounded by the assumptions and parameter settings. To mimic real-world scenario, the simulation based evaluation is driven by parameters obtained from real-system experiments, large-scale application characteristics, empirical parameters obtained from HPC facility. The simulation based study uses power capping related coefficients that are experimentally obtained from the different Intel Xeon platforms for a variety of scientific applications. Simulation based study is driven by failure and I/O data obtained from the Titan supercomputer and the temperature dependence of MTBF has also been simulated based on the Titan supercomputer data. At the same time, I also acknowledge that it is not always possible to obtain real-world data to drive simulation based studies. In such cases, I have used a range of parameters to simulate the impact of such factors. Failure events are also simulated using Weibull distribution to mimic real-world scenarios and obtained similar accuracy and results.

Fig. 3.7 shows the total execution time and energy consumption under different power capping levels for a Titan-like petascale system. The figure curves correspond to the power-capping aware models and the simulation. I also mark the OCIs obtained by previous models and OCIs obtained by new power-capping aware model. The simulation setup assumes a 120 hour long application running on a Titan-like supercomputer, which is composed of 20,000 nodes. It assumes to have the same MTBF as Titan under the same temperature. The power capping effect is modeled after Xeon E5-2630 processor. The regression coefficients corresponds to that platform and pseudo applications from the NAS benchmark suite (LU, SP, BT) on that platform. Note that these applications have similar regression coefficients. I also perform sensitivity analysis with regard to such coefficients as well. The checkpointing power is taken as 21.4 watts as on measured on this platform. The baseline power consumption is 64.1 watts under no power capping. Baseline temperature is calculated based on Eq. 3.3, and baseline MTBF is calculated based on MTBF and temperature data from Titan logs using Arrhenius Equation with the empirical value of activation energy.



(a) Power Cap 60 watts

(b) Power Cap 50 watts

(c) Power Cap 40 watts

(d) Power Cap 30 watts

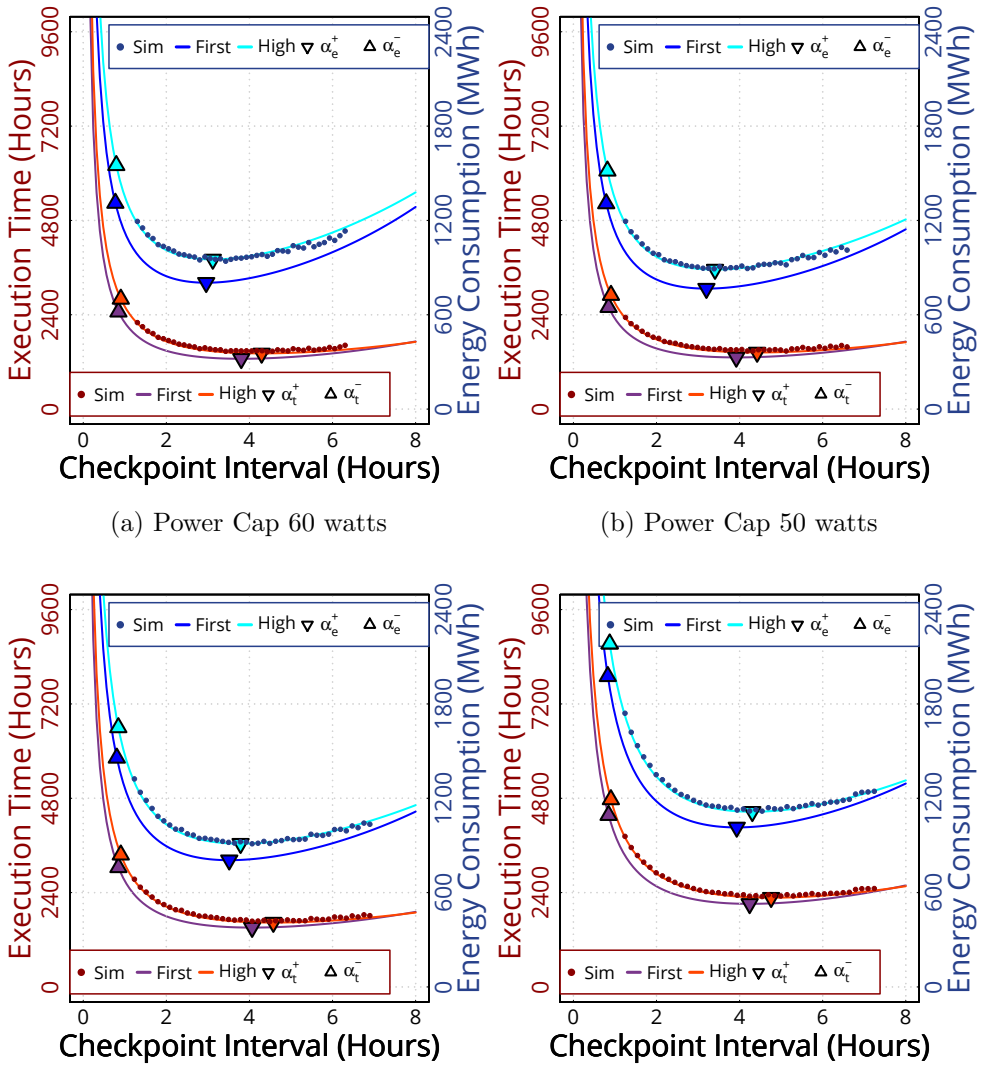
Figure 3.7.: Execution time and energy consumption under various checkpointing intervals. Legends at the bottom represent execution time (T_{total}), and legends at the top represent energy consumption. "Sim" is simulation results, "First" represents first order model, and "High" denotes high order model. OCIs derived from prior models (i.e., α_t^- and α_e^-) are marked with a triangle facing upwards and OCIs calculated from the power-capping aware models (i.e., α_t^+ and α_e^+) are marked with a triangle facing the downwards.

The checkpointing time is taken as to be 3.6% of the compute time as obtained from the experiments from the BLCR checkpointing library. I assume that time to restart equals to time to checkpoint, since the former one primarily reads checkpoints from storage system and the latter one mainly writes checkpoints to storage system. I simulate more than 40 checkpointing intervals under each power cap.

Several observations can be made from Fig. 3.7. First, across various power caps, the power-capping aware model predicted OCI closely matches with the simulation results corresponding to minimum execution time and energy consumption. Second, the OCI predicted by previous models, which do not take power capping effects into account, are significantly far from optimal OCI points. In most cases, this results in more than 10% performance loss and additional energy consumption. The difference between the first and high order model is not significant in the cases presented here. Finally, applying the power-capping aware model results in significant savings as the power cap becomes smaller. For example, the performance difference between power-capping aware OCI model and high order model increases from 8.8% to 17.2% when power cap drops from 60 watts to 25 watts. This is primarily because the new model captures the MTBF change due to power capping better as the power cap drops.

Finding 6 *The power-capping aware model predicted OCI closely matches the minimal execution time and energy consumption achieved by the simulation runs. Applying the power-capping aware model results in significant performance and energy savings compared to previous models. Additionally, such savings increase significantly as the power cap gets tighter.*

Next, I show that the power-capping aware model is validated for an exascale-like system as well. Fig. 3.8 shows that the model closely follows the simulation and power-capping aware model predicts OCI accurately. Interestingly, the improvements in performance and energy due to new model is higher compared to the petascale system. This is because at exascale the MTBF becomes smaller



(a) Power Cap 60 watts

(b) Power Cap 50 watts

(c) Power Cap 40 watts

(d) Power Cap 30 watts

Figure 3.8.: Execution time and energy consumption under various checkpointing intervals for exascale system. Legends have the same meanings as in Figure 3.7.

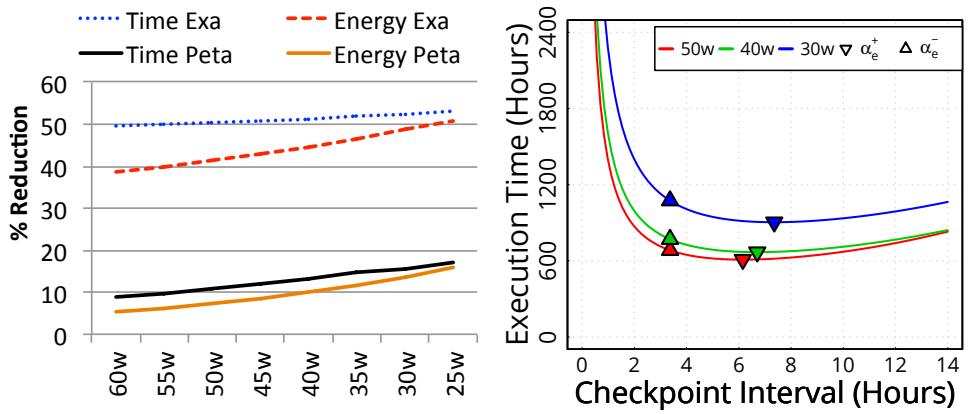
and hence, previous models take checkpoint more frequently and incur very high I/O overhead. However, the power-capping aware model adjusts the OCI taking both the system scale and power capping into account. It estimates the OCI to be a bit higher and hence, results in significantly less I/O overhead. As a key summary, Fig. 3.9(a) shows that applying the power-capping aware OCI

model can reduce the total execution time and energy consumption, compared to prior OCI models at both peta- and exa-scale. For a petascale system, execution time can be improved between 8.8% to 17.2% using the high order power-capping aware OCI model. This effect is even more pronounced for exascale system where execution time can be improved between 49.4% to 52.9% using the high order power-capping aware OCI model. Similar savings can be observed for energy consumption as well when applying power-capping aware OCI model.

In Fig. 3.7 and 3.8, it can be noticed that the execution time and energy curves shift upward when the power cap is reduced. To illustrate and understand this trend better, Fig. 3.9(b) shows the execution time curves under power capping levels of 50 watts, 40 watts, and 30 watts. α_t^+ and α_t^- are also marked for each power cap in the figure. Note that α_t^- are on same vertical line because power cap unaware OCI stays the same. On the other hand, as the power cap is reduced the curves shift upward due to increased execution time, and the curve shifts towards the right due to increasing MTBF. Therefore, α_t^+ increases as the power cap decreases due to change in MTBF. Prior models can not take this effect into account and lead to suboptimal OCI estimation. This explains why and how the power-capping aware model outperforms the prior models in different situations.

Finding 7 *As the system scale increases, the benefit of power-capping aware OCI model also increases significantly compared to the prior models.*

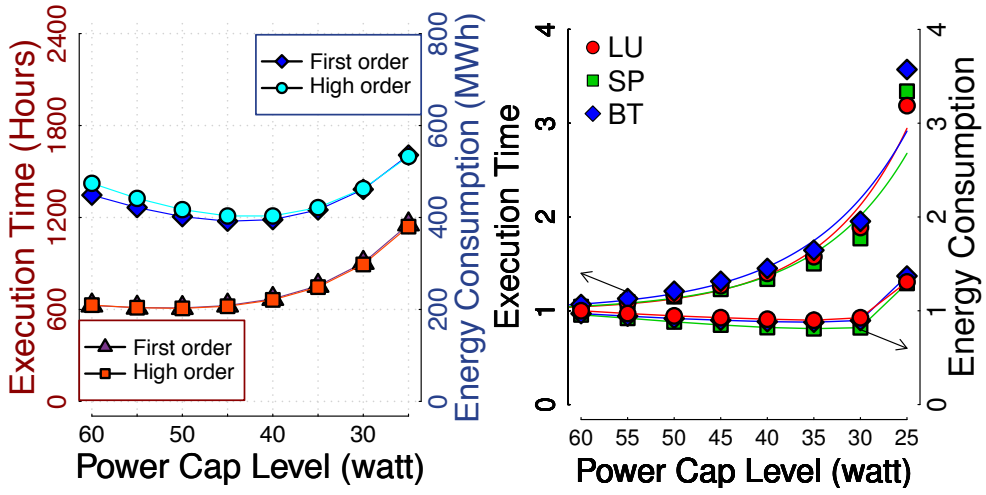
Next, I show that it is critical to choose the correct power cap level to achieve minimum execution time and energy consumption. Fig. 3.10(a) shows the best performance is achieved when power cap is 50 watts, and the lowest energy consumption is achieved when power cap is 45 watts. This illustrates that the optimal power capping level itself depends upon the metric of optimization (e.g., performance, and energy). The corresponding OCI on these power capping levels would be different as well, this can be obtained via the power-capping aware model. Note that this result includes the failure events, checkpointing, and restart phase.



(a) (b)

Figure 3.9.: Improvement in performance and energy consumption at petascale and exascale under different power caps (a), and execution time curves under different power capping levels with OCIs for power aware model and prior model without power capping awareness (b).

On the other hand, the results in Fig. 3.10(b) do not take failures, checkpoint, and restart into account. Interestingly, in this case the lowest execution time is achieved with power cap of 60 watt, and least energy consumption is observed for power cap of 35 watts when we do not consider failures, checkpoint, and restart. This is a critical finding that illustrates that optimal power cap levels can not alone be decided by how power capping affects the application in isolation, without taking failures, checkpointing, and restart phase into account. These shifts in optimal power caps are caused by the impact of power capping on MTBF. I find that using 45 watt power cap instead of 35 watt power cap leads to 20.2% savings in energy consumption. Note that this reduction in energy consumption is between power-unaware model based OCI at 35 watt and power-aware model based OCI at 45 watts. Similarly, using 50 watt power cap instead of 60 watt power cap leads to 12.6% reduction in execution time.



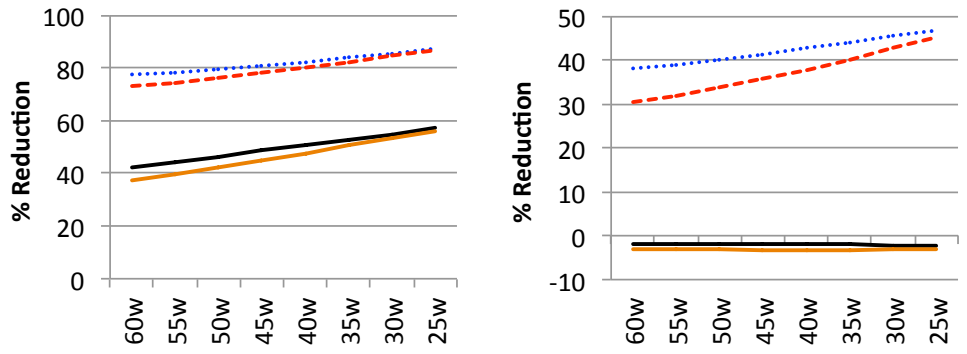
(a) Checkpointing with OCI

(b) No checkpointing

Figure 3.10.: Total execution time and energy consumption under various power caps with the optimal checkpointing interval (a), and when checkpointing overhead is ignored (b).

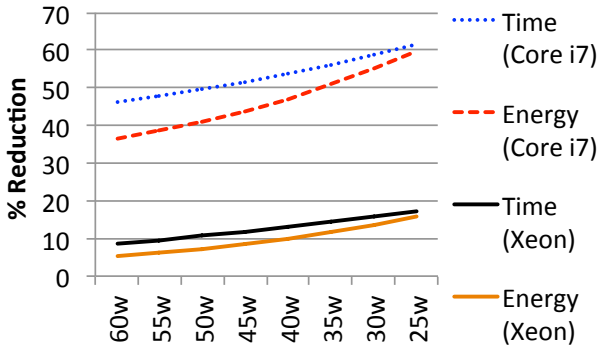
Finding 8 *The optimal power cap levels for minimizing execution time and energy consumption are different and so are their corresponding OCIs. The optimal power cap levels for minimizing execution time and energy consumption change once failures, checkpoint, and restart phases are taken into account. The corresponding difference in execution time and energy consumption is significant. The results also show that power-capping aware OCI model leads to significant improvements.*

Next, I study how improvement obtained by the power-capping aware OCI model changes when the application specific parameters (A and B), and platform specific parameters (C and D) change. I also study the impact of time to checkpoint (β) on the improvements. It is followed by evaluation for real scientific applications. I compare the power-capping aware OCI model with prior OCI models, in terms of execution time, energy consumption, and overall checkpointing data volume.



(a) Checkpoint time

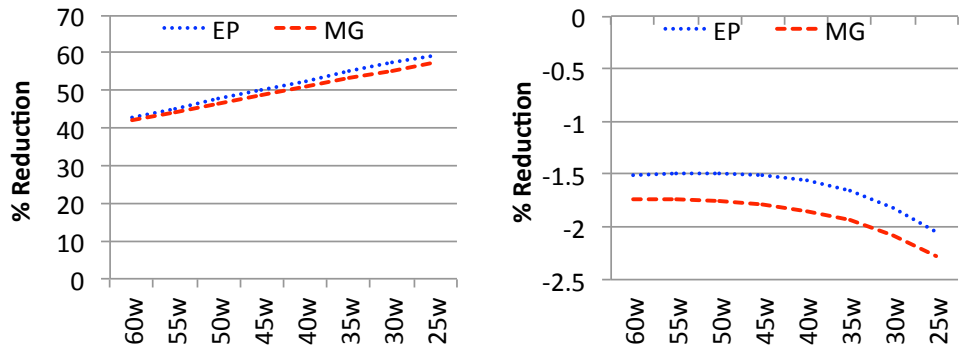
(b) Wasted time



(c) Total execution time

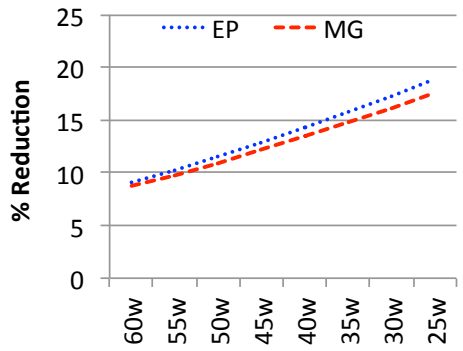
Figure 3.11.: Percentage reduction in time and energy spent in checkpointing, wasted work, and the total time/energy under various power caps and for different cooling platform parameters.

Percentage reduction in execution time and energy consumption for checkpointing, wasted, and total between prior OCI models and the power-capping aware OCI model under various power caps are shown in Fig. 3.11. First, I focus on results for Xeon platform ($C = 0.26$, $D = 38.6$). Comparing with prior models, the power-capping aware model can achieve 9% to 17% reduction in total execution time, and 42% to 57% reduction in checkpointing time. There is a minor increase in waste work but it is offset by significant savings in checkpointing time. Percentage changes for total and checkpointing time increase as power cap



(a) Checkpoint time

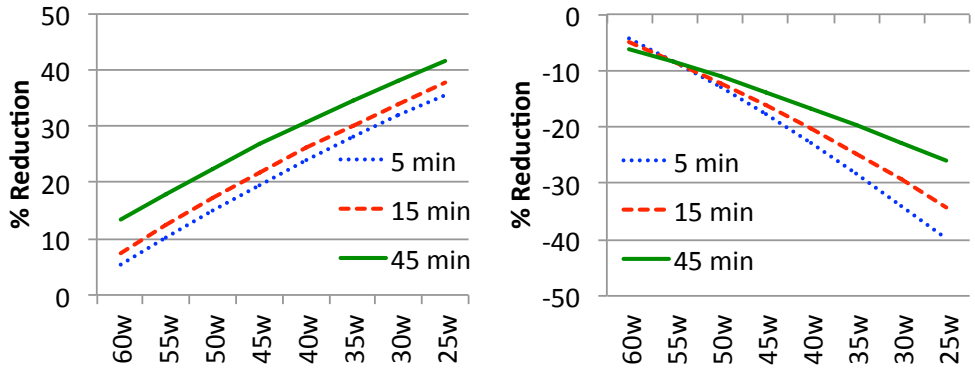
(b) Wasted time



(c) Total execution time

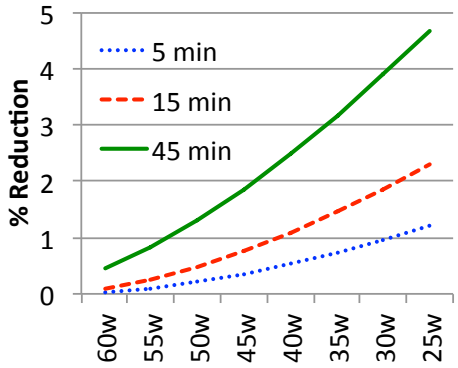
Figure 3.12.: Sensitivity study on application-specific coefficients (parameter A and B). Reduction in time spent in checkpointing, wasted work, and total execution under different power caps.

decreases. Energy consumption also follows similar trend. The reason is that the difference between OCIs derived from prior OCI models and the power-capping aware OCI model is increasing as power cap decreases, as shown in Fig. 3.9. I chose Core i7 platform because it has higher sensitivity to temperature with respect to power consumption (higher value of C parameter, $C = 0.75$, $D = 29.1$). I find that platforms with higher temperature gradient with respect to power capping benefit significantly more by applying power-capping aware OCI model (Fig. 3.11), and their corresponding OCI is also significantly different that can



(a) Checkpoint time

(b) Wasted time

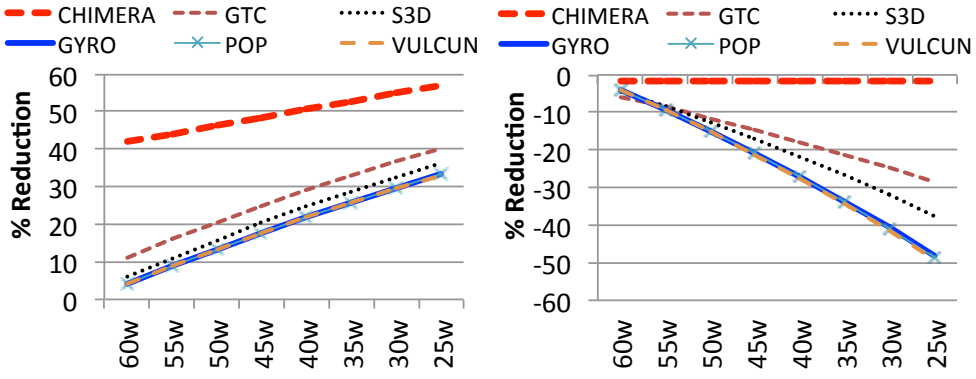


(c) Total execution time

Figure 3.13.: Sensitivity study on time to checkpoint (β). Reduction in time spent in checkpointing, wasted work, and total execution under different power caps.

