How much SSD is useful for resilience in supercomputers

A dissertation submitted to
the faculty of the division of the physical sciences
in candidacy for the degree of
master’s

Department of computer science

by

Aiman Fang

Chicago, Illinois
2015
To see a World in a Grain of Sand
And a Heaven in a Wild Flower

– William Blake
# TABLE OF CONTENTS

LIST OF FIGURES .............................................................. vii
LIST OF TABLES .............................................................. viii
ACKNOWLEDGMENTS ........................................................... ix
ABSTRACT ................................................................. x

1 INTRODUCTION .............................................................. 1

2 BACKGROUND .............................................................. 6
   2.1 Supercomputer Basics .................................................. 6
      2.1.1 Supercomputers ................................................... 6
      2.1.2 Burst Buffer System ............................................. 7
      2.1.3 Solid State Drives .............................................. 9
      2.1.4 Batch Scheduling ................................................ 10
      2.2 Resilience in Supercomputers .................................... 12
         2.2.1 Checkpoint/Restart and Derivatives ....................... 12
         2.2.2 Optimal Checkpoint Interval ............................... 13

3 PROBLEM ................................................................. 14
   3.1 SSD Lifetime Allocation Problem ................................... 14
   3.2 SSD Lifetime Provisioning Problem ................................ 16
   3.3 Summary ............................................................ 16

4 MODEL ................................................................. 18
   4.1 Assumptions ........................................................ 18
   4.2 Model ............................................................... 19
   4.3 Allocation Problem ................................................ 22
      4.3.1 Allocation Objectives ......................................... 22
      4.3.2 Formulation and Solutions .................................. 23
   4.4 Summary ............................................................ 25

5 APPLYING THE MODEL .................................................. 26
   5.1 System Model ........................................................ 26
      5.1.1 Simple System ................................................. 26
      5.1.2 Realistic System ............................................... 27
   5.2 Workload Model ..................................................... 27
      5.2.1 Simple Workloads .............................................. 28
      5.2.2 Realistic Workloads .......................................... 28
   5.3 Metrics .............................................................. 29
   5.4 Studies ............................................................. 30
      5