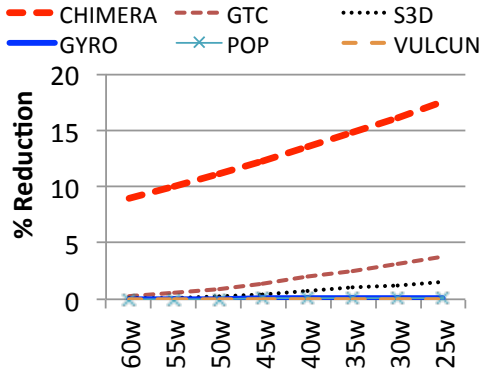
be obtained from the power-capping aware model. Overall, the power-capping aware OCI also results in reduction in the checkpoint time.

Reducing the number of checkpoints can relieve the burden on the storage system of an HPC system which is a shared and constraint resource. Therefore, power-capping aware OCI may in return improve the overall I/O performance of the whole system and other applications.



(a) Checkpoint time

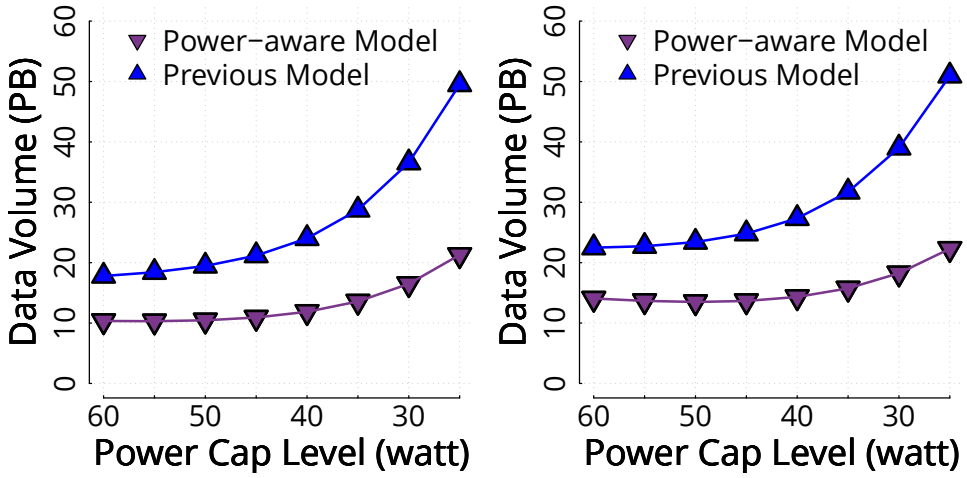
(b) Wasted time



(c) Total execution time

Figure 3.14.: Percentage reduction in time spent in checkpointing, wasted work, and total execution under different power caps for several leadership applications.

Next, I study the sensitivity toward application specific parameters (A and B) that represent the impact of power capping on application performance (Section 3.2). I perform experiments with the applications that have regression functions in Fig. 3.1(b) at both extremes (i.e., MG and EP). Reduction in time spent on checkpointing, wasted, and total execution is shown in Fig. 3.12. From the figure we can observe that the application specific parameters do not have a significant impact on the improvements of power-capping aware OCI. This is expected because parameter A and B do not directly impact the OCI estimation



(a) OCI optimized for time

(b) OCI optimized for energy

Figure 3.15.: Checkpoint data volume of application CHIMERA under different power caps.

α_t^+ and α_e^+ (as noted earlier in the modeling section). Additionally, the OCI is same for MG and EP. Note that slight difference in the percentage reduction is because the execution time still depends on the parameter A and B.

Finding 9 *When comparing the power-capping aware OCI model with prior OCI models, percentage changes on checkpointing, wasted, and total execution time are not highly sensitive to the application-specific coefficients. Additionally, the platforms with higher temperature gradient with respect to power capping benefit significantly more by applying power-capping aware OCI model.*

Next, I also perform a sensitivity study on the time-to-checkpoint. I present experimental results for β equals to 5, 15, and 45 minutes. Fig. 3.13 shows that reduction in checkpointing, wasted, and total execution time increases when time to checkpoint increases. This reduction is even more pronounced for lower power caps and shows significant reduction in checkpoint time. This indicates that applying the power-capping aware model can reduce the checkpointing, and total execution time more significantly when time to checkpoint is larger.

Finding 10 *Power-capping aware OCI model has increasing gains over prior OCI models as the time to checkpoint increases. With increasing system/problem scales and relatively slow growing I/O bandwidth, the power-capping aware model can obtain increasing benefits in I/O bandwidth constrained systems.*

Finally, I perform the evaluations based on the checkpoint data size and execution time of leadership applications run on OLCF machines [1, 2], as shown in Table 3.1. Checkpointing time is the quotient of checkpoint data size divided by average PFS bandwidth. These leadership applications utilize application level checkpointing instead of system level checkpointing. Their checkpointing time does not necessarily scale up with problem size, and is user-specific.

I keep the assumptions in this section except the checkpointing time obtained from BLCR. I use average 10GB/s bandwidth as obtained from Spider parallel filesystem (PFS) attached to the the Titan supercomputer [81] to calculate checkpointing time. Percentage changes for checkpointing, wasted, and total execution time between prior OCI models and the power-aware OCI model under different power caps are shown in Fig. 3.14. Although results for reduction in energy consumption are not shown, energy consumption follows similar trends as execution time, as observed in Fig. 3.11.

As shown in Fig. 3.13, the checkpoint time has significant impact on the savings achieved by power-capping aware OCI. Similarly, we can see that applying the power-aware OCI model to CHIMERA reduces the total execution time by 9% to 18% compared to prior OCI models because it has large checkpoint data size. Applications such as GTC and S3D have moderate checkpoint data sizes. Total execution time decreases by 4% and 2% respectively, when applying the power-aware model to GTC and S3D. For applications with small checkpoint data sizes, i.e., GYRO, POP, and VULCUN/2D, the power-aware model has about the same total execution time as prior models.

Finding 11 *Using the power-capping aware OCI model, applications with large checkpoint data size can achieve substantial reduction on checkpointing time and total execution time over prior OCI models.*

Although the power-capping aware OCI model has limited benefits in terms of total execution time when application checkpoint data size is small, it can still significantly reduce the total volume of checkpoint data being written to storage systems. For GTC and S3D, the power-capping aware model can reduce the checkpoint data volume by up to 40% and 36% respectively. Even for applications such as GYRO, POP, and VULCUN, the power-capping aware model can still reduce the checkpoint data volume by up to 33%. This means that less checkpoints are written to the storage system, which helps resolving the PFS bottleneck problem, and improving application and checkpointing I/O performance. Checkpointing data volume of CHIMERA for all four OCI schemes are shown in Fig. 3.15. The power-capping aware OCI reduces the checkpoint volume by 57% for 25 watts power cap level, compared to the prior models. The gap between the power-capping aware OCI model and the first order model increases when the power cap decreases.

CHAPTER 4
RELIABILITY-AWARE PLATFORM CONFIGURATION FOR HPC WORKFLOWS

In this chapter, I discuss the HPC workflow and reliability-aware platform configuration. The background and model overview are introduced in section 4.1. In section 4.2, I characterize HPC workflows in terms of processor heterogeneity, instruction composition, and error resilience. The impact of power capping and scaling on workflow performance is studied in section 4.3. The optimal platform configuration model is introduced in section 4.4. I present model validation and model-driven studies in section 4.5.

4.1 Background and Overview

This work is primarily modeled after supercomputer Titan and its data analysis cluster Rhea. The U.S. fastest supercomputer, Titan, consists of 18,688 compute nodes, which is equipped with AMD Opteron 6274 CPUs and Nvidia Tesla K20X GPUs. The analysis cluster Rhea is equipped with Intel Xeon E5-2650 CPUs. The processor choices of this work are not constrained to Titan and Rhea. In recent years, utilizing wimpy processors in HPC has become a popular trend [47]. The evaluation and characterization also cover wimpy processors. The proposed reliability-aware platform configuration model is also applicable to wimpy processors.

It should be noted that performing power-related experiments with HPC workflows directly on Titan and Rhea is not viable, because power measurement and power capping are currently not supported on the platforms. To overcome the platform limitation, on hardware side, I choose platforms that support power

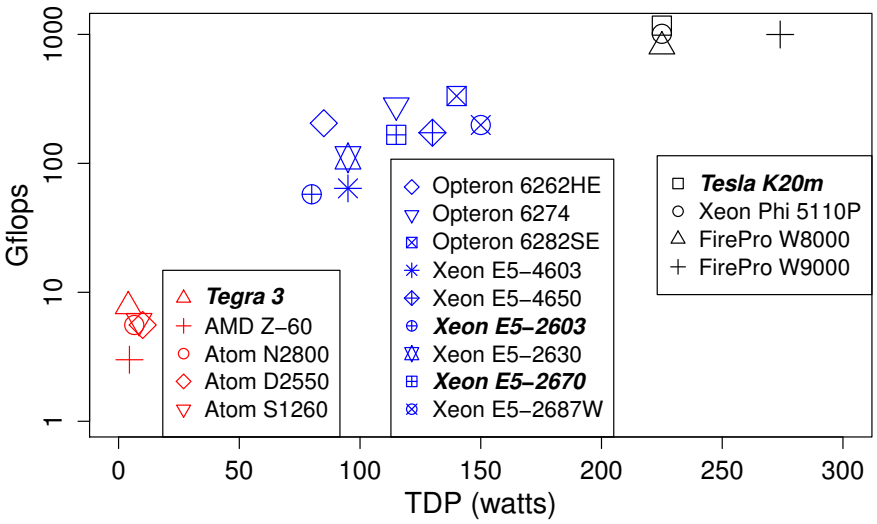


Figure 4.1.: Mainstream processors from the same generation as supercomputer Titan. The y-axis is plotted with a log scale in order to clearly show 3 different orders of magnitude in a single graph. The 5 processors in bold and italic fonts are chosen to represent 3 groups of heterogeneous processors.

Table 4.1.: Platform hardware and software details

Platform	Intel Xeon E5 2670	Intel Xeon E5 2603	Nvidia Tegra 3	Nvidia Tesla K20
Memory	32GB	32GB	2GB	5GB
OS	Linux 2.6.32	Linux 2.6.32	Linux 3.1.10	Linux 2.6.32
Compiler	GCC 4.4.7	GCC 4.4.7	GCC 4.6.3	CUDA 5.5
Cryptonym	XeonH	XeonL	ARM	GPU

capping from the same generation as Titan and Rhea. Mainstream processors from this generation are listed in Fig. 4.1, which are clustered into three categories: wimpy CPUs, brawny CPUs, and coprocessors. One high-end CPU, Intel Xeon E5 2670 (XeonH), and one low-end CPU, Intel Xeon E5 2603 (XeonL), are chosen for evaluation in the category of brawny CPUs. In the category of wimpy

Table 4.2.: Benchmark domain and problem size (Psize).

	Program	Domain	Psize	
Rodinia	Kmeans	Data Mining	256K	Analysis
	BFS (Breadth-First Search)	Graph Algorithms	8M	
	NN (k-Nearest Neighbors)	Data Mining	5M	
	B+tree	Search	10M	
	SC (Streamcluster)	Data Mining	64K	
	LUD (LU Decomposition)	Linear Algebra	2K	Simulation
	LavaMD	Molecular Dynamics	4K	
	CFD (CFD Solver)	Fluid Dynamics	97K	
NPB	LU (Gauss-Seidel)			
	SP (Scalar Pentadiagonal)	Fluid Dynamics	B	
	BT (Block Tridiagonal)			

CPUs, Nvidia Tegra 3 (ARM) is evaluated. In the category of coprocessors, Nvidia Tesla K20 (GPU) is evaluated. The details of the platforms are listed in Table 4.1. The host processor of the GPU platform is Intel Xeon E5-2630. Power and energy consumption for the GPU platform include the host processor and main memory, unless otherwise mentioned. On software side, these platforms cannot host large-scale HPC workflows which depend on platform-specific libraries. Therefore, I employ a wide variety of scientific simulation and analysis applications from Rodinia benchmark suite and NPB benchmark suite (as listed in Table 4.2), to mimic HPC workflows and to study their behaviors under power capping, scaling, and failure events. Analysis algorithms consist of various amounts of filtering and aggregation operations [19]. The five analysis benchmarks, which mainly perform these kinds of operations, represent common analysis tasks on HPC including data mining, search, and graph algorithms. The rest of the benchmarks are simulation programs or kernels derived from

simulation programs that simulate scientific phenomena. These programs represent common HPC simulation tasks such as fluid dynamics, molecular dynamics, bioinformatics, and Linear Algebra.

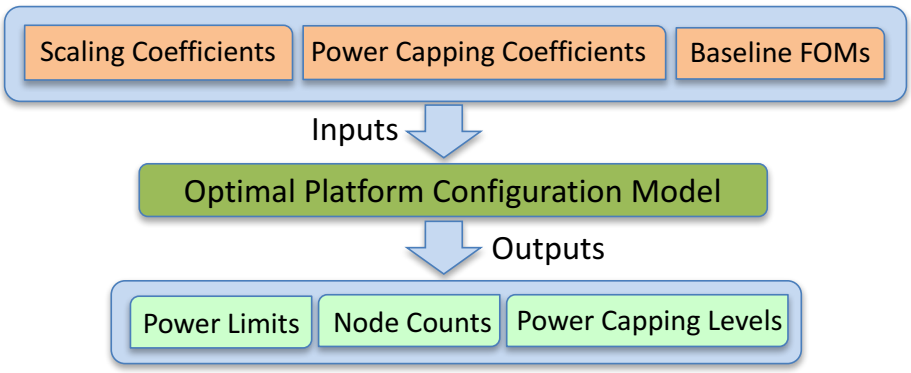


Figure 4.2.: An illustration of how to apply the optimal platform configuration model.

As illustrated in Fig. 4.2, the optimal platform configuration model takes scaling coefficients, power capping coefficients, and baseline figures of merit (FOMs) as inputs. The scaling coefficients (in Eq. 4.1 and Eq. 4.2) and power capping coefficients (in Eq. 4.3 and Eq. 4.4) are introduced in section 4.3, which need to be profiled for each workflow. Baseline figures of merit include execution time, scale, and power consumption, which are obtained when power capping is not enforced. The coefficients and baseline figures of merit can be obtained through offline tests. Based on the model inputs, solving Eq. 4.10 and Eq. 4.20 in section 4.4, we can derive the optimal platform configurations. The model outputs the optimal power limits, node counts, power capping levels, and platform types for simulation and analysis respectively.

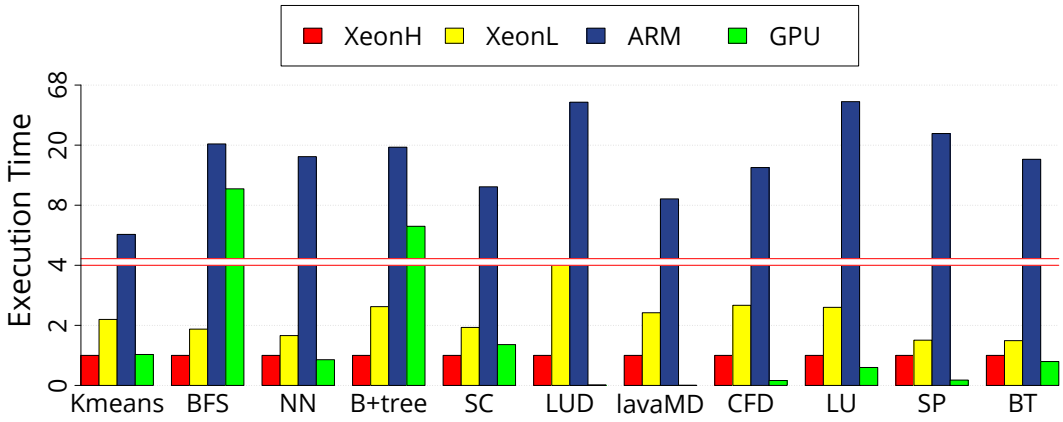
Librapl [24] and NVML [82] are used to profile CPU and GPU power consumption respectively. Intel Power Governor [35] is utilized to cap the package power consumption, and is only supported on the Xeon platforms. To compare the error resilience of simulation and analysis under hardware errors, a software fault injector PINFI [83] is utilized to inject bit flips into programs. PINFI in-

struments the program at runtime, and faults are injected at the assembly code level. The findings may get bounded by the assumptions and parameter settings. To mimic real-world scenarios, the simulation and model-driven studies employ parameters obtained from real-system experiments and large-scale application characteristics obtained from HPC facilities.

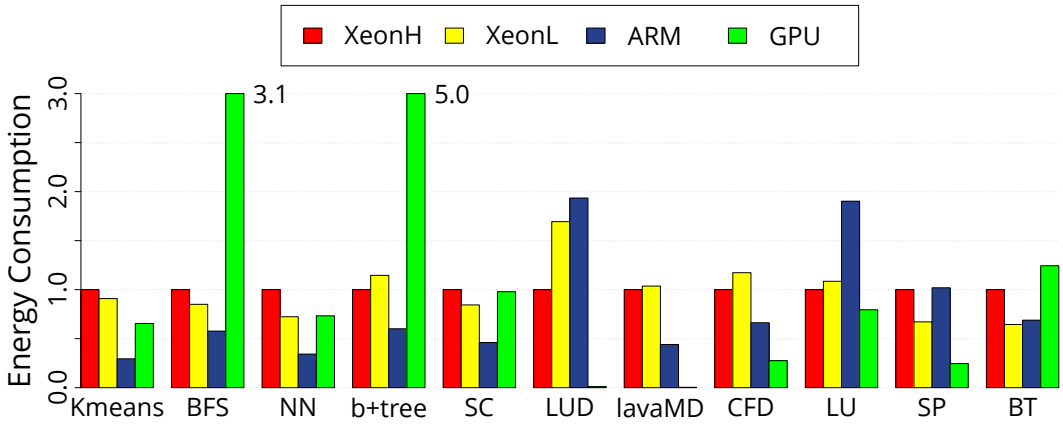
4.2 Characterization of HPC Workflows

In this section, first, I evaluate the performance and energy efficiency of simulation and analysis programs across heterogeneous processors. Then, I revisit whether SoCs are ready for HPC based on the evaluation. Finally, I examine the possibility of relaxing the error resilience requirement in executing HPC workflows through characterizing different behaviors between simulation and analysis programs in terms of their resilience to hardware errors.

Figure 4.3a shows the execution time of all benchmarks. Analysis programs perform better on Xeon and simulation programs perform better on the GPU. Figure 4.3b shows the energy consumption of all benchmarks. All analysis programs are more energy-efficient on ARM. However, some of the simulation programs are more energy-efficient on ARM than on XeonH, and only one simulation benchmark is more energy-efficient on ARM than on the GPU. In terms of energy efficiency, analysis programs behave well on ARM but the simulation programs do not. I have characterized the instruction composition of these applications using TAU profiling tool [73] and PAPI counters [74]. As shown in Figure 4.4, simulation programs have either more floating-point operations (e.g., CFD) or more memory accesses (e.g. LUD) than analysis programs. It is known that wimpy processors (e.g., ARM) have less floating-point units and lower memory bandwidth. Therefore, wimpy processors are more efficient in executing analysis programs that are less floating-point intensive and less memory intensive. In



(a) Execution Time



(b) Energy Consumption

Figure 4.3.: Execution time and energy consumption of simulation and analysis. Execution time (energy consumption) on each platform is normalized to XeonH. Y-axis of 4.3a is presented in a logarithmic scale above 4 and a normal scale below 4, in order to show Xeon and GPU platforms clearly.

contrast, coprocessors are inefficient in executing analysis programs that have more conditional branches and less floating-point operations.

Previous research [47] showed that SoCs are ready for HPC based on performance and energy consumption. However, the above energy efficiency evaluation shows that it is not entirely the case. The previous research did not involve GPU platforms and analysis programs. The results of this work reveal that GPU is

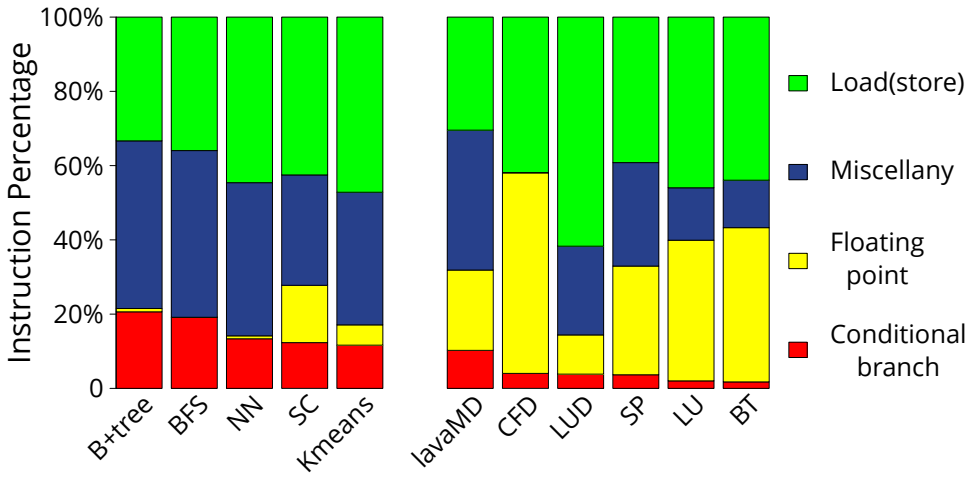


Figure 4.4.: Percentage of load(store), conditional branch and floating-point instructions for each benchmark.

more energy-efficient than ARM for simulation programs, as shown in Figure 4.3. Moreover, scalability becomes more critical when running HPC applications on SoCs [47] [48]. However, CORAL (Department of Energy's consortium of labs for O(100) petaflop machine acquisitions) scalability data [84] shows that, applications such as QBOX, NAMD, SNAP and CAM-SE can only achieve 44% of ideal speedup on average at large scale.

Finding: *In terms of energy efficiency, GPUs perform much better than SoCs for simulation programs, but SoCs are more energy-efficient for analysis programs.*

Finally, I compare the error resilience between simulation and analysis parts of HPC workflows. In the experiments, each benchmark runs for 1,000 iterations with 1 bit-flip fault injected into a random assembly instruction per iteration. Benchmark problem sizes are reduced to guarantee that PINFI instrumentation finishes within a reasonable amount of execution time (less than 80 hours per 1,000 iterations). Comparing the percentage of correct execution, silent data corruption (SDC) and crashes reveals the error resilience capability of each bench-

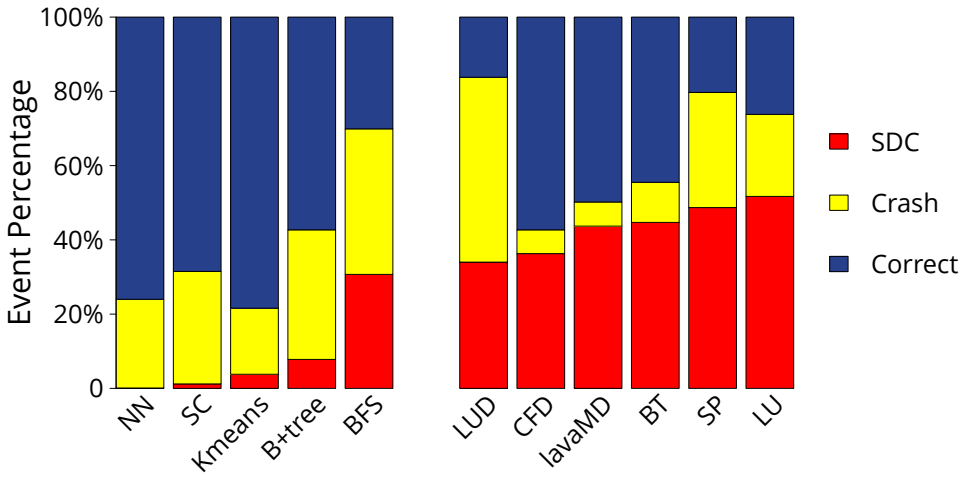


Figure 4.5.: Percentage of correct execution, silent data corruption (SDC) and crash for simulation and analysis.

mark. Correct execution means that a program finishes correctly, and the results are also correct. SDC means that the program finishes correctly but generates incorrect results (output data). Crash is an event when a program either quits with an error or is terminated by the operating system before it finishes correctly.

The results are ordered based on SDC, and are shown in Figure 4.5. Most analysis programs have 60% to 80% correct execution and less than 8% SDC. By comparison, most simulation programs have 16% to 45% correct execution and 30% to 55% SDC. However, BFS behaves similar to simulation, while CFD and lavaMD have less than 7% crashes and more than 50% correct execution.

Finding: *In general, analysis programs are more resilient to SDC, and have more correct execution when compared to simulations. However, resilience to crashes cannot be clearly distinguished between simulation and analysis programs.*

Therefore, it is viable to run the analysis part of a HPC workflow without checkpointing. When a failure occurs, the analysis part just reprocesses the analysis dump. This will be discussed in more detail in section 4.4.2. Additionally, it is also viable to run analysis programs on processors and memory chips

with lower error resilience capability. This can potentially reduce the runtime overheads and building costs introduced by error resilience mechanisms.

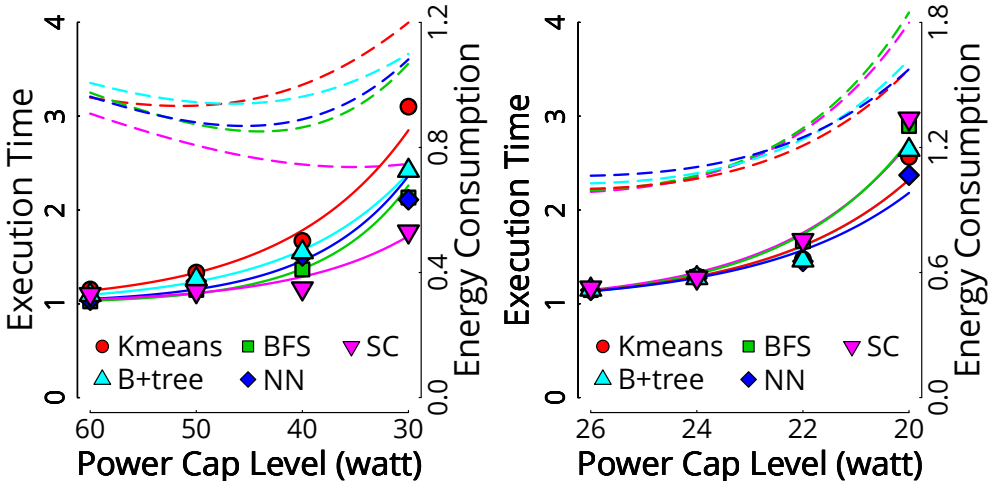
4.3 Power Capping and Scaling

In this section, I statistically quantify the impact of power capping and scaling on execution time and energy consumption for scientific simulation and analysis.

First, I study the computation behaviors of scientific simulation and analysis under power capping. I evaluate the execution time and energy consumption of scientific analysis applications under different power capping levels on Xeon E5-2670. The average power consumption of each benchmark ranges from 63 to 78 watts. An effective power capping level for all benchmarks should be lower than 63 watts. The minimum package power that the power governor can enforce is 22 watts. Therefore, I choose power capping levels of 60, 50, 40, and 30 watts. Similarly, I choose effective power capping levels of 26, 24, 22, and 20 watts on Xeon E5-2603.

The impact of CPU power capping on simulation applications has been previously studied [53] and quantified [18]. In this work, I quantify the power capping impact on analysis applications, which typically have more conditional branches and less floating-point operations compared with simulation applications. The results show that the execution time of analysis applications increases dramatically as power capping level decreases. For example, the execution time on Xeon E5-2670 increases more rapidly when the power capping level reduces from 40 watts to 30 watts, as shown in Fig. 4.6a. The execution time of analysis applications on Xeon E5-2603 (Fig. 4.6b) shows similar increasing trends as on Xeon E5-2670, when power capping level decreases. I quantify the trends utilizing regression analysis, and observe that they can be fitted with exponential functions, as shown in Eq. 4.1.

$$\overline{T_a}(P_i)/T_a = A \times e^{B \times Pi} + 1 \tag{4.1}$$

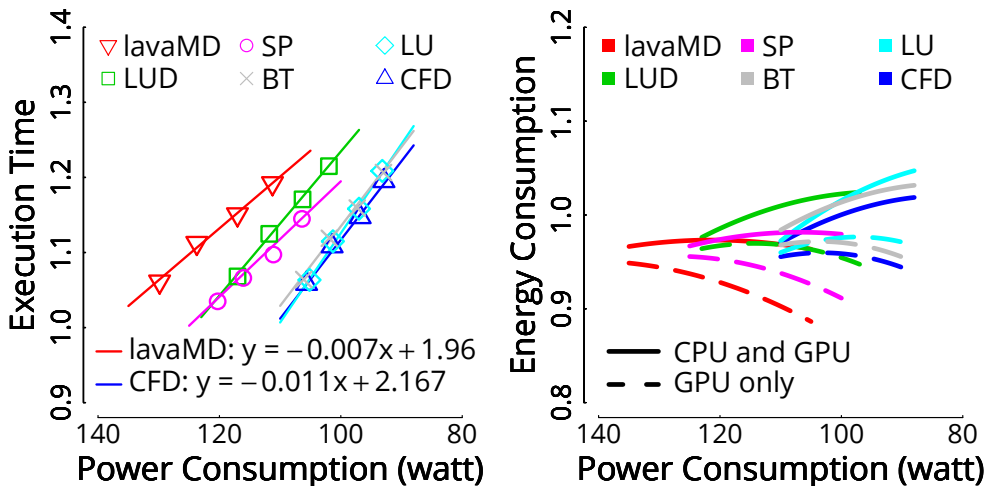


(a) Xeon E5-2670 (b) Xeon E5-2603

Figure 4.6.: Normalized execution time (solid lines) and normalized energy consumption (dotted lines) for analysis applications under power capping on Xeon platforms.

T_a is baseline execution time (without power capping), and $\overline{T_a}(P_i)$ denotes the execution time under power capping level P_i. e is Euler's number. A and B are coefficients of the generalized regression function. The average R-squared value of regression functions for all the benchmarks is 0.96, which indicates statistically sound fit. In Fig. 4.6, energy consumption under various power capping levels are plotted as dotted lines. On Xeon E5-2670, I observe that the optimal power capping level optimized for minimal energy shifts from 50 watts (Kmeans) toward 30 watts (SC), as shown in Fig. 4.6a. By comparison, on Xeon E5-2603, the optimal power capping level optimized for minimal energy stabilizes at the highest power capping level for all benchmarks, as shown in Fig. 4.6b. Consequently, power capping is not applied to Xeon E5-2603 for analysis applications.

Then, I evaluate the execution time and energy consumption under various power levels on GPU. The minimum power limit on GPU is 150 watts, which is too high to perform the power capping experiments. Alternatively, I mimic power



(a) Execution time

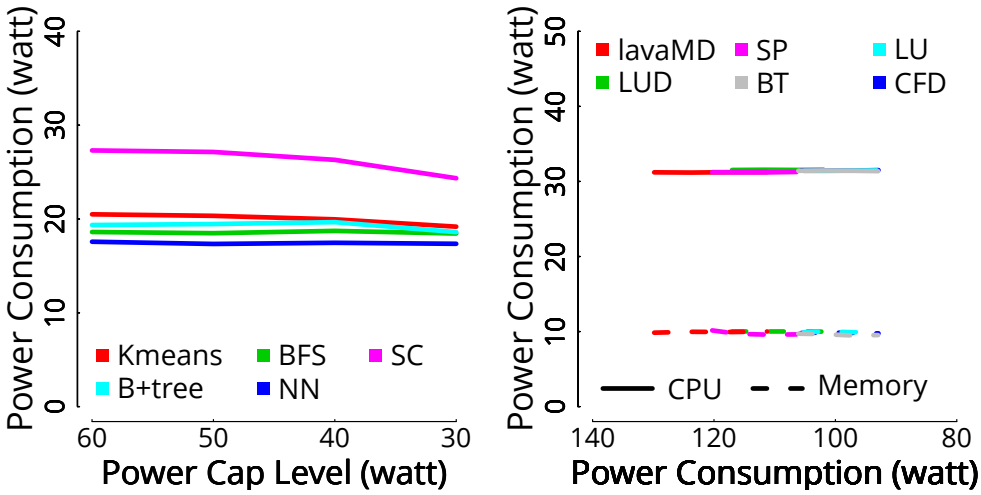
(b) Energy Consumption

Figure 4.7.: Execution time and energy consumption under various GPU power levels. X-axis represents GPU power consumption. Y-axis is truncated for better illustration. Time and energy are normalized to the highest frequency.

capping based on frequency scaling, to obtain various execution time and power consumption pairs. As shown in Fig. 4.7a, execution time increases steadily as power consumption decreases. I quantify the trends with regression analysis, and observe that they can be fitted with linear functions, as shown in Eq. 4.2.

$$\overline{T_s}(P_i)/T_s = C \times Pi + D \tag{4.2}$$

T_s is execution time under maximum power consumption, and $\overline{T_s}(P_i)$ denotes the execution time under power consumption P_i on GPU. C and D are coefficients of the generalized regression function. The R-squared values of regression functions are all above 0.99, which indicates statistically sound fit. Fig. 4.7b shows the energy consumption under various GPU power levels. Energy consumption for only GPU is plotted as dotted lines. As GPU power drops, GPU energy consumption decreases for lavaMD and SP, and keeps relatively steady for the rest of benchmarks. However, in this work, I also consider the host CPU and



(a) CPU memory power

(b) GPU host power

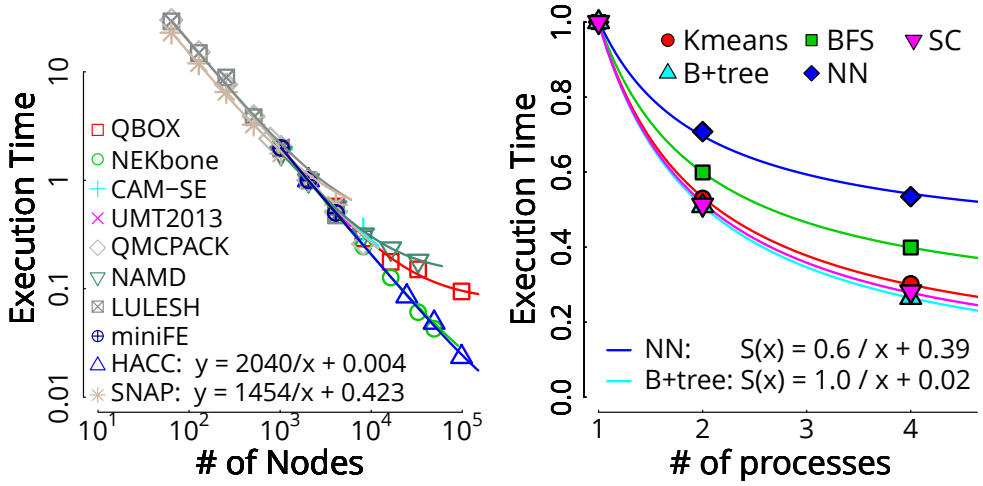
Figure 4.8.: CPU memory power consumption and GPU host power consumption under various power capping levels. X-axis represents CPU power capping level and GPU power consumption respectively in Fig. 4.8a and Fig. 4.8b.

memory as a part of the GPU platform. The total energy consumption of the GPU platform is plotted as solid lines in Fig. 4.7b, which keeps relatively steady for lavaMD and SP, and increases for the rest of benchmarks.

Besides, I also study the impact of power capping on main memory and host power consumption. As shown in Fig. 4.8a, when processor power capping level decreases, the memory power consumption of Xeon E5-2670 decreases slightly for SC and remains steady for the other benchmarks. I profile the memory throughput and find that SC is at least 5 times the memory throughput of other benchmarks, which makes SC memory power consumption more sensitive to power capping. The power consumption of host CPU and memory keeps steady for all benchmarks under various GPU power levels, as illustrated in Fig. 4.8b.

Finding 12 *On GPU platforms, host side (CPU and memory) power consumption is independent of application characteristics and GPU power levels. When*

taking host CPU and main memory into account, running at lower power levels makes GPU less energy-efficient.



(a) Simulation scaling results [84]

(b) Analysis scaling results

Figure 4.9.: Scaling results for simulation and analysis applications. In 4.9a, both time and scale are normalized to the first data point, which are plotted in log scale to clearly show the first few data points. The scaling results are measured on BlueGene/Q and Cray XK7 with a maximum scale of 98,304 nodes, and are normalized to the scale of 2,048 nodes.

Finally, I quantify the scalability of scientific simulation and analysis. In terms of simulation applications, I exploit the CORAL scaling results [84] as shown in Fig. 4.9a. According to Amdahl's law, these data points can be fitted with rational functions as shown in Eq. 4.3.

$$S_s(x) = E/x + F \tag{4.3}$$

$S_s(x)$ is normalized execution time and x is scale (i.e., number of nodes). E and F are coefficients of the generalized regression function. The R-squared values of regression functions are all above 0.99, which indicates statistically sound fit.

With respect to online analysis, a group of analysis processors is usually assigned per unit of simulation nodes. Strong scaling is assumed within the group when processing data from one unit of simulation nodes, and weak scaling among groups when processing global data from the entire simulation. I am aware that there are certain analysis stages that exhibit data dependencies among groups. Analysis applications can be decomposed into a massively parallel stage and a small-scale parallel or serial stage [19]. This work focuses on the massively parallel stage which is usually the main body of scientific analysis. Afterwards, a small scale of processors can be utilized to finish up the task. As shown in Fig. 4.9b, execution time decreases in a sublinear manner for analysis applications when parallelism increases. These data points can be fitted with rational functions as in Eq. 4.4.

$$S_a(x) = G/x + H \tag{4.4}$$

$S_a(x)$ is normalized execution time and x denotes scale (i.e., number of cores). G and H are coefficients of the generalized regression function. R-squared values of these regression functions are all above 0.996.

4.4 Optimal Platform Configuration

In section 4.3, I have quantified the impact of power capping and scaling on execution time for both simulation and analysis. Based on these relationships, I develop a reliability-aware model to predict the optimal platform configurations for HPC workflows, which include power limits, power capping levels, and scales for scientific simulation and analysis respectively. The variables used in the model are listed in Table 4.3.

4.4.1 Base Model

In order to facilitate the derivation of the reliability-aware model, first, I develop the base model in this subsection. Facing power constraints, workflows

Table 4.3.: Symbols in the analytical model. Subscript 's' denotes simulation and subscript 'a' denotes analysis.

Constant	Description	Variable	Description
$N_{s\|a}$	Baseline scale	$n_{s\|a}$	Scale
n_{cores}	Processor core count	$e_{s\|a}$	Energy consumption
$T_{s\|a}$	Baseline execution time	$t_{s\|a}$	Execution time
P_{host}	GPU host power consumption	r_n	Ratio between n_a and n_s
P_{mem}	Memory power consumption	$p_{s\|a}$	Processor and memory
$\overline{P}$	Workflow power limit		power consumption

always run with a power limit ($\overline{P}$). Workflow power consumption comes from processor, memory, and network. Using optical links to replace copper links has become a practical trend in exascale computing, because power consumption of optical links remains steady as bit-rate scales up [23]. Therefore, the network power consumption remains constant and is excluded from power limit $\overline{P}$. The peak power consumption of workflows should always be under the power limit, as defined in formula 4.5.

$$n_s \times p_s + n_a \times p_a \leq \overline{P} \tag{4.5}$$

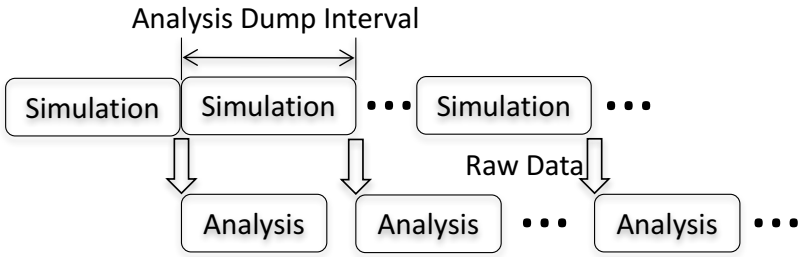


Figure 4.10.: Overview of workflow execution

Simulation and analysis run in a pipelined manner, as illustrated in Fig. 4.10. In HPC workflows, online analysis reduces the amount of simulation data to

persistent storage, which in return expedites the simulation. Therefore, analysis should finish earlier than simulation, rather than impeding simulation. The overall execution time of a workflow ($T_{workflow}$) is dominated by the part (either simulation or analysis) that takes longer time, as defined in Eq. 4.6.

$$T_{workflow} = \text{MAX}(t_s, t_a) \tag{4.6}$$

t_a is analysis execution time and t_s is simulation execution time, which are defined in Eq. 4.7 and Eq. 4.8 respectively. t_a and t_s are calculated based on the baseline execution time T_a and T_s respectively, after scaling and power capping. In the optimal configuration sets, t_a is always smaller than t_s.

$$t_a = S_a(r_n \times n_{cores})/S_a(n_{cores}) \times \overline{T_a}(p_a - P_{mem}) \tag{4.7}$$

P_{mem} denotes the main memory power consumption, which is included in the energy calculation. Baseline analysis time (T_a) is the per-processor (n_{cores}) execution time without power capping, as shown in Eq. 4.1. Baseline analysis scale (N_a) is 1. Scaling factor is calculated based on Eq. 4.4, where r_n analysis processors are utilized to perform the same task as the baseline case.

$$t_s = S_s(n_s)/S_s(N_s) \times \overline{T_s}(p_s - P_{host}) \tag{4.8}$$

P_{host} denotes the host power consumption on the GPU platform, which consists of CPU and main memory power consumption. Host power consumption is included in the energy calculation. Baseline simulation time (T_s) is the execution time under baseline scale (N_s) without power capping, as shown in Eq. 4.2. Baseline simulation time and scale can be obtained from large-scale runs. Scaling factor is calculated based on Eq. 4.3.

The overall energy consumption of a workflow ($E_{workflow}$) consists of energy consumption from the simulation (e_s) and analysis (e_a), as shown in Eq. 4.9.

$$E_{workflow} = e_s + e_a = n_s \times p_s \times t_s + n_a \times p_a \times t_a \tag{4.9}$$

To be noted that network energy consumption is not included in $E_{workflow}$. As discussed earlier, power consumption of optical links remains steady as bit-rate scales up [23], which makes network energy consumption dependent on workflow execution time ($T_{workflow}$). As long as $T_{workflow}$ is minimized, network energy consumption is optimal.

The objective function $F_{(n_s,p_s,r_n,p_a)}$ to achieve the minimal workflow execution time ($T_{workflow}$) is given in Eq. 4.10.

$$F_{(n_s,p_s,r_n,p_a)} = minimize(t_s), \quad t_s \geq t_a \tag{4.10}$$

In order to solve the objective function, we should minimize t_s while ensuring that t_a is smaller than t_s. When analysis variables r_n and p_a are treated as constants in t_s, simulation time (t_s) is minimized when simulation scale (n_s) reaches the upper limit. According to formula 4.5, the upper limit of n_s can be expressed as a function of variable p_s, as shown in formula 4.11.

$$n_s \leq \overline{P}/(p_s + r_n \times p_a) \tag{4.11}$$

Based on formula 4.11, the minimum t_s is converted into a quadratic function of the variable p_s, as shown in Eq. 4.12.

$$t_s = S_s(\overline{P}_{/(p_s+r_n\times p_a)})/S_s(N_s) \times \overline{T_s}(p_s-P_{host}) \tag{4.12}$$

Solving the quadratic function (Eq. 4.12), the x coordinate of the vertex (i.e., maximum t_s) is defined in Eq. 4.13.

$$x = -0.5 \times (D/C - P_{host} + r_n \times p_a + F/E \times \overline{P}) \tag{4.13}$$

As a result, p_s for minimal simulation execution time (t_s) is defined in Eq. 4.14.

$$p_s = \begin{cases} P_{max}, & \text{if } x \leq (P_{max} + P_{min})/2; \\ P_{min}, & \text{if } x > (P_{max} + P_{min})/2. \end{cases} \tag{4.14}$$

P_{max} and P_{min} denotes the maximum and minimum effective power capping level for simulation. In the solution above, analysis variables r_n and p_a are treated

as constants at first, to derive the p_s for minimal t_s. In resolving the objective function $F(_{n_s,p_s,r_n,p_a})$, next, I enumerate the combinations of r_n and p_a to obtain the minimal t_s that satisfies the condition $t_s \geq t_a$.

4.4.2 Reliability-aware Model

In subsection 4.4.1, I develop the base model to predict the optimal platform configurations for a failure-free environment. In reality, it is common to see failures happen, especially in large-scale HPC facilities. In this subsection, first, I revisit how the power-capping aware checkpointing model [18] calculates failure rate, wasted time, and OCI (Optimal Checkpointing Interval), as shown in Eq. 4.15 to Eq. 4.19. Then, I propose a reliability-aware model based on the base model, to consider failures in the real world.

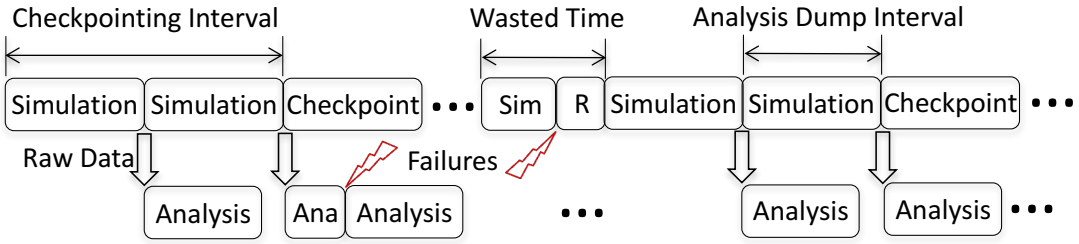


Figure 4.11.: Overview of workflow execution with failures, checkpoint, and restart

Taking failures, checkpoint, and restart into account, the total execution time of scientific applications ($\overline{T_{total}}(P_i)$) includes computation time ($\overline{T_{comp}}(P_i)$), checkpointing time ($\overline{T_{chkp}}(P_i)$), and wasted time ($\overline{T_{waste}}(P_i)$), as illustrated in Fig. 4.11. Computation time is the time to perform scientific simulation (t_s) and data analysis (t_a). In order to tolerate failures, simulation applications perform checkpointing periodically with an interval α. The analysis dump interval is independent of the checkpointing interval. Time to checkpoint (denoted as β) is the time taken to store checkpoints into permanent storage systems. When failures occur, the

simulation application rollbacks to the latest checkpoint, or the analysis application reprocesses the raw data. Additional checkpoints are not taken for analysis applications because analysis is more resilient to failures as illustrated in section 4.2. The lost work caused by such rollback is wasted time. The checkpointing time and wasted time are defined in Eq. 4.15 and Eq. 4.16 respectively.

$$\overline{T_{chkp}}(P_i) = (\frac{\overline{T_{comp}}(P_i)}{\alpha} - 1) \times \beta \tag{4.15}$$

$$\overline{T_{waste}}(P_i) = \frac{\overline{T_{comp}}(P_i)}{\alpha} \times (e^{\frac{\alpha+\beta}{\overline{MTBF}(P_i)}} - 1) \times (\epsilon \times (\alpha + \beta) + \epsilon) \tag{4.16}$$

ϵ represents the fraction of lost work due to failures. α denotes OCI, which is derived by solving $\frac{d}{d\alpha}(\overline{T_{total}}(P_i)) = 0$, leading to the minimal total execution time ($\overline{T_{total}}(P_i)$). In this work, in order to accurately predict OCI under all conditions, I utilize the MATLAB numeric solver "vpasolve" to calculate OCI.

In Eq. 4.16, $\overline{MTBF}(P_i)$ represents the MTBF (Mean Time Between Failures) at power capping level P_i, which is defined in Eq. 4.17.

$$\overline{MTBF}(P_i) = (MTTF_{base}/F_A(\overline{TEMP}(P_i)) + \gamma) \times N_s/n_s \tag{4.17}$$

$MTTF_{base}$ denotes the baseline mean time to fail. γ is the time to recover from failures. $F_A(x)$ is the acceleration factor of the Arrhenius Equation at temperature x, as defined in Eq. 4.18.

$$F_A(x) = e^{\frac{E_a}{k} \times (1/TEMP_{base} - 1/x)} \tag{4.18}$$

k is Boltzmann constant (8.617×10^{-5} eV/°K) and E_a is activation energy (0.7eV). $TEMP_{base}$ denotes the baseline temperature. In Eq. 4.17, $\overline{TEMP}(P_i)$ denotes processor temperature at power capping level P_i, which is defined in Eq. 4.19.

$$\overline{TEMP}(P_i) = I \times P_i + J \tag{4.19}$$

The sensitivity of processor temperature to changes in power capping (i.e., coefficients I and J) is dominated by the cooling infrastructure. Processor temperature increases linearly as power capping level grows, as illustrated in Eq. 4.19. When

processor temperature gets higher, it becomes less reliable, as illustrated in Eq. 4.17.

I propose the reliability-aware model to explore various configuration sets and minimize the total execution time ($\overline{T_{total}}(P_i)$). Rather than minimizing the simulation time t_s as in the base model, the reliability-aware model minimizes the total execution time of simulation, which includes computation time, checkpointing time, and wasted time. Reliability plays a major role in the trade-off between checkpointing time and wasted time. In addition, under power constraints, power capping level and scale affect both reliability and computation time inversely. For example, an increment in power capping level is accompanied by decreased reliability level and shortened computation time. Under a power limit, this causes a decrement in scale, which occurs with increased reliability level and prolonged computation time. The reliability-aware model explores the various interplay among power capping level, scale, reliability, computation time, checkpointing time, and wasted time. The objective function is defined in Eq. 4.20.

$$F(_{n_s,p_s,r_n,p_a}) = minimize(\overline{T_{total}}(_{p_s})), \quad t_s \geq t_a + \widetilde{t_a} \tag{4.20}$$

$\widetilde{t_a}$ is the wasted time of analysis applications. In order to avoid the interference between consecutive analysis tasks, the summation of analysis time (t_a) and analysis wasted time ($\widetilde{t_a}$) should not exceed the simulation time (t_s).

Total execution time ($\overline{T_{total}}(_{p_s})$) is determined by $\overline{T_{comp}}(_{p_s})$ and α, both of which can be expressed as a function of power capping level p_s. In solving the objective function, I utilize the same methodology as discussed in subsection 4.4.1. The difference lies in how to derive the optimal p_s to achieve minimal $\overline{T_{total}}(_{p_s})$. Solving $\frac{d}{dp_s}(\overline{T_{total}}(_{p_s})) = 0$ leads to the optimal power capping level p_s. In this work, the MATLAB numeric solver "vpasolve" is utilized to calculate the optimal configurations.

4.5 Evaluation and Model-driven Studies

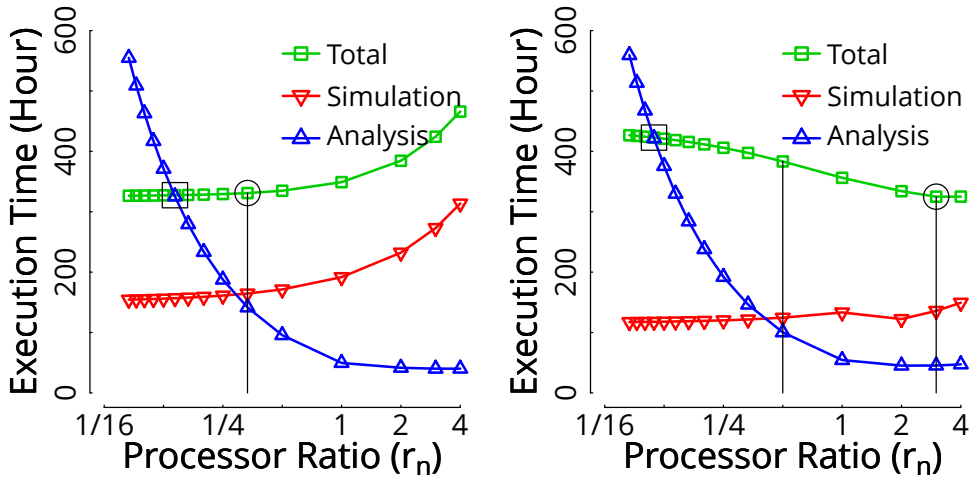
In this section, I evaluate the proposed model in terms of scale, power capping level, and power limit. In addition, I also study the sensitivity of the reliability-aware model to various parameters, compared with the base model.

In real world, n_s and r_n have facility-dependent boundaries. p_s and p_a have platform-dependent boundaries. In solving the objective function $F(_{n_s,p_s,r_n,p_a})$, I enumerate the possible values of r_n and p_a since they have small ranges. In this way, the 4-tuple (n_s, p_s, r_n, p_a) can be converted into 2-tuple (n_s, p_s), which can be further converted into 1-tuple p_s based on Eq. 4.11. Then I follow the steps as discussed above to solve the objective function for each r_n-p_a pair. Finally, I traverse the enumeration set to obtain the optimal solution.

In the baseline setup, power consumption and coefficients are the average of measured data across all the applications on real platforms. The baseline setup assumes a 120 hour job (scientific simulation) running on a Titan-like supercomputer using 20,000 nodes. It is assumed that this scientific simulation produces raw data at 1 TB/s, which equals to the peak I/O bandwidth of the Spider Lustre filesystem at Oak Ridge National Laboratory. This is a valid assumption since not all the data will reach the Lustre filesystem after online analysis. The job consumes 120 watts power on each GPU, with an upper (P_{max}) and lower (P_{min}) power limit of 120 watts and 90 watts respectively. Online analysis is performed on a separate group of Xeon processors, whose baseline per-processor analysis throughput is 140 MB/s. Per-processor analysis throughput is chosen based on the geometric mean instead of arithmetic mean, due to the significant variation across different applications. In the baseline setup, analysis time is calculated as simulation time multiplied by the throughput ratio between simulation and analysis.

When running simulation applications on CPUs, reducing active cores relieves memory contention and improves performance [24]. Unlike CPU applications, the

parallelism of GPGPU applications is orders of magnitude larger than the available CUDA cores. GPGPU is designed to serve such huge parallelism. Therefore, it is not viable to reduce the cores in exploring the optimal configuration.



(a) Power limit 2 MW

(b) Power limit 6 MW

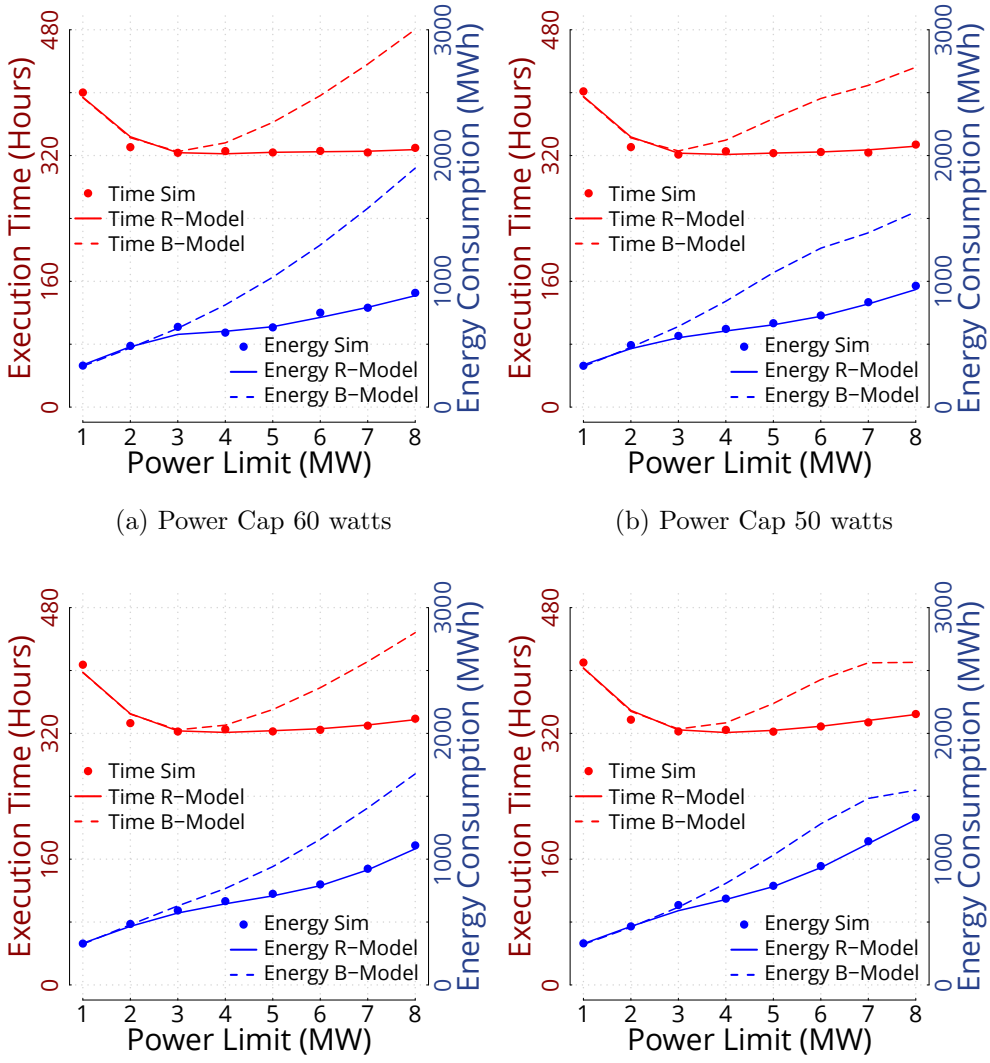
Figure 4.12.: Minimal total simulation time ($\overline{T_{total}}(P_i)$), and associated workflow computation time t_s (simulation) and t_a (analysis), with various processor ratios and under analysis power capping level 60 watts. They show similar trends under different analysis power capping levels. X-axis is plotted in a log scale to clearly show the data points when $r_n < 1$.

Fig. 4.12 shows the minimal total simulation time ($\overline{T_{total}}(P_i)$) which includes simulation computation time, checkpointing time, and wasted time. Processor ratio refers to the number of analysis processors used to process the data generated by one simulation processor. Under a power limit of 2 MW, as shown in Fig. 4.12a, both total simulation time and simulation computation time increase as processor ratio increases. With power under-provisioned, increasing the number of analysis processors means decreasing the power limit of simulation, which leads to the decreased simulation scale and increased simulation time. The minimal total simulation time is marked with a circle when analysis time ($t_a + \widetilde{t_a}$) becomes

slightly smaller than simulation computation time (t_s). Under a power limit of 6 MW, as shown in Fig. 4.12b, although simulation computation time increases slightly, the total simulation time deceases as processor ratio increases. The reason is that, with power over-provisioned, smaller simulation scale and lower power capping level result in a higher reliability level (or larger MTBF), which in return reduces the total checkpointing time and wasted time. Consequently, the total simulation time decreases as analysis scale increases. The minimal total simulation time is achieved (marked with a circle) when analysis time ($t_a + \widetilde{t_a}$) becomes much smaller than simulation computation time (t_s), rather than the point where $t_a + \widetilde{t_a}$ becomes slightly smaller than t_s. Running a workflow under power constraints, it is intuitive to assign less power to analysis (letting analysis match the simulation time), and distribute more power to simulation in order to boost performance. However, this can raise the checkpointing time and wasted time, when reliability is taken into account.

Finding 13 *Distributing excessive power to simulation can prolong the total simulation time. Matching analysis and simulation in terms of execution time does not always lead to the minimal total time.*

To be noted that, extending the upper limit of analysis time from simulation computation time (t_s) to total simulation time ($\overline{T_{total}}(P_i)$) can reduce the total simulation time by 1.1%, marked with a square in Fig. 4.12a. However, the reliability-aware model does not treat such configuration as optimal for the following two reasons. First of all, the performance gain is limited. However, it may cause significant performance loss as illustrated in Fig. 4.12b. In addition, extending the analysis time causes interferences between consecutive analysis tasks. For example, an application performs checkpointing every α time, the time taken to store a checkpoint is β, and analysis data dump frequency is α/n. If analysis takes up to $(\alpha + \beta)/n$ time, when the second analysis dump starts at time α/n, the analysis resources are still busy processing the first data dump. To be worse, such delay propagates to the consecutive $n - 1$ analysis tasks.



(a) Power Cap 60 watts

(b) Power Cap 50 watts

(c) Power Cap 40 watts

(d) Power Cap 30 watts

Figure 4.13.: Minimal workflow execution time and energy consumption under various power limits. Solid lines represent the reliability-aware model (R-Model) and dotted lines are for the base model (B-Model). Solid dots are simulation results. Color red is utilized to plot execution time (left y-axis) and blue is used for energy consumption (right y-axis).

In Fig. 4.12, I show the minimal workflow execution time under power limits 2 MW and 6 MW. Next, I validate the reliability-aware model and explore

the minimal execution time and energy consumption under various power limits (ranging from 1 MW to 8 MW) for HPC workflows.

In order to validate the reliability-aware model, the model results are compared against simulation results. I utilized an event-driven simulator developed in [12], which simulates different phases (i.e., computation, checkpointing, and rollback) and failure events of simulation and analysis in HPC workflows. The simulator generates random failures which follow a Poisson process, and the intervals between failures follow the exponential distribution. In the simulation, execution time is adjusted based on the power capping level and scale in accordance with the relationships derived in section 4.3.

Simulation results for both execution time (red dots) and energy consumption (blue dots) are shown in Fig. 4.13. We can observe that the reliability-aware model results (solid lines) closely match the simulation results under various power limits and platform configurations. I also show the results of the base model in Fig. 4.13 to compare with the reliability-aware model. First, I observe that these two models are close to each other when power limit is small (smaller than 3 MW). As power limit increases, applying the base model results in larger execution time and energy consumption, and the gap is enlarging. Finally, I observe that workflow execution time decreases first, keeps steady, and then increases slightly as power limit increases. Therefore, the optimal power limit for performance is the starting point of the steady phase. Concerning workflow energy consumption, it increases significantly as power limit increases.

Finding 14 *The reliability-aware model can accurately predict the optimal platform configurations for minimal workflow execution time and energy consumption. Energy consumption increases as power limit increases, but an excessive high power limit may degrade the workflow performance.*

Although processor ratio strongly affects the workflow execution time (as illustrated in Fig. 4.12), different (r_n, p_a) pairs (e.g., $(60, 1/3)$, $(50, 1/3)$, $(40, 1/2)$ and $(30, 1)$ under 2 MW power limit) lead to similar minimal total execution time,

as shown in Fig. 4.13. However, using the lowest analysis power capping level requires 3 times analysis processors as using the highest power capping level. Moreover, as shown in Fig. 4.13, employing a lower analysis power capping level results in higher workflow energy consumption, especially with a larger power limit. Although a lower power capping level tends to improve the reliability level, it improves the analysis scale, which in return reduces the overall MTBF. However, in Fig. 4.6a, I observe that, to achieve the minimal energy consumption, the optimal analysis power capping levels for most applications are between 40 watts and 50 watts. In contrast, when taking scalability and reliability into account, a higher analysis power capping level typically leads to lower energy consumption.

Finding 15 *The combination of analysis power capping level and scale offsets their individual impact on execution time within a range. The optimal analysis power capping level should be the highest effective level. Otherwise, it leads to resource over-utilization and excessive energy consumption.*

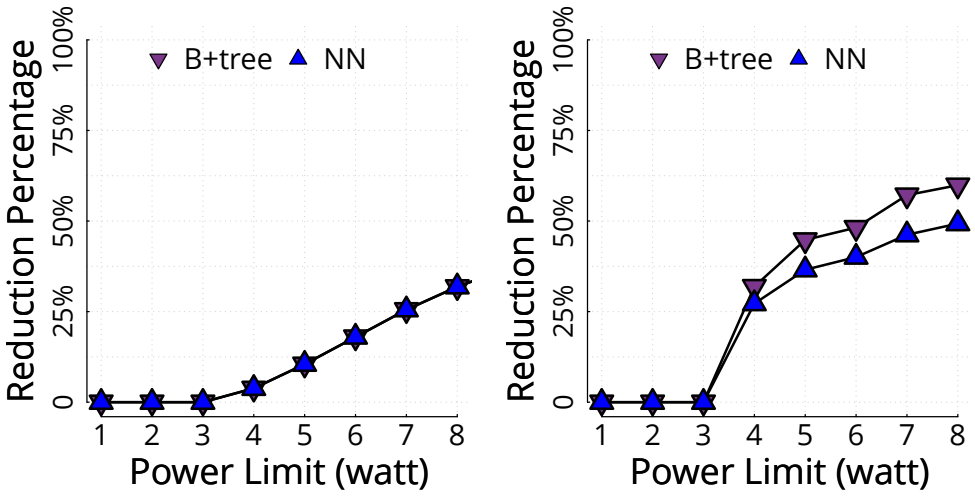
To be noted that, under power constraints, the platform configuration for minimal execution time does not necessarily lead to minimal energy consumption, as illustrated in Fig. 4.13. Under power limits, execution time reflects the performance per power limit. Conventionally, energy efficiency is measured as performance per watt, which can be reflected by energy consumption.

Finding 16 *Execution time and energy consumption show distinct trends as power limit changes. Under power constraint, performance per power limit does not reflect real energy efficiency which is typically measures as performance per watt.*

While determining the optimal power limit, users usually pursue minimizing execution time. In contrast, facility owners typically want to minimize the electricity bills (energy consumption). The different goals result in a dilemma in deciding the optimal power limit. To resolve this, I resort to the metric, energy-

delay product (or performance per joule), when perform comparison among various workflow power limits, in order to achieve the trade-off between different goals.

The optimal platform configuration model is based on parameters obtained from real-system experiments. Consequently, the results are dependent on such parameters, which may compromise the findings or conclusions. Therefore, I perform sensitivity studies for the reliability-aware model on such parameters. In sensitivity study, reduction percentage is utilized to measure the percentage of time or energy reduced by applying the reliability-aware model compared with the base model.



(a) Execution time

(b) Energy consumption

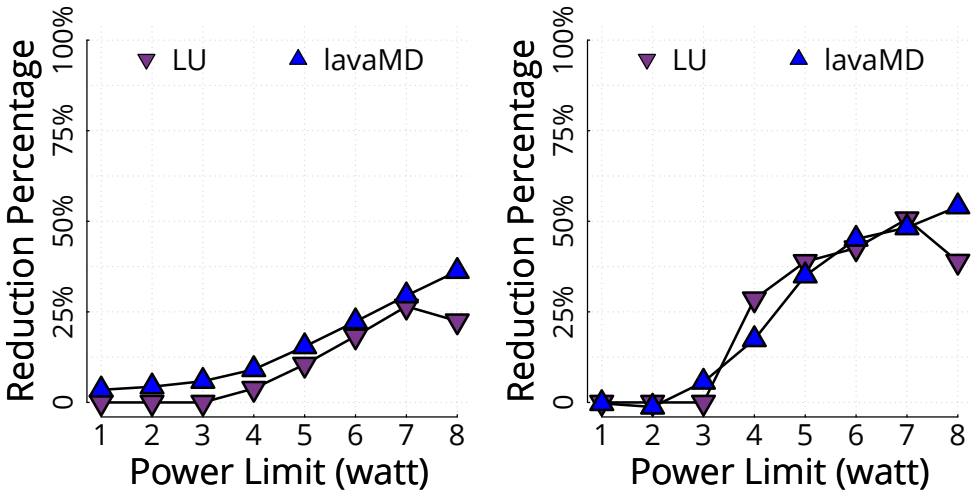
Figure 4.14.: Reduction percentage of execution time and energy consumption for sensitivity study on analysis scaling coefficients.

According to finding 15, analysis power capping level and scale neutralizes each other's impact on execution time. Therefore, the reliability-aware model is not sensitive to the variation in these coefficients. In Fig. 4.14, I compare the borderline analysis programs in terms of scalability (coefficients G and H), to illustrate the sensitivity of the reliability-aware model to analysis scaling coeffi-

cients. As shown in Fig. 4.9b, application B+tree and NN have the best and worst scalability. I find that the reliability-aware model shows identical reduction percentages over the base model for B+tree and NN under all power limits, as shown in Fig. 4.14a. Concerning energy consumption, the reduction percentage only grows faster for B+tree when power limit gets larger than 3 MW. As power limit keeps increasing, the energy reduction difference between the borderline coefficients is up to 10.6%. Regarding analysis power capping coefficients (A and B), I have performed sensitivity study with borderline analysis applications Kmeans and SC (as shown in Fig. 4.6a), and find that the reliability-aware model also shows similar behaviors as the base model with respect to reduction percentages.

Finding 17 *The advantage of applying the reliability-aware model over the base model is not sensitive to the variations in analysis power capping and scaling coefficients.*

Then, I perform sensitivity studies on coefficients that are relative to simulation. The sensitivity study on simulation power capping coefficients (C and D) is based on two boundary applications (lavaMD and LU). As shown in Fig. 4.7a, the performance of application LU is more sensitive to changes in power capping levels, compared with application lavaMD. I observe that the reduction percentage for lavaMD is slightly higher than the reduction percentage of LU, as shown in Fig. 4.15. For applications (e.g., lavaMD) that are less sensitive to power capping changes, the reliability-aware model explores the opportunities to reduce the simulation power capping level with increased scale, in order to improve reliability level and minimize the execution time. Overall, simulation power capping coefficients have small impact on the reduction percentage of the reliability-aware model against the base model. The reduction percentages on both execution time and energy consumption increase as power limit gets larger, as shown in Fig. 4.15.



(a) Execution time

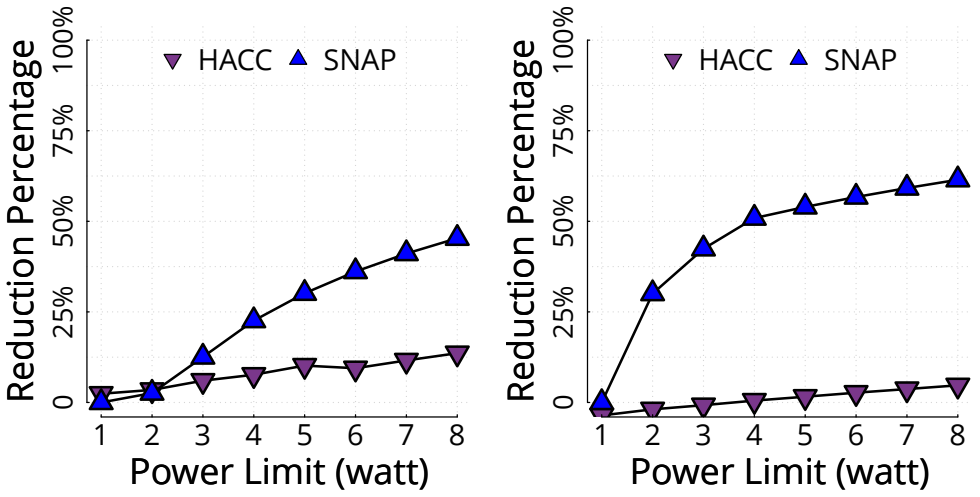
(b) Energy consumption

Figure 4.15.: Reduction percentage of execution time and energy consumption for sensitivity study on simulation power capping coefficients.

The sensitivity study on simulation scaling coefficients (E and F) is based on two boundary applications, HACC and SNAP, which has the highest and lowest scalability as shown in Fig. 4.9a. I find that these scaling coefficients have significant impact on the reduction percentage. As shown in Fig. 4.16a, when simulation scalability is low (i.e., SNAP), applying the reliability-aware model reduces the execution time by up to 45.3%, which is 31.7% higher than running with coefficient of HACC. Concerning energy consumption, the reduction percentage gap between borderline applications is up to 56.7%. When the scalability of simulation applications is low, the reliability-aware model explores the trade-off between reliability and scale. In such circumstance, although running at a smaller scale increases the computation time, it increases MTBF and reduces the total checkpointing time and wasted time, which in return reduces the total simulation time and energy consumption.

Finding 18 *The advantage of applying the reliability-aware model over the base model is not sensitive to the variations in power capping coefficients, but is sensi-*

tive to the simulation scalability. When simulation suffers from poor scalability, the reliability-aware model explores the trade-off between reliability and scale while minimizing the execution time and energy consumption.



(a) Execution time

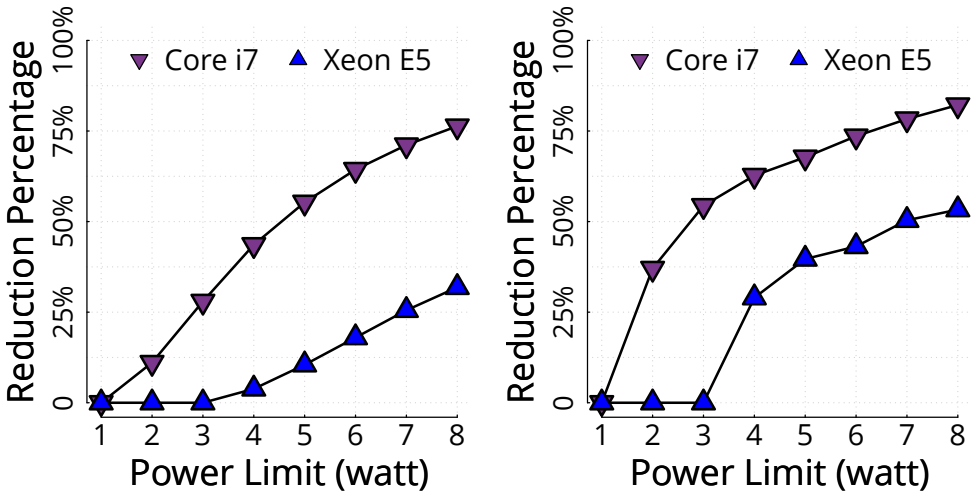
(b) Energy consumption

Figure 4.16.: Reduction percentage of execution time and energy consumption for sensitivity study on simulation scaling coefficients.

Apart from the coefficients relative to power capping and scaling, I also investigate the sensitivity of the reliability-aware model to parameters of cooling infrastructures and time to take a checkpoint.

According to [18], the sensitivity of temperature to power capping level is dependent of the cooling infrastructure. When equipped with a powerful cooling infrastructure, temperature increases slowly as power capping level increases. In this section, I evaluate the reliability-aware model with server processors which are equipped with powerful server cooling infrastructures. In order to study the sensitivity of the reliability-aware model to parameters of cooling infrastructures, I compare with the parameters obtained from a desktop platform. The Core i7-2600 platform is equipped with desktop cooling infrastructures, and its temperature is more sensitive (higher temperature gradient) to changes in power

consumption ($I = 0.75$, $J = 29.1$). The comparison results are shown in Fig. 4.17. I find that, as cooling infrastructures become less powerful, the reduction percentage of applying the reliability-aware model over the base model increases significantly, with respect to both execution time and energy consumption. Compared with the base model, applying the reliability-aware model reduces the execution time and energy consumption by up to 32.2% and 40.8% respectively, when running on platforms with powerful cooling infrastructures. In contrast, when running on platforms with insufficient cooling capacity, the reduction percentages boost to 74.1% and 78.0% for time and energy respectively. When temperature gradient is high, the reliability-aware model exploits the opportunity of reducing power capping level in order to increase the overall MTBF. Although increased scale reduces MTBF, the reliability-aware model explores the trade-off between power capping level and scale to maximize MTBF.

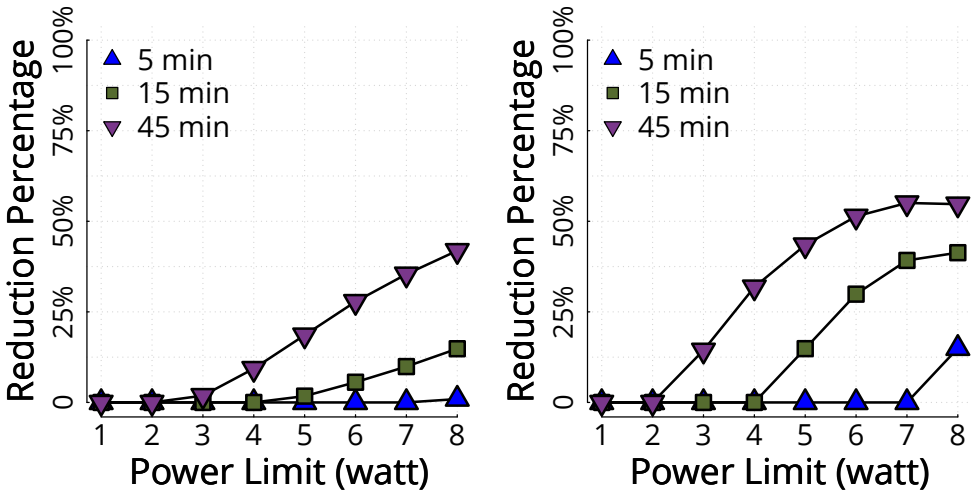


(a) Execution time

(b) Energy consumption

Figure 4.17.: Reduction percentage of execution time and energy consumption for sensitivity study on cooling platform parameters.

Finally, I study the sensitivity of the reliability-aware model to the time to take a checkpoint. In this section, I assume that time to checkpoint is 30 min.



(a) Execution time

(b) Energy consumption

Figure 4.18.: Reduction percentage of execution time and energy consumption for sensitivity study on time to take a checkpoint.

I study the sensitivity of the reliability model to this parameter using cases where it takes 5 min, 15 min, and 45 min to store the checkpointing data. As shown in Fig. 4.18, reduction percentage of applying the reliability model to the base model increases as time to checkpoint increases. When time to checkpoint increases, its weight in the total simulation time also grows. The reliability-aware model trades computation performance in to minimize the checkpointing overhead through reducing the scale and power capping level. Exploring such trade-off minimizes the total simulation time and energy especially when the time to checkpoint gets longer. Either large checkpoint data size of applications or small I/O bandwidth or storage systems can result in long time to checkpoint.

Finding 19 *Platforms with insufficient cooling capacity, applications with large checkpoint data size, and storage systems with limited I/O bandwidth benefit more from applying the reliability-aware model.*

CHAPTER 5
PROACTIVE BURST BUFFER DRAINING

In this chapter, I introduce the proactive burst buffer draining scheme in the following aspects. In section 5.1, I discuss the background and motivation. In section 5.2, I introduce the design of the proactive burst buffer draining scheme. The I/O provisioning model is introduced in section 5.3. I present the validation and evaluation results in section 5.4. In section 5.5, I discuss the proactive burst buffer draining scheme in various aspects.

5.1 Background and Motivation

In this section, I first describe the infrastructure of HPC storage systems. Then, I discuss the HPC I/O characteristics and the motivation of this work.

I take the OLCF (Oak Ridge Leadership Computing Facility) Spider storage system [85] as an example to describe the storage infrastructure. As illustrated in Fig. 5.1, the computing resources and permanent storage system are connected through a scalable I/O network consisting of InfiniBand switches. There are about 26,000 nodes mounted on the Spider storage system, where supercomputer Titan takes up more than two thirds of the nodes [86]. The center-wide storage system is shared by multiple computing resources, including supercomputer Titan, analysis cluster Rhea, visualization cluster Everest, etc. To meet the increasing data-sharing requirements, it is practical for HPC facilities to employ a center-wide storage system. The storage system at OLCF experienced the transition from a machine-exclusive model to a center-wide, data-centric model in 2005 [87]. On the storage side, the Lustre-based parallel filesystem runs on 288 Object Storage Servers (OSS). A total of 2,016 Object Storage Targets (OST) are

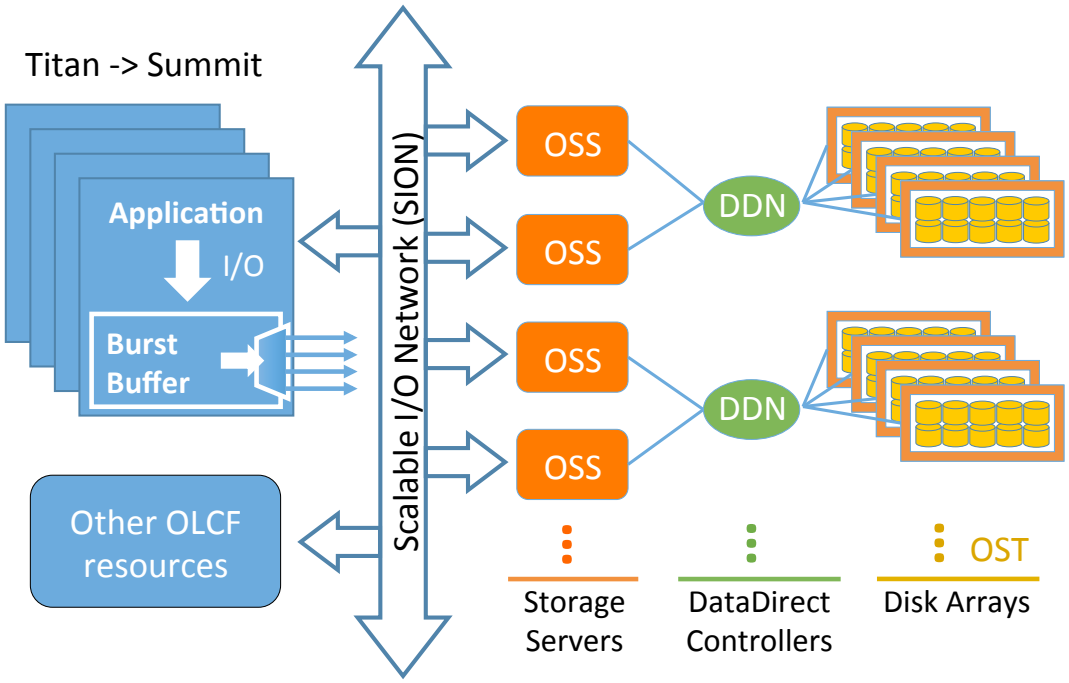


Figure 5.1.: Storage infrastructure at Oak Ridge Leadership Computing Facility [29].

connected to DataDirect Networks (DDN) SFA12K-40 RAID controllers. Each OST consists of 10 hard drives as a RAID6. An I/O monitoring tool regularly polls the DDN controllers and stores the I/O statistics (e.g., throughput, IOPS, etc.) into a database [88]. On the next-generation supercomputer Summit, node-local non-volatile memory is introduced as burst buffers.

Fig. 5.2 shows a 24-hour write trace of the Spider storage system. I observe that, for most of the times, aggregate write throughputs are below 10 GB/s. However, for the other times, aggregate write throughputs are above 10 GB/s and even larger than 20GB/s. As demonstrated by the I/O log, mostly, the I/O load is much lower than what the storage system can provide. However, the storage system is provisioned in order to provide sufficient bandwidth to the few I/O peaks. If a storage system is provisioned to serve I/O peaks, it is underutilized for the time between I/O peaks. Although modern systems exploit

various buffers to serve the I/O peaks, such peaks still exist on the permanent storage system without judiciously draining the buffers [9].

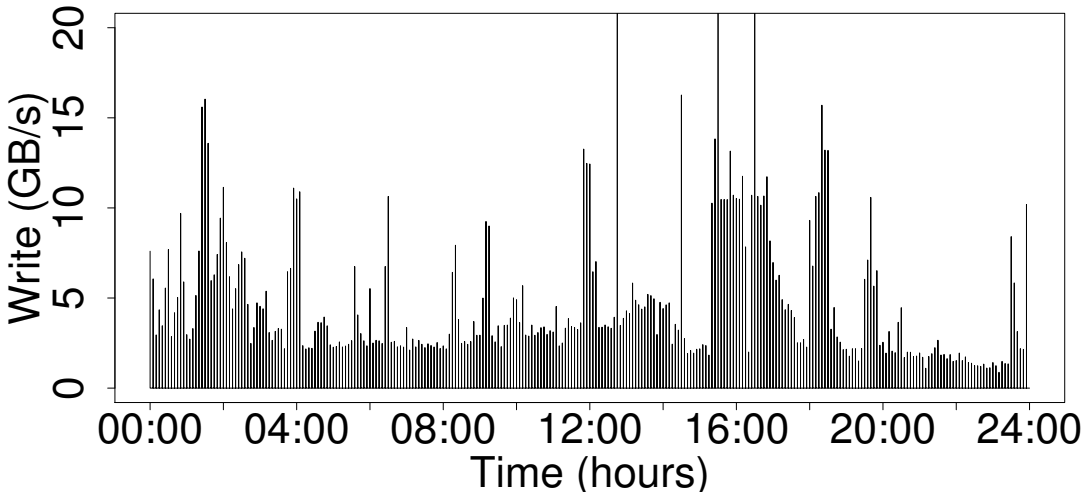
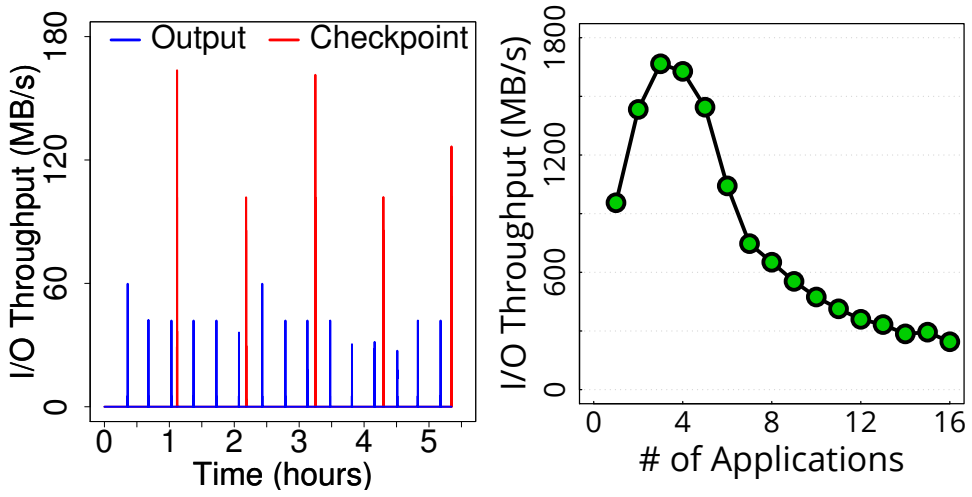


Figure 5.2.: One-day I/O (write) log for Spider storage system hosted at Oak Ridge National Laboratory.

HPC applications are known for periodic bursty I/O behaviors [8]. Taking the Community Earth System Model (CESM) as an example, climate data are stored to the permanent storage system per month simulation. Checkpoints are taken after every 3-month simulation. As illustrated in Fig. 5.3a, both application outputs and checkpoints are periodic and bursty. At machine level, various applications run simultaneously, which makes I/O peaks from different applications overlap with each other and create even higher peaks, as shown in Fig. 5.2.

Approaching exascale computing, the storage I/O is expected to have an O(100) increase comparing with petascale systems [20] [21] [22] [23]. I continue using the OLCF resources as an example. In the current petascale system, the Spider storage system consumes 400 KW power [29], and the peak power consumption of supercomputer Titan is 9 MW [17]. In an exascale system with a power limit of 20 MW [4], it would consume a considerable amount of the



(a) CESM I/O trace

(b) I/O contention

Figure 5.3.: Fig. 5.3a shows a sample I/O (write) trace from a 15-month CESM simulation on supercomputer Titan with component sets F_2000 and resolution f19_f19. This small-scale sample run is used to illustrate the I/O periodicity of scientific applications. Fig. 5.3b shows the aggregate I/O throughput for various numbers of applications on 4 OSTs.

power budget to provision for the O(100) increase in storage I/O. In order to expedite the storage I/O performance under power constraints, burst buffers are introduced to absorb the bursty bulk data and relax the I/O provisioning requirement of the underlying permanent storage systems. However, without judiciously draining the distributed burst buffers, I/O bursts still exist in the underlying storage systems [9]. Since the aggregate bandwidth of storage system is lower than the burst buffers, I/O bursts are reduced passively at the storage system. As a result, the I/O contention is exacerbated due to under-provisioning at the original I/O peaks from the applications. The read and write performance of the permanent storage systems suffer from the severe I/O contention at the I/O peaks.

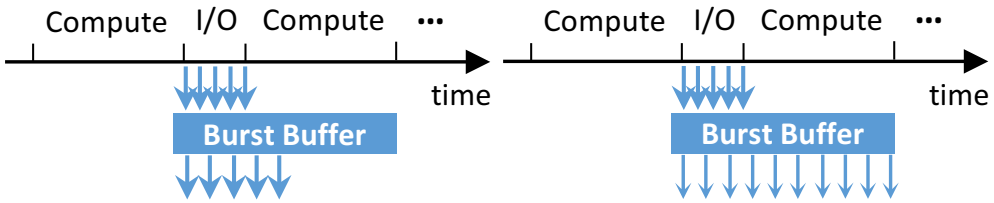
I run a real-system emulation to illustrate the I/O contention issues on under-provisioned storage systems. When running multiple applications on shared storage resources, under-provisioning can lead to severe resource contention. When the number of applications sharing the same storage resources is relatively small, running more applications can achieve higher aggregate I/O throughput. As illustrated in Fig. 5.3b, when the number of applications increases from 1 to 3, the aggregate I/O throughput increases from 950 MB/s to 1700 MB/s. However, when the number of applications running on shared storage resources keeps growing, the aggregate I/O throughput drops significantly. As shown in Fig. 5.3b, when the number of applications increases from 3 to 16, the aggregate I/O throughput decreases from 1700 MB/s to 250 MB/s. Running more applications than the serving capability of the storage system (i.e., the storage system is under-provisioned) causes severe resource contention. As a result, the performance of the storage system is degraded severely (85% degradation as shown in Fig. 5.3b).

In this work, I aim to minimize the storage I/O provisioning requirement and resolve the problems associated with I/O peaks utilizing the periodicity attribute of HPC I/O. As discussed in Fig. 5.3a, HPC applications dump data periodically to burst buffers. An intuitive strategy is to reactively flush the data as they arrive at burst buffers, known as **reactive draining** [70]. Instead of employing the reactive draining strategy, I propose to drain the data proactively to the permanent storage system, in order to flatten I/O peaks in the storage system.

5.2 Proactive Burst Buffer Draining Scheme

The proactive burst buffer draining scheme exploits the I/O periodicity. Since application I/O performance mainly relies on the burst buffer instead of the permanent storage system, the proactive burst buffer draining scheme can scatter

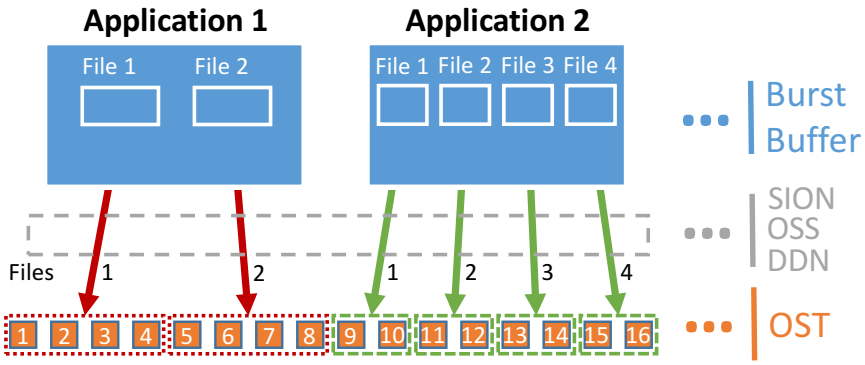
the I/O draining process within the I/O interval while not impacting the application I/O performance. As illustrated in Fig. 5.4, applications dump data periodically and burst buffers perform draining in an asynchronously manner. Using reactive draining, as shown in Fig. 5.4a, burst buffer flushes data quickly, which passes the I/O bursts on to the underlying storage system. By comparison, using proactive draining as illustrated in Fig. 5.4b, data are divided into small blocks and draining requests are dispersed evenly across the entire computation and I/O period. Consequently, I/O bursts stop at the burst buffer layer.



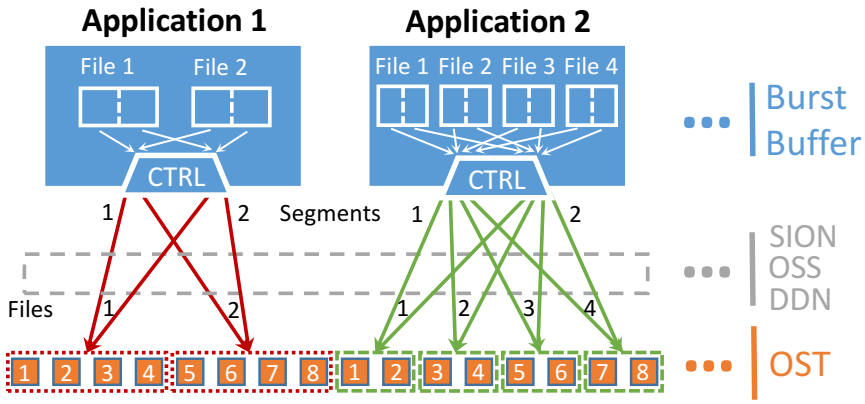
(a) Reactive burst buffer draining (b) Proactive burst buffer draining

Figure 5.4.: An illustration of reactive and proactive burst buffer draining. Arrows represent application I/O and burst buffer draining requests.

In a HPC storage system, OSTs are typically assigned to applications in a round-robin manner. As illustrated in Fig. 5.5a, OST 1 to 8 are assigned to application one and OST 9 to 16 are assigned to application two. Each file of application one is striped across 4 OSTs and each file of application two is striped across 2 OSTs. In reactive burst buffer draining, burst buffers flush data reactively to the underlying permanent storage system when data become available in burst buffers. After burst buffers get the assigned OSTs, the OSTs are appended to the waiting queue. Using proactive burst buffer draining, in this example, each file is divided into two segments, as illustrated in Fig. 5.5b. The burst buffer draining controller drains the first segment of each file, and then drains the second segment. Burst buffers issue draining requests to the assigned OSTs for an extended time period. The draining segments arrive at



(a) Reactive burst buffer draining



(b) Proactive burst buffer draining

Figure 5.5.: An comparison between reactive and proactive burst buffer draining. Arrows represent burst buffer draining paths of different files and segments. The burst buffers drain 2 files of application one with a stripe count of 4, and drain 4 files of application two with a stripe count of 2. Using proactive draining, each file is divided into 2 segments which are drained sequentially.

the same group of OSTs periodically. As shown in Fig. 5.5b, the draining segments of application two are interleaved with application one on the same OSTs. In proactive draining, segment is the basic scheduling and draining unit. I/O requests within draining segments of different applications do not overlap with each other. Otherwise, sequential I/O requests from multiple applications

interleave with each other and turn into random I/O requests, causing request contention on OSSs and resource contention on OSTs.

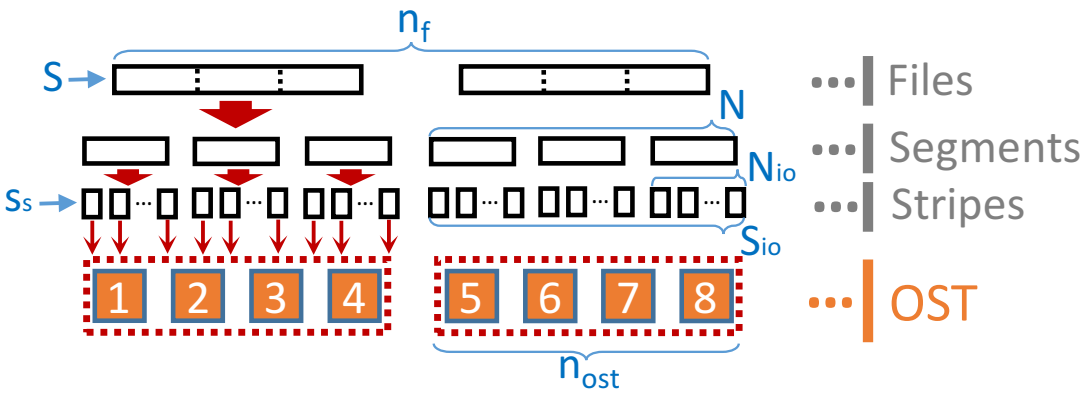


Figure 5.6.: An illustration of the data partition for proactive burst buffer draining. Layers SION, OSS, and DDN are not plotted for simplicity.

I use application one in Fig. 5.5b as an example to illustrate the data partition of the proactive burst buffer draining. As shown in Fig. 5.6, burst buffers drain n_f files (application output or checkpoint) to the permanent storage system. Each file is divided into N segments and each segment is divided into N_{io} stripes. The entire file is divided into S_{io} stripes, and is striped across n_{ost} OSTs. The file size is S and the stripe size is s_s. The relationship between these parameters is defined in Eq. 5.1.

$$N = \frac{S_{io}}{N_{io}} = \frac{S}{N_{io} \times s_s} \tag{5.1}$$

For node-local burst buffers, each compute node has a burst buffer draining controller. In each node, the draining controller calculates the application I/O interval based on the time difference between the start of application and the start of data dump. From the perspective of a specific group of OSTs, there is a set of applications sharing them. In order to coordinate the draining process of different applications and avoid contention across applications on the shared OSTs, the proactive burst buffer draining controllers need to calculate the maximum time

span for draining segments of different applications. Then, the draining controller splits the data into appropriate draining segments accordingly, and drains the segments periodically. Draining segments from different applications arrive at the shared OSTs periodically, interleaving with each other. The OSS keeps a map for each group of OSTs to record the application set. Every time a new application is added into the set or an existing one is removed from the set, the operation is registered in the related map. The proactive burst buffer draining controller regularly polls the map and updates the maximum time span for the draining segments.

5.3 I/O Provisioning Model

In section 5.2, I have discussed the design of the proactive burst buffer draining scheme. Based on the scheme, I propose the I/O provisioning model to calculate the minimized I/O provisioning requirement for permanent storage systems. Symbols used in the model are listed in Table 5.1.

When burst buffers perform draining reactively, the draining I/O throughput is similar to the application I/O throughput, if the storage system is provisioned to an adequate degree. I/O throughput is defined as the ratio between application data size and I/O time. If the storage system is provisioned to allow r fraction of application computation time to perform I/O operations, the reactive draining throughput is defined as Eq. 5.2.

$$P_{(r)} = \frac{\sum_{i=1}^{n_f} S_i}{r \times T} \tag{5.2}$$

In proactive burst buffer draining, data are divided into draining segments which are dispersed evenly over the I/O interval (T). The burst buffer draining throughput is controlled through adjusting the number of I/O requests issued each time (i.e. the size of the draining segments). Without considering the

Table 5.1.: Symbols and definitions

Symbols	Definitions
T	I/O interval of checkpoint or analysis dump
$P_{(r)}$	I/O throughput of reactive burst buffer draining
r	ratio between I/O time and I/O interval
B	proactive draining throughput without interference
$\widetilde{B}$	proactive draining throughput with interference
S	application data size per file
S_{io}	number of application data stripes per file
s_s	data stripe size
n_{ost}	number of OSTs to stripe over
N	number of draining segments per file
N_{io}	number of data stripes per draining segment
t	time of a draining segment per file
$\widetilde{t}$	hard drive positioning time
n_f	number of application files
S_i	application data size of the ith file
N_i	number of draining segments of the ith file
t_i	time of a draining segment of the ith file
n_a	number of applications sharing common OSTs
$T_{(x)}$	the xth application I/O interval
$S_{(x)}$	application data size of the xth application
$N_{(x)}$	number of draining segments of the xth application
$t_{(x)}$	time of a draining segment of the xth application

interference of other applications, the proactive draining throughput is defined in Eq. 5.3.

$$B = \sum_{i=1}^{n_f} \frac{S_i}{N_i \times t_i} \tag{5.3}$$

In a throughput-oriented storage system, the aggregate I/O throughput normally increases as the number of concurrent I/O requests increases. In order to quantify this relationship, I perform a regression analysis based on the simulation results. I employ the Disksim [89] simulator with the validated disk parameter set "Seagate Cheetah 15k.5". The simulation shares the real-system emulation settings in section 5.4, except that each disk array is configured to have 9 disks as a RAID5, which has the same number of non-parity disks (8 disks per OST) as the emulation settings.

In the I/O provisioning model, the number of data stripes per draining segment (N_{io}) has upper and lower bounds. The lower bound is calculated based on device block size and data stripe size. The device block size is 512 bytes in the simulation settings. The product of N_{io} and data stripe size should be larger than the product of device block size and the number of non-parity disks. The lower bound of N_{io} is 12 in the simulation settings. Concerning the upper bound, when N_{io} grows beyond the upper bound, aggregate I/O throughput keeps steady and the OSTs become saturated.

Table 5.2.: Coefficients for regression analysis

Coefficients	MAX HDD 1 MB	HDD 1 MB	HDD 2 MB	SSD
a	0.585	0.693	0.648	0.552
b	1.044	0.163	0.688	3.653

As shown in Fig. 5.7, the aggregate I/O throughput reaches a plateau, as the number of data stripes gets larger. Concerning HDD as storage resources, beyond an upper bound of N_{io}, the aggregate I/O throughput remains steady at 4.0 GB/s. When data stripe size equals to 1 MB and 2 MB, the upper bounds of N_{io} are 1536 and 768 respectively. When it comes to SSD, the upper bound

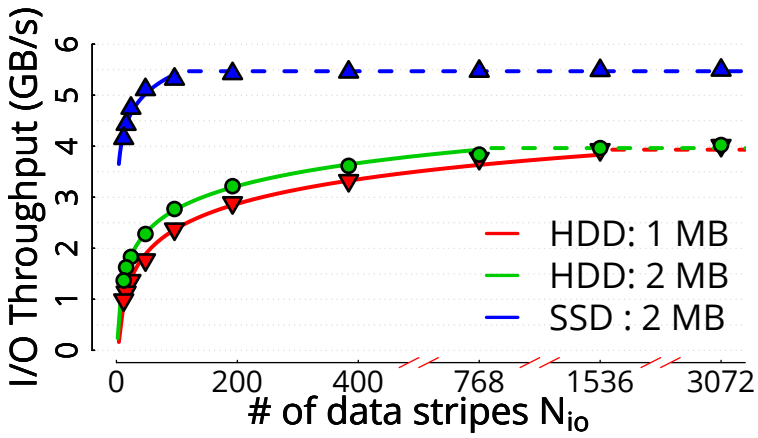


Figure 5.7.: Aggregate draining I/O throughput under various data stripe counts on 4 OSTs.

reduces significantly to 100 and the aggregate I/O throughput remains steady around 5.5 GB/s beyond the upper bound of N_{io}. In the proactive burst buffer draining scheme, I propose to drain the application data in a number of segments with smaller N_{io}, in order to reduce I/O peaks. Therefore, I only consider the range between the upper and lower bounds of N_{io}. I find that the I/O throughput trend can be captured by natural logarithm functions, as illustrated by the solid lines in Fig. 5.7. The R-squared values of regression functions are above 0.997 for HDD and are above 0.981 for SSD, which indicate a statistically sound fit.

The relationship between the number of data stripes (N_{io}) and aggregate I/O throughput per file (B') can be expressed as Eq. 5.4.

$$B' = \frac{S}{N \times t} = a \times \ln(N_{io}/n_{ost}) + b \tag{5.4}$$

Coefficients a and b vary across different storage settings, which are listed in Table 5.2. Combining Eq. 5.1 and Eq. 5.4, time of a draining segment per file (t) can be expressed as Eq. 5.5.

$$t = \frac{N_{io} \times s_s}{a \times \ln(N_{io}/n_{ost}) + b} \tag{5.5}$$

According to Eq. 5.5, the time of a draining segment of the ith file (t_i) is independent of the file size (S_i), but is dependent on the number of data stripes per draining segment (N_{io}).

In the permanent storage system, applications request and release OSTs in a round-robin manner. After applying the proactive burst buffer draining scheme, draining segments arrive periodically for an extended period after the OSTs are released, making the OSTs being shared by multiple applications. Sequential draining segments from multiple applications interleave and interfere with each other, which incurs notably large disk positioning time in front of each draining segment. Disk positioning time ($\widetilde{t}$) consists of seek time and rotational latency. Although a large draining segment can hide the disk positioning time, it raises I/O throughput as illustrated in Fig. 5.7. Therefore, it is essential to consider the impact of disk positioning time in the context of multi-applications, especially with small draining segments. When burst buffers perform proactive draining as illustrated in Fig. 5.5b, considering the interference across applications, the I/O throughput is defined in Eq. 5.6.

$$\widetilde{B} = \sum_{i=1}^{n_f} \frac{S_i}{N_i \times (t_i + \widetilde{t})} \tag{5.6}$$

Between each of the N draining segments, the storage resources are utilized by the other applications. Since draining segments only append to each other instead of breaking into each other, there exists extra hard drive positioning time $\widetilde{t}$ at the beginning of each segment. Hard drive positioning time is dominated by the seek time as rotational latency remains constantly small. Seek time can be expressed as a function of seek distance [90] [91]. Each of the N segments has a distinct hard drive positioning time depending on the seek distance. In order to simplify the I/O provisioning model, I propose an aggressive strategy and a conservative strategy for calculating the draining I/O throughput $\widetilde{B}$. When the storage system is shared with latency-sensitive applications (e.g., data analysis and visualization), the conservative strategy should be used in order to preserve

the available I/O bandwidth for the other applications. Otherwise, the aggressive strategy can be used in order to reduce I/O provisioning requirement.

- Using the aggressive strategy, $\widetilde{t}$ is the average positioning time for all N segments. In this case, parameters a and b in Eq. 5.4 are derived from the average I/O throughput.
- Under the conservative strategy, $\widetilde{t}$ is the minimum positioning time among all N segments. Parameters a and b are measured based on the maximum I/O throughput.

The draining I/O throughput is reduced passively by random I/O requests, due to extra disk positioning time between draining segments. Therefore, to achieve draining I/O throughput $\widetilde{B}$, the permanent storage system is actually provisioned at bandwidth B. According to Eq. 5.4, when the size of draining segments (N_{io}) decreases, draining I/O throughput decreases. In contrast, the gap between $\widetilde{B}$ and B grows larger as N_{io} decreases.

On shared storage resources, the minimum I/O throughput is constrained by the minimum draining throughput of each client. The minimum draining throughput of a client is defined as the ratio between data size ($S_{(x)}$) and I/O interval ($T_{(x)}$). When multiple clients share the storage resources, the draining throughput of client x should be larger than the aggregate minimum throughput of all clients, in order to let clients finish draining within their I/O intervals, as defined in Formula 5.7.

$$\frac{S_{(x)}}{N_{(x)} \times (t_{(x)} + \widetilde{t})} \geq \sum_{x=1}^{n_a} \frac{S_{(x)}}{T_{(x)}} \tag{5.7}$$

Using smaller draining segments (N_{io}) leads to lower draining throughput and longer draining time. On shared storage resources, if one client drains data at lower throughput and takes longer time, the other clients will have to perform draining at higher throughputs due to shortened time slices. Therefore, in order to minimize the overall I/O throughput, all clients sharing the same group of

OSTs should adopt the same size of draining segments. According to Formula 5.7, the minimum segment size (N_{io}) should satisfy the following formula.

$$\frac{1}{a \times \ln(N_{io}/n_{ost}) + b} + \frac{\widetilde{t}}{N_{io} \times s_s} \leq \frac{1}{\sum_{x=1}^{n_a} \frac{S_{(x)}}{T_{(x)}}} \tag{5.8}$$

Every time a new application is assigned to a group of OSTs or an existing application finishes, the maximum time span of each client changes accordingly. The unified number of data stripes per draining segment (N_{io}) is updated based on formula 5.8, and each client adopts the updated N_{io} to drain data.

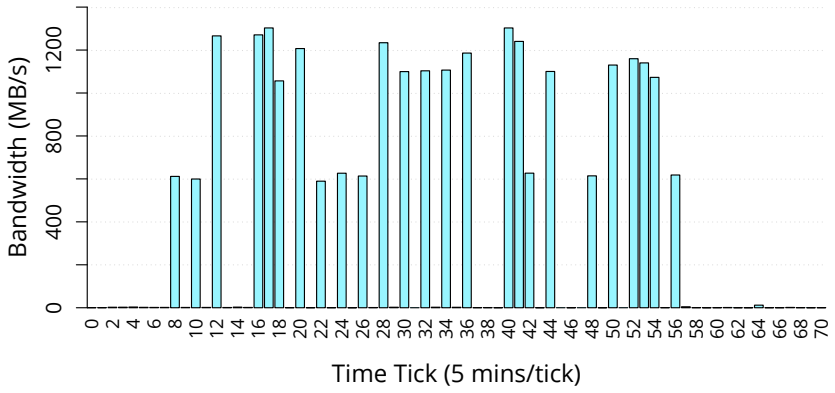
5.4 Evaluation

I perform the evaluation based on emulation, simulation, and model-driven methods. In subsection 5.4.1, I evaluate the proactive burst buffer draining scheme and compare against the reactive burst buffer draining scheme. In subsection 5.4.2, first, I validate and evaluate the I/O provisioning model. Then, I evaluate the model with leadership applications.

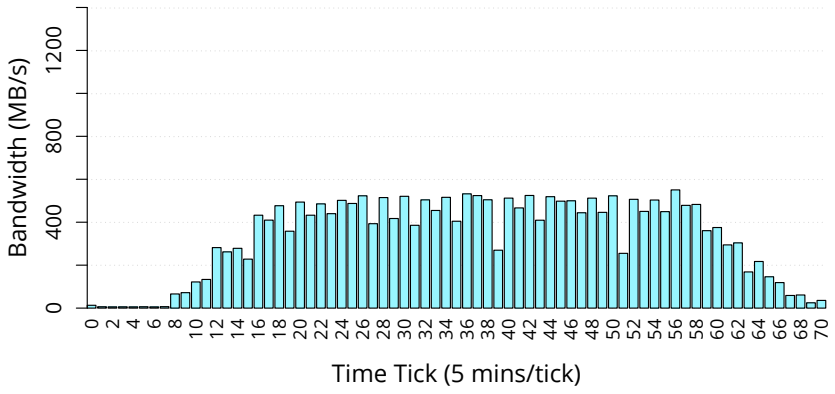
5.4.1 Burst Buffer Draining Schemes

I evaluate and compare the proactive draining scheme and reactive draining scheme based on real-system emulation and simulation.

First of all, I employ a real-system emulation to demonstrate how the proactive burst buffer draining scheme proactively flattens the aggregate I/O peaks for all applications. The emulation setting is described in section 5.2. Throughput data are collected from the DDN RAID controllers of the Spider storage system. Emulation results are shown in Fig. 5.8 and Fig. 5.9. Comparing Fig. 5.8a and Fig. 5.9a, I observe that without applying the proactive burst buffer draining scheme, I/O peaks drop from 1300 MB/s to 800 MB/s in a passive manner when storage provisioning is halved from 4 OSTs to 2 OSTs. This indicates that, using



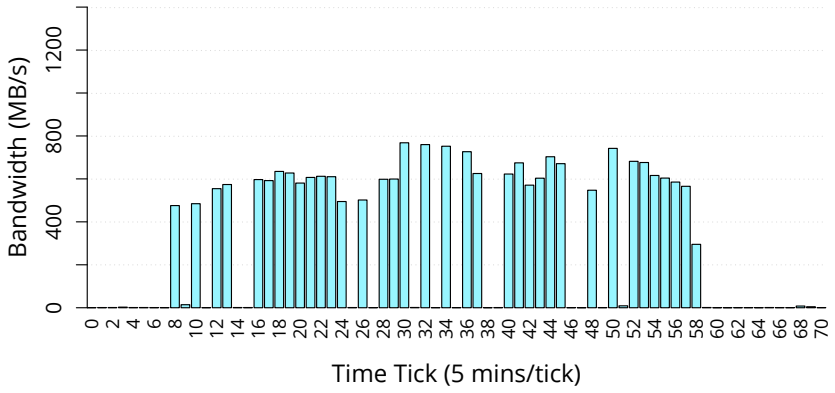
(a) Reactive buffer draining



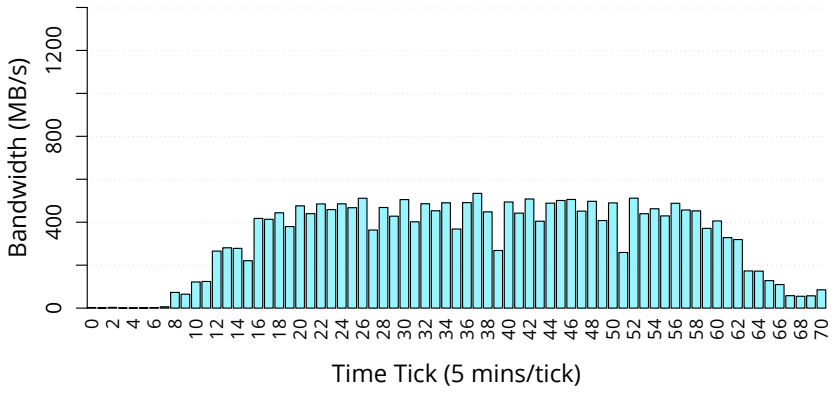
(b) Proactive buffer draining

Figure 5.8.: Aggregate bandwidth for all 12 applications on 4 OSTs

2 OSTs, applications have higher I/O provisioning requirement and the storage system is overloaded near the original I/O peaks. In contrast, comparing Fig. 5.8a and Fig. 5.8b, when applying the proactive burst buffer draining scheme, I/O peaks are flattened at 500 MB/s in an active manner. When storage provisioning is halved from 4 OSTs to 2 OSTs, as shown in Fig. 5.9b, I/O peaks remain steady around 500 MB/s and the storage system is underloaded, whose peak bandwidth can be as high as 800 MB/s according to Fig. 5.9a. From the comparison, I observe that decreasing the I/O provisioning reduces the I/O



(a) Reactive buffer draining



(b) Proactive buffer draining

Figure 5.9.: Aggregate bandwidth for all 12 applications on 2 OSTs

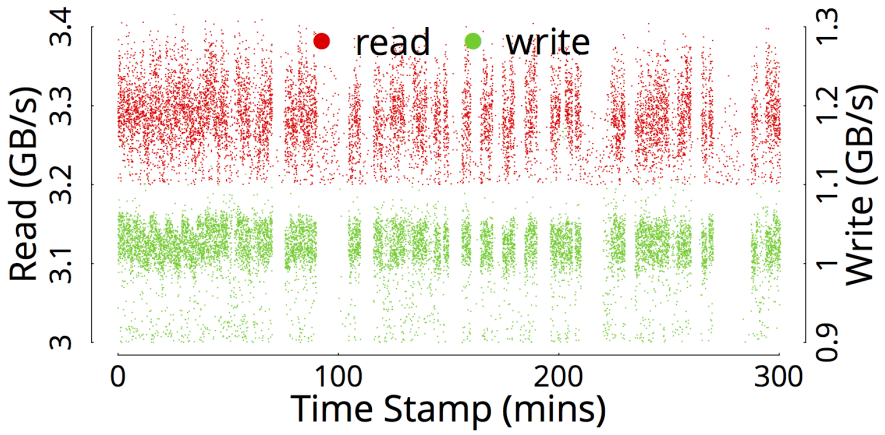
bursts passively, when burst buffers drain data reactively. This usually causes severe I/O contention issues among draining processes and other clients sharing the storage system. However, applying the proactive burst buffer draining scheme actively flattens the I/O bursts, which in return preserves the available bandwidth for other clients of the storage system.

A center-wide storage system is shared by various computing resources. For example, the OLCF Spider storage system is shared by production cluster Titan, preparation cluster Eos, analysis cluster Rhea, visualization cluster Everest,

various small-scale clusters, an archival system HPSS, and users on login nodes, as illustrated in Fig. 5.1. The introduction of burst buffers expedites the application write performance. However, draining burst buffers reactively causes I/O contention issues on the underlying storage system, which in return slows down the draining process and accumulates data in burst buffers. More importantly, the I/O contention also impacts the storage system read performance. Computing tasks, such as application restart, analysis tasks, visualization tasks, and user accessing files, are sensitive to read performance degradation.

In order to illustrate such impact, I run an IOR instance with repeating read and write requests to measure the system I/O performance. I run the IOR instance together with the background burst buffer draining requests (i.e., the 12 applications in Fig. 5.8 and Fig. 5.9), and compare the system I/O performance under different burst buffer draining schemes. The read and write throughputs of the IOR instance are shown in Fig. 5.10 and Fig. 5.11. In Fig. 5.10a, I observe that there are many gaps along the axis of time, where major throughput drops happen. By contrast, read and write throughputs stabilize along the axis of time when applying the proactive burst buffer draining scheme, as shown in Fig. 5.10b. When I/O provisioning is halved from 4 OSTs to 2 OSTs, the gaps grow much bigger (especially for write requests) when draining burst buffers reactively, as shown in Fig. 5.10a and Fig. 5.11a. In contrast, when applying the proactive draining scheme with 2 OSTs, the throughputs of most I/O requests still remain stabilized along the axis of time, as shown in Fig. 5.11b. According to the comparison, I observe that applying the proactive burst buffer draining scheme preserves the I/O performance for the other clients sharing the storage system, especially when I/O provisioning diminishes.

In order to further analyze the differences between the two draining schemes at block level, I resort to simulation due to the lack of support to collect block level I/O trace on the Spider storage system. I collect application-level I/O traces



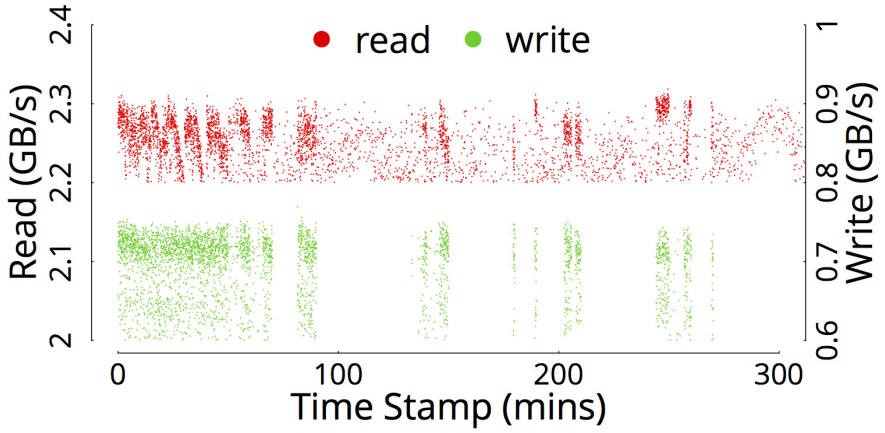
(a) Reactive buffer draining

(b) Proactive buffer draining

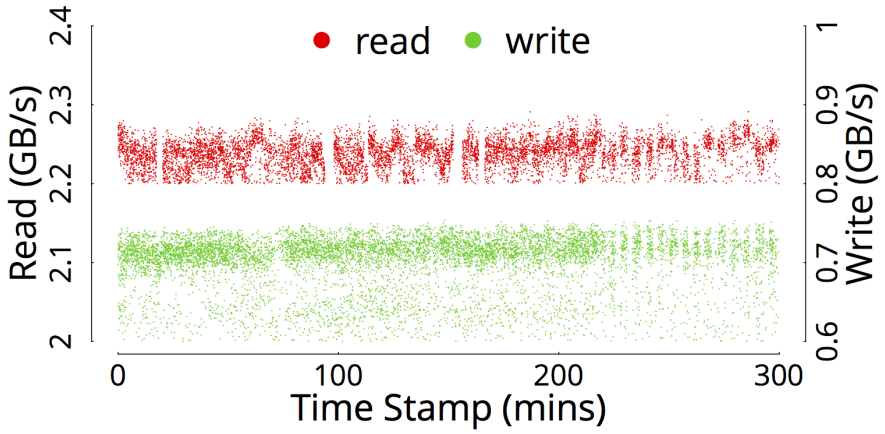
Figure 5.10.: Available read and write bandwidth of the permanent storage system (4 OSTs) with the 12 applications (in Fig. 5.8) running at background. Using 4 OSTs, read throughput above 3.2 GB/s and write throughput above 0.9 GB/s are compared.

from the real-system emulations and replay the traces on the Disksim simulator. I utilize the same simulation settings as in section 5.3.

The cumulative distribution function of I/O response time is shown in Fig. 5.12. Concerning system response time, as shown in Fig. 5.12a, applying the proactive burst buffer draining scheme reduces the system response time signifi-



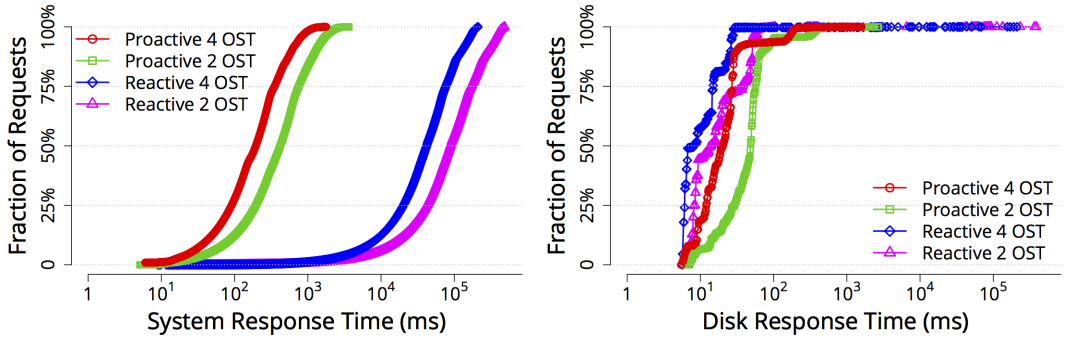
(a) Reactive buffer draining



(b) Proactive buffer draining

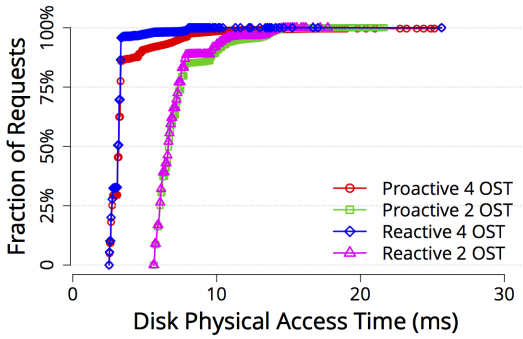
Figure 5.11.: Available read and write bandwidth of the permanent storage system (2 OSTs) with the 12 applications (in Fig. 5.9) running at background. Using 2 OSTs, read throughput above 2.2 GB/s and write throughput above 0.6 GB/s are compared.

cantly. The huge gap between proactive draining and reactive draining is mainly caused by the difference in I/O arrival time. Applying the proactive burst buffer draining scheme delays the arrival time of most of the requests, which leads to a relatively smaller response time. This indicates that applying the proactive burst buffer draining scheme relieves the burden of the system I/O queue and reduces



(a) System response time

(b) Disk response time



(c) Disk physical access time

Figure 5.12.: Cumulative Distribution Function (CDF) of system response time, disk response time, and disk physical access time. System response time is the difference between I/O completion time and I/O arrival time at the disk array level. Disk response time is the aforementioned difference at the disk level. Disk physical access time is the difference between disk response time and queuing time. X-axis is plotted in logarithm scale for better illustration.

the system queuing time. For latency-sensitive tasks, a shorter queuing time means quicker response. With respect to disk response time, as shown in Fig. 5.12b, the latency tail is between 1 and 10 seconds using proactive draining and is between 100 and 1000 seconds using reactive draining. Therefore, applying the proactive burst buffer draining scheme reduces the I/O latency tail by orders of magnitude. It should be noted that, the disk response time for many requests are

smaller using reactive draining than utilizing proactive draining. This is because the available system I/O bandwidth is much larger between I/O peaks using reactive draining, although the available bandwidth gets much smaller around I/O peaks. In contrast, the available system I/O bandwidth remain relatively steady utilizing proactive burst buffer draining scheme. As a result, compared with proactive draining, many of the I/O requests have a smaller disk response time while a few of them have a much larger disk response time in reactive draining. In order to investigate the cause of the variation in disk response time distribution for both draining schemes, I study the distribution of disk physical access time. As illustrated in Fig. 5.12c, for most of the I/O requests, the disk physical access time in proactive draining is similar as reactive draining. This indicates that the variation in disk response time is not caused by resource contention. Therefore, the variation is mainly determined by the disk queuing time, which indicates that request contention is the root cause. Comparing with reactive draining, proactive draining scheme reduces system response time significantly and relieves the burden of the system I/O queue. In addition, using proactive draining scheme reduces the variation in disk response time distribution, which preserves the available bandwidth to other clients sharing the storage system.

5.4.2 Model Validation and Model-driven Study

In this subsection, first, I validate the accuracy of the I/O provisioning model. Then, I perform sensitivity studies on the model. Finally, I evaluate the I/O provisioning model with leadership applications.

In the I/O provisioning model, I/O throughput B is derived from simulation results. As long as I/O throughput $\widetilde{B}$ is validated, the remaining derivative metrics are valid. I validate the I/O throughput $\widetilde{B}$ based on simulation results. The simulation keeps the real system emulation settings in section 5.2 with varying numbers of data stripes per sub-request (N_{io}). Model and simulation results are

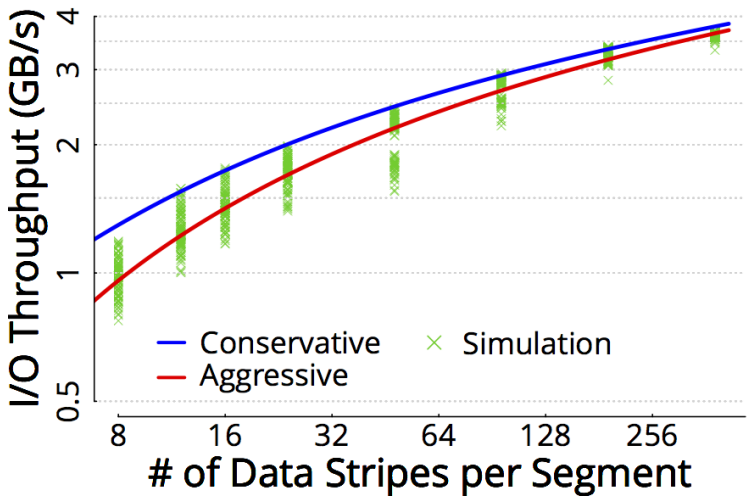
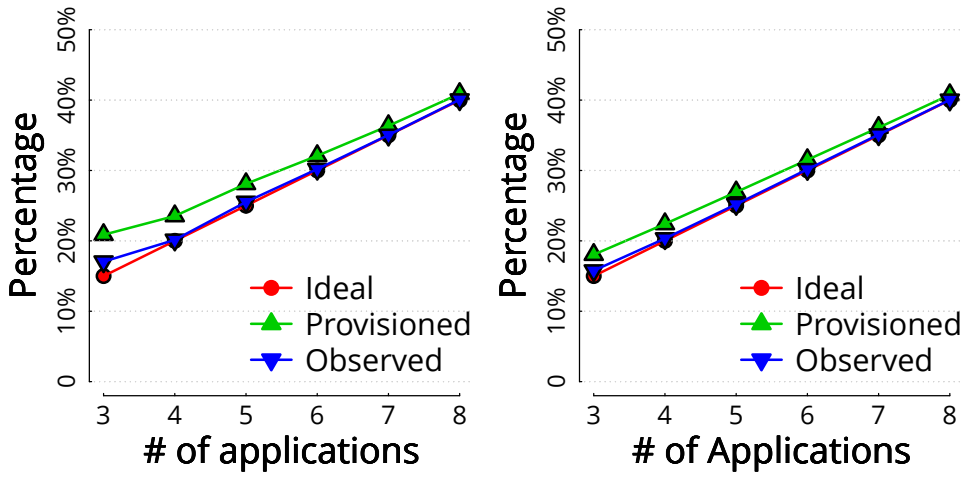


Figure 5.13.: Aggregate I/O throughput of conservative model, aggressive model, and simulation

shown in Fig. 5.13. Simulation results are denoted as points (crosses), conservative model results are plotted using a blue line, and aggressive model results are depicted as a red line. As illustrated in the figure, the I/O throughput for each draining segment under the same stripe count varies within the vertical range. The aggressive model can accurately capture the average I/O throughput for all the draining segments. Furthermore, the conservative model covers all the I/O throughput data under various stripe counts in a tight manner.

Then, I perform sensitivity studies on the number of applications (n_a) sharing common OSTs and application data size per file (S). Both application-specific parameters directly affect the decision on the draining segment size (N_{io}), as illustrated in Formula 5.8. In the remainder of this chapter, it is assumed that an application requires I/O overhead to be less than 5% of its computation time ($r = 5\%$).

In proactive burst buffer draining, draining segments from different applications interleave with each other on shared OSTs. I evaluate the sensitivity of the model to the number of applications per OST (n_a) for the range between 3



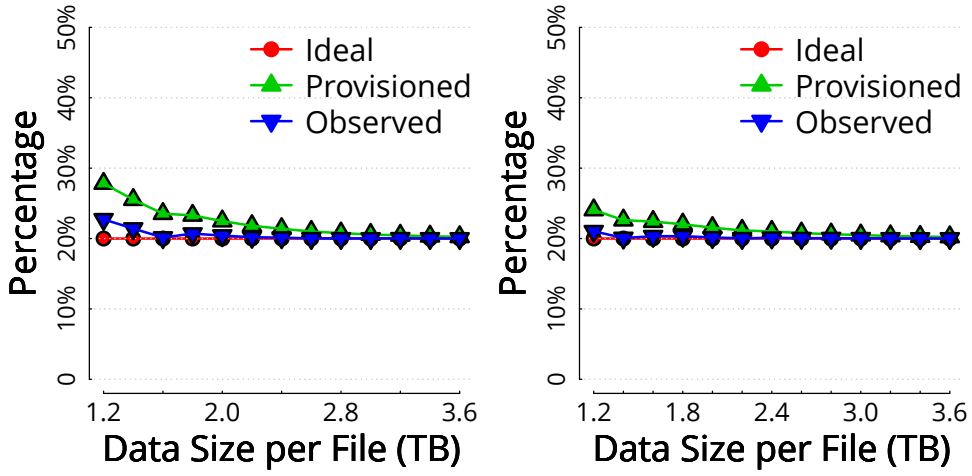
(a) Conservative strategy

(b) Aggressive strategy

Figure 5.14.: Sensitivity study on the number of applications (n_a) per OST. The file size (S) of each application is 1.6 TB striping across 4 OSTs. Y-axis represents the percentage of the reactive draining throughput ($P_{(5\%)}$). "Observed" refers to the actual throughput ($\widetilde{B}$) observed on the storage system, as illustrated in Eq. 5.6. "Provisioned" refers to the bandwidth (B) provisioned in order to achieve the observed throughput, as illustrated in Eq. 5.3. "Ideal" is defined as $P_{(1)}/P_{(5\%)}$.

and 8. As shown in Fig. 5.14a, using the conservative strategy, the provisioned bandwidth of proactive draining increases from 20.9% to 40.9% of reactive draining throughput, as the number of applications sharing common OSTs increases. Using the aggressive strategy, the provisioned bandwidth of proactive draining increases from 18.0% to 40.7%, which is slightly lower than the conservative strategy. Observed throughputs of both strategies are close to the ideal case. In addition, the provisioned bandwidth gets closer to the observed throughput and ideal case as n_a increases. Therefore, as system I/O load escalates, consolidating more applications into shared OSTs improves the bandwidth utilization, because the lower bound of the draining segment size gets larger according to to Formula

5.8. To be noted that, the size of draining segments still needs to be minimized to the lower bound rather than increasing the draining segment size.



(a) Conservative strategy

(b) Aggressive strategy

Figure 5.15.: Sensitivity study on the application data size per file (S) striping across 4 OSTs. There are 4 application per OST. Y-axis and legends have the same meaning as in Fig. 5.14.

Concerning the application data size per file (S), it is bounded by the product of time span and maximum I/O throughput. An application outputs n_f files and each file is striped across a group of OSTs. On a specific group of storage resources, the maximum time span that an application can take is defined in Formula 5.7. Based on the maximum I/O throughput, we can get the upper bound of the file size which is 3.6 TB in the sensitivity study. I evaluate the range between 1.2 TB and 3.6 TB for per application file size (S). As illustrated in Fig. 5.15, the ideal case remains the same with different file sizes, because both reactive draining throughput and proactive draining throughput change synchronously. As data size increases, the provisioned bandwidth gets much closer to the observed throughput and ideal case, especially for the conservative strategy. With a fixed data size, increasing the data size per file decreases the

number of files. Therefore, employing less files per application tends to improve the system bandwidth utilization, due to enlarged draining segments.

Table 5.3.: Checkpointing and analysis characteristics of leadership applications [1] [12] [18]. Checkpointing frequency (freq.) is once per hour for all applications.

Application Name	**Scientific Domain**	**Checkpoint Data Size**	**Analysis Data Size**	**Analysis Dump Freq.**
CHIMERA	Astrophysics	160 TB	160 TB	1 / hour
GTC	Fusion	20 TB	10 GB	1 / minute
S3D	Combustion	5 TB	5 TB	1 / 30 minutes
GYRO	Fusion	50 GB	10 GB	1 / minute
POP	Climate	26 GB	1.4 GB	1 / minute

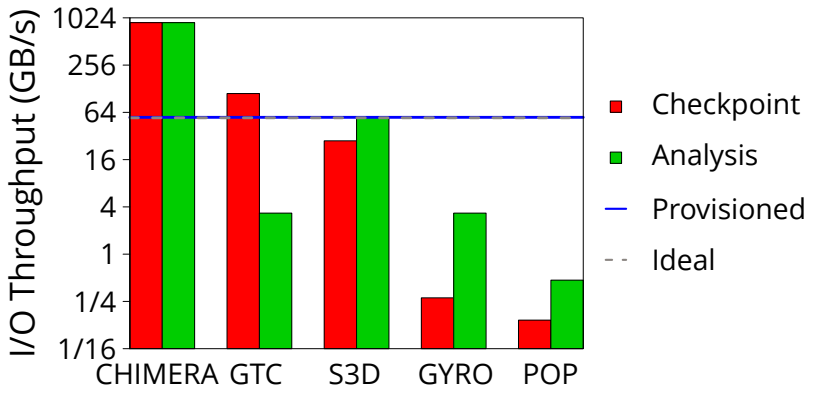


Figure 5.16.: I/O provisioning requirement of running multiple leadership applications on shared storage resources. Bars denote I/O provisioning requirement for each leadership application using reactive draining. Solid line represents the provisioned bandwidth using proactive burst buffer draining. Dotted line represents the minimum bandwidth in ideal case.

Finally, I evaluate the I/O provisioning model with leadership applications as listed in Table 5.3. These are the top 5 largest data-producing applications in the

National Center for Computational Sciences at Oak Ridge National Laboratory. I evaluate the case where they run simultaneously on shared storage resources. For simplicity, I assume that each application runs at the same scale and adopts the File-per-Process I/O pattern. The I/O provisioning requirement for both checkpoint and analysis for the leadership applications are shown in Fig. 5.16. Among the leadership applications, CHIMERA has the highest I/O provisioning requirement of 889 GB/s. Exploiting proactive burst buffer draining, ideally, the aggregate draining throughput can be reduced to 54.5 GB/s. According to the I/O provisioning model, the actual I/O provisioning requirement is reduced to 55.3 GB/s, which is very close to the ideal case as shown in Fig. 5.16. Therefore, compared with reactive draining, applying the proactive burst buffer draining scheme significantly reduces the I/O provisioning requirement.

5.5 Discussion

In this section, I discuss the concerns that may compromise the benefits of applying the proactive burst buffer draining scheme from various aspects.

Unknown or Dynamically Changing I/O Periodicity

As discussed in section 5.2, the draining period equals to the timespan of the previous computation segment. The proactive draining controller does not need to know the I/O periodicity of the application in advance. Even if the I/O periodicity changes dynamically, the draining period changes accordingly.

Random Write Behaviors

When scientific applications implement parallel I/O (File-per-Process or Single-shared-file), they may show random write behaviors on the burst buffers. After all the data are buffered, they are divided into draining segments and are drained to the permanent storage system sequentially.

The Impact of Burst Buffer Draining on Host CPUs

Although the proactive burst draining process overlaps with the computation, it is straightforward that I/O operations impose little burden on CPUs. As illustrated in [18], CPU power consumption is extremely low during checkpointing and CPU throttling has little impact on checkpointing performance. Moreover, the proactive draining controller drains smaller groups of data throughout the draining period, which also helps minimize the impact on host CPUs.

Storage Requirement on Burst Buffers

The proactive burst buffer draining holds data for an extended timespan. However, it does not impose any higher requirement to the storage capacity of burst buffers, because the draining controller drains all the data before the next data dump. In addition, I assume that node-local burst buffers are provisioned with sufficient capacity to hold the application data. Taking the supercomputer Summit as an example, the 3,400 nodes will be equipped with 800GB of NVRAM each node, which are mainly used as burst buffer or additional node memory [28]. As listed in Table 5.3, the maximum data size of leadership applications is 160 TB, which takes less than 6% of the total burst buffer size.

Support for Failure Restart

The proactive draining controller keeps the restart files in the burst buffers until the end of the draining period. Once failures happen, the application can restart from data in node-local storage, for example, using the SCR framework [6]. If the failure cannot be recovered using from node-local storage, it can restart from the permanent storage system. The application can either wait for the checkpoint data to be drained or restart from the latest available checkpoint on the storage system.

CHAPTER 6
CONCLUSIONS

In this dissertation, I proposed three techniques to improve the performance and energy efficiency of computing and storage resources in power-constrained HPC facilities. The proposed techniques are summarized in the following three sections respectively.

6.1 Power-capping Aware Checkpointing

In this work, I investigated the effects of power capping on the optimal checkpointing interval. I studied the effect of power capping on compute and checkpointing phases for a variety of scientific applications. I also demonstrated and quantified how power capping affects the system reliability due to change in temperature. I developed an power-aware OCI model and validation results show that the proposed model can accurately predict the OCI under power capping. The evaluation shows that applying my model to a set of large-scale applications can save up to 18% energy and execution time. Moreover, applying the power-capping aware model reduces the volume of data movement by up to 57% for these large-scale applications.

6.2 Reliability-aware Platform Configuration

In this work, I characterized HPC workflows in terms of performance, energy efficiency, processor heterogeneity, power capping, and error resilience. Then, I proposed a reliability-aware optimal platform configuration model to determine the optimal combination of power limit, processor power capping level, and

scale for simulation and analysis respectively. Validation results show that the proposed model is accurate in predicting the optimal platform configurations. Furthermore, I presented model-driven studies and made several insightful findings on how platform configurations impact the reliability, execution time, and energy consumption of HPC workflows under power constraints.

6.3 Proactive Burst Buffer Draining

In this work I observed that, without judiciously managing the burst buffer draining process, I/O peaks will be passed on to the underlying storage system, causing severe I/O contention. Therefore, I designed a proactive burst buffer draining scheme to regulate bursty I/O behaviors. The goal is to relax the I/O provisioning requirement and reduce the storage system power consumption. Evaluation results show that applying the proactive draining scheme flattens I/O bursts, relieves I/O contention, and preserves the available I/O bandwidth for the other clients sharing the storage system. Then, based on the proactive draining scheme, I developed a model to predict the minimal I/O provisioning requirement. I evaluated the model with leadership applications, and showed that the I/O provisioning requirement is reduced to a near minimal level. In conclusion, the proactive draining scheme fully exploits burst buffers to minimize the I/O provisioning requirement, reduce storage system power consumption, and preserve the I/O performance of underlying storage systems.

REFERENCES CITED

[1] Douglas Kothe and Ricky Kendall. Computational Science Requirements for Leadership Computing. Technical report, Oak Ridge National Laboratory, 2007.

[2] Wayne Joubert, Douglas Kothe, and Hai Ah Nam. Preparing for Exascale: ORNL Leadership Computing Facility Application Requirements and Strategy. Technical report, Oak Ridge National Laboratory, 2009.

[3] David Filani, Jackson He, Sam Gao, Murali Rajappa, Anil Kumar, Pinkesh Shah, and Ram Nagappan. Dynamic Data Center Power Management: Trends, Issues, and Solutions. *Intel Technology Journal*, 12(1):59–68, 2008.

[4] DOE ASCAC Subcommittee Members. Top ten exascale research challenges. Technical report, DOE ASCAC Subcommittee, 2014.

[5] J. Duell, P. Hargrove, and E Roman. The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart. Technical report, Berkeley Lab, 2002.

[6] Adam Moody, Greg Bronevetsky, Kathryn Mohror, and Bronis R. de Supinski. Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2010.

[7] Leonardo Bautista-Gomez, Seiji Tsuboi, Dimitri Komatitsch, Franck Cappello, Naoya Maruyama, and Satoshi Matsuoka. FTI: high performance fault tolerance interface for hybrid systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 32:1–32:12, 2011.

[8] Yang Liu, Raghul Gunasekaran, Xiaosong Ma, and Sudharshan S. Vazhkudai. Automatic identification of application i/o signatures from noisy server-side traces. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies*, pages 213–228, 2014.

[9] Ning Liu, Jason Cope, Philip Carns, Christopher Carothers, Robert Ross, Gary Grider, Adam Crume, and Carlos Maltzahn. On the Role of Burst Buffers in Leadership-Class Storage Systems. In *28th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–11, 2012.

[10] J. Bent, G. Grider, B. Kettering, A. Manzanares, M. McClelland, A. Torres, and A. Torrez. Storage challenges at Los Alamos National Lab. In *IEEE MSST*, pages 1–5, 2012.

[11] Franck Cappello. Fault Tolerance in Petascale/ Exascale Systems: Current Knowledge, Challenges and Research Opportunities. *International Journal of High Performance Computing Applications*, 23(3):212–226, 2009.

[12] D Tiwari, S Gupta, and S S. Vazhkudai. Lazy Checkpointing: Exploiting Temporal Locality in Failures to Mitigate Checkpointing Overheads on Extreme-Scale Systems. In *44th Annual IEEE/IFIP Int'l Conference on Dependable Systems and Networks*, pages 25 – 36, 2014.

[13] Leonardo Bautista-Gomez, Ana Gainaru, Swann Perarnau, Devesh Tiwari, Saurabh Gupta, Christian Engelmann, Franck Cappello, and Marc Snir. Reducing waste in extreme scale systems through introspective analysis. 2016.

[14] John W. Young. A First Order Approximation to the Optimum Checkpoint Interval. *Communications of the ACM*, 17(9):530–531, 1974.

[15] John Daly. A Model for Predicting the Optimum Checkpoint Interval for Restart Dumps. In *Proceedings of the International Conference on Computational Science*, pages 3–12, 2003.

[16] J.T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computer Systems*, 22(2006):303–312, 2004.

[17] Buddy Bland. Present and Future Leadership Computers at OLCF. In *OLCF User Meeting*, 2015.

[18] 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 *46th Annual IEEE/IFIP Int'l Conference on Dependable Systems and Networks*, pages 311–322, 2016.

[19] Janine C. Bennett, Hasan Abbasi, Peer-Timo Bremer, Ray Grout, and Attila Gyulassy. Combining In-situ and In-transit Processing to Enable Extreme-Scale Scientific Analysis. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 49:1–49:9, 2012.

[20] Jack Dongarra. Impact of architecture and technology for extreme scale on software and algorithm design. In *Department of Energy Workshop on Cross-cutting Technologies for Computing at the Exascale*, 2010.

[21] DOE ASCAC Subcommittee Members. The Opportunities and Challenges of Exascale Computing. Technical report, DOE ASCAC Subcommittee, 2010.

[22] R. Stevens and A. White. A DOE laboratory plan for providing exascale applications and technologies for critical DOE mission needs. In *SciDAC Workshop*, 2010.

[23] John Shalf, Sudip Dosanjh, and John Morrison. Exascale Computing Technology Challenges. In *Proceedings of the 9th International Conference on High Performance Computing for Computational Science*, pages 1–25, 2011.

[24] Tapasya Patki, David K. Lowenthal, Barry Rountree, Martin Schulz, and Bronis R. de Supinski. Exploring Hardware Overprovisioning in Power-constrained, High Performance Computing. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, pages 173–182, 2013.

[25] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross. Storage access characteristics of computational science applications. In *IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–14, 2011.

[26] Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. Understanding and Improving Computational Science Storage Access Through Continuous Characterization. *ACM Transactions on Storage (TOS)*, 7(3):8:1–8:26, 2011.

[27] Debbie Bard, Wahid Bhimji, and Dave Paul. Cori phase 1 burst buffer. Technical report, NERSC Users Group, 2016.

[28] Oak Ridge Leadership Computing Facility. Summit Fact Sheet. Technical report, Oak Ridge National Laboratory, 2014.

[29] Sarp Oral, David A. Dillow, Douglas Fuller, Jason Hill, Dustin Leverman, Sudharshan S. Vazhkudai, Feiyi Wang, Youngjae Kim, James Rogers, James Simmons, and Ross Miller. OLCF's 1 TB/s, Next-Generation Lustre File System. In *Proceedings of Cray User Group Conference*, pages 1–12, 2013.

[30] Charles Lefurgy, Xiaorui Wang, and Malcolm Ware. Power capping: a prelude to power shifting. *Cluster Computing*, 11(2):183–195, 2008.

[31] Anshul Gandhi, Mor Harchol-Balter, Rajarshi Das, Jeffrey O. Kephart, and Charles Lefurgy. Power Capping Via Forced Idleness. In *Proceedings of Workshop on Energy-Efficient Design*, 2009.

[32] Ahmed Hussein, Antony L Hosking, Mathias Payer, and Christopher A Vick. Don't race the memory bus: taming the gc leadfoot. In *Proceedings of the 2015 ACM SIGPLAN International Symposium on Memory Management*, pages 15–27. ACM, 2015.

[33] Yuichi Inadomi, Tapasya Patki, Koji Inoue, Mutsumi Aoyagi, Barry Rountree, Martin Schulz, David Lowenthal, Yasutaka Wada, Keiichiro Fukazawa, Masatsugu Ueda, Masaaki Kondo, and Ikuo Miyoshi. Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 78:1–78:12, 2015.

[34] Neha Gholkar, Frank Mueller, and Barry Rountree. Power Tuning HPC Jobs on Power-Constrained Systems. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*, pages 179–191, 2016.

[35] Martin Dimitrov, Carl Strickland, Seung-Woo Kim, Karthik Kumar, and Kshitij Doshi. Intel Power Governor. https://software.intel.com/en-us/articles/intel-power-governor, July 2012.

[36] Intel. *Intel 64 and IA-32 Architectures Software Developer's Manual*, volume 3B. Intel Corporation, 2015.

[37] Saurabh Agarwal, Rahul Garg, Meeta S. Gupta, and Jose E. Moreira. Adaptive Incremental Checkpointing for Massively Parallel Systems. In *Proceedings of the 18th Annual International Conference on Supercomputing*, pages 277–286, 2004.

[38] Kurt Ferreira, Jon Stearley, James H. Laros, III, Ron Oldfield, Kevin Pedretti, Ron Brightwell, Rolf Riesen, Patrick G. Bridges, and Dorian Arnold. Evaluating the Viability of Process Replication Reliability for Exascale Systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 44:1–44:12, 2011.

[39] Matthew Forshaw, A Stephen McGough, and Nigel Thomas. Energy-efficient checkpointing in high-throughput cycle-stealing distributed systems. *Electronic Notes in Theoretical Computer Science*, 310:65–90, 2015.

[40] Min Li, Sudharshan S. Vazhkudai, Ali R. Butt, Fei Meng, Xiaosong Ma, Youngjae Kim, Christian Engelmann, and Galen Shipman. Functional Partitioning to Optimize End-to-End Performance on Many-core Architectures. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2010.

[41] Fan Zhang, Ciprian Docan, Manish Parashar, Scott Klasky, Norbert Podhorszki, and Hasan Abbasi. Enabling In-situ Execution of Coupled Scientific Workflow on Multi-core Platform. In *IEEE 26th International Parallel and Distributed Processing Symposium*, pages 1352 – 1363, 2012.

[42] D. Li, D. S. Nikolopoulos, K. Cameron, B. R. de Supinski, and M. Schulz. Power-aware MPI task aggregation prediction for high-end computing systems. In *IEEE International Symposium on Parallel Distributed Processing*, pages 1–12, 2010.

[43] D. Li, B. R. de Supinski, M. Schulz, K. Cameron, and D. S. Nikolopoulos. Hybrid mpi/openmp power-aware computing. In 1-12, editor, *IEEE International Symposium on Parallel Distributed Processing*, 2010.

[44] P. E. Bailey, D. K. Lowenthal, V. Ravi, B. Rountree, M. Schulz, and B. R. d. Supinski. Adaptive Configuration Selection for Power-Constrained Heterogeneous Systems. In *43rd International Conference on Parallel Processing*, pages 371–380, 2014.

[45] A. Langer, H. Dokania, L. V. Kalé, and U. S. Palekar. Analyzing energy-time tradeoff in power overprovisioned hpc data centers. In *IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 849–854, 2015.

[46] Zhonghong Ou, Bo Pang, Yang Deng, J.K. Nurminen, A. Ylä-Jääski, and Pan Hui. Energy- and Cost-Efficiency Analysis of ARM-Based Clusters. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 115–123, 2012.

[47] Nikola Rajovic, Paul M. Carpenter, Isaac Gelado, Nikola Puzovic, Alex Ramirez, and Mateo Valero. Supercomputing with commodity CPUs: are mobile SoCs ready for HPC? In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 40:1–40:12, 2013.

[48] Nikola Rajovic, Alejandro Rico, Nikola Puzovic, Chris Adeniyi-Jones, and Alex Ramirez. Tibidabo: Making the case for an ARM-based HPC system. *Future Generation Computer Systems*, 36:322–334, 2014.

[49] Ernily Blem, Jaikrishnan Menon, and Karthikeyan Sankaralingam. Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture*, pages 1–12, 2013.

[50] 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, 2014.

[51] 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, 2015.

[52] Qingyuan Deng, David Meisner, Abhishek Bhattacharjee, Thomas F. Wenisch, and Ricardo Bianchini. CoScale: Coordinating CPU and Memory System DVFS in Server Systems. In *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 143–154, 2012.

[53] O. Sarood, A. Langer, L. Kalé, B. Rountree, and B. de Supinski. Optimizing power allocation to cpu and memory subsystems in overprovisioned hpc systems. In *IEEE International Conference on Cluster Computing*, pages 1–8, 2013.

[54] R. Ge, X. Feng, Y. He, and P. Zou. The case for cross-component power coordination on power bounded systems. In *45th International Conference on Parallel Processing*, pages 516–525, 2016.

[55] Bing Xie, Jeffrey Chase, David Dillow, Oleg Drokin, Scott Klasky, Sarp Oral, and Norbert Podhorszki. Characterizing output bottlenecks in a supercomputer. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 8:1–8:11, 2012.

[56] Huaiming Song, Yanlong Yin, Xian-He Sun, Rajeev Thakur, and Samuel Lang. Server-side i/o coordination for parallel file systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 17:1–17:11, 2011.

[57] Matthieu Dorier, Gabriel Antoniu, Rob Ross, Dries Kimpe, and Shadi Ibrahim. Calciom: Mitigating i/o interference in hpc systems through

cross-application coordination. In *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 155–164, 2014.

[58] Matthew Wachs, Michael Abd-El-Malek, Eno Thereska, and Gregory R. Ganger. Argon: Performance Insulation for Shared Storage Servers. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, 2007.

[59] Ajay Gulati, Irfan Ahmad, and Carl A. Waldspurger. PARDA: Proportional Allocation of Resources for Distributed Storage Access. In *Proccedings of the 7th Conference on File and Storage Technologies*, pages 85–98, 2009.

[60] David Shue, Michael J. Freedman, and Anees Shaikh. Performance Isolation and Fairness for Multi-tenant Cloud Storage. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, 2012.

[61] Dushyanth Narayanan, Austin Donnelly, Eno Thereska, Sameh Elnikety, and Antony Rowstron. Everest: Scaling down peak loads through i/o offloading. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, pages 15–28, 2008.

[62] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. Write offloading: Practical power management for enterprise storage. *ACM Transactions on Storage*, 4(3):10:1–10:23, 2008.

[63] Hasan Abbasi, Matthew Wolf, Greg Eisenhauer, Scott Klasky, Karsten Schwan, and Fang Zheng. Datastager: Scalable data staging services for petascale applications. In *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing*, pages 39–48, 2009.

[64] Samer Al-Kiswany, Abdullah Gharaibeh, and Matei Ripeanu. The Case for a Versatile Storage System. *ACM SIGOPS Operating Systems Review*, 44(1):10–14, 2010.

[65] Lauro Beltrão Costa, Samer Al-Kiswany, Hao Yang, and Matei Ripeanu. Supporting Storage Configuration for I/O Intensive Workflows. In *Proceedings of the 28th ACM International Conference on Supercomputing*, 2014.

[66] L. B. Costa, S. Al-Kiswany, M. Ripeanu, and H. Yang. Support for provisioning and configuration decisions for data intensive workflows. *IEEE Transactions on Parallel and Distributed Systems*, 27(9):2725–2739, 2016.

[67] S. Thapaliya, P. Bangalore, J. Lofstead, K. Mohror, and A. Moody. Managing I/O Interference in a Shared Burst Buffer System. In *45th International Conference on Parallel Processing*, 2016.

[68] Teng Wang, Kathryn Mohror, Adam Moody, Kento Sato, and Weikuan Yu. An ephemeral burst-buffer file system for scientific applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 69:1–69:12, 2016.

[69] Anthony Kougkas, Matthieu Dorier, Rob Latham, Rob Ross, and Xian-He Sun. Leveraging Burst Buffer Coordination to Prevent I/O Interference. In *IEEE 12th International Conference on eScience*, 2016.

[70] T. Wang, S. Oral, M. Pritchard, B. Wang, and W. Yu. Trio: Burst buffer based i/o orchestration. In *IEEE International Conference on Cluster Computing*, pages 194–203, 2015.

[71] Arrhenius equation. https://en.wikipedia.org/wiki/Arrhenius_equation.

[72] Top500 List. https://www.top500.org/lists, 2016.

[73] Sameer S. Shende and Allen D. Malony. The Tau Parallel Performance System. *International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.

[74] Philip J. Mucci, Shirley Browne, Christine Deane, and George Ho. PAPI: A Portable Interface to Hardware Performance Counters. In *Proceedings of Department of Defense HPCMP Users Group Conference*, 1999.

[75] Devesh Tiwari, Saurabh Gupta, James Rogers, Don Maxwell, Paolo Rech, Sudharshan Vazhkudai, Daniel Oliveira, Dave Londo, Nathan DeBardeleben, Philippe Navaux, et al. Understanding gpu errors on large-scale hpc systems and the implications for system design and operation. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 331–342. IEEE, 2015.

[76] Saurabh Gupta, Devesh Tiwari, Christopher Jantzi, James Rogers, and Don Maxwell. Understanding and exploiting spatial properties of system failures on extreme-scale hpc systems. In *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*, pages 37–44. IEEE, 2015.

[77] Bin Nie, Devesh Tiwari, Saurabh Gupta, Evgenia Smirni, and James H Rogers. A large-scale study of soft-errors on gpus in the field. In *High Performance Computer Architecture (HPCA), 2016 IEEE 22nd International Symposium on*, 2016.

[78] Devesh Tiwari, Saurabh Gupta, George Gallarno, Jim Rogers, and Don Maxwell. Reliability lessons learned from gpu experience with the titan supercomputer at oak ridge leadership computing facility. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 38. ACM, 2015.

[79] Nosayba El-Sayed, Ioan A Stefanovici, George Amvrosiadis, Andy A Hwang, and Bianca Schroeder. Temperature management in data centers: why some (might) like it hot. *ACM SIGMETRICS Performance Evaluation Review*, 40(1):163–174, 2012.

[80] Paul Ellerman. Calculating Reliability using FIT and MTTF: Arrhenius HTOL Model. Technical report, microsemi, 2012.

[81] Galen Shipman and et al. A next-generation parallel file system environment for the OLCF. In *Proceedings of the Cray User Group Conference*, 2012.

[82] NVIDIA. *NVML API REFERENCE MANUAL*. NVIDIA Corporation, 2012.

[83] Jiesheng Wei, A. Thomas, Guanpeng Li, and K. Pattabiraman. Quantifying the Accuracy of High-Level Fault Injection Techniques for Hardware Faults. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, pages 375–382, 2014.

[84] CORAL Benchmark Codes. *https://asc.llnl.gov/CORAL-benchmarks*.

[85] G. Shipman, D. Dillow, S. Oral, and F. Wang. The Spider Center Wide File System: From Concept to Reality. In *In Proceedings of the Cray User Group*, 2009.

[86] Oak Ridge Leadership Computing Facility. OLCF Annual Report. Technical report, Oak Ridge National Laboratory, 2013.

[87] Sarp Oral, James Simmons, Jason Hill, Dustin Leverman, Feiyi Wang, Matt Ezell, Ross Miller, Douglas Fuller, Raghul Gunasekaran, Youngjae Kim, Saurabh Gupta, Devesh Tiwari, Sudharshan S. Vazhkudai, James H. Rogers, David Dillow, Galen M. Shipman, and Arthur S. Bland. Best practices and lessons learned from deploying and operating large-scale data-centric parallel file systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014.

[88] R. Miller, J. Hill, D. A. Dillow, R. Gunasekaran, S. Galen, and D. Maxwell. Monitoring tools for large scale systems. In *In Proceedings of the Cray User Group*, 2010.

[89] John S. Bucy, Jiri Schindler, Steven W. Schlosser, and Gregory R. Ganger. *The DiskSim Simulation Environment Version 4.0 Reference Manual*. Carnegie Mellon University, 2008.

[90] Edward K. Lee and Randy H. Katz. An Analytic Performance Model of Disk Arrays. In *Proceedings of the 1993 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 98–109, 1993.

[91] Peter M. Chen and Edward K. Lee. Striping in a raid level 5 disk array. In *Proceedings of the 1995 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pages 136–145, 1995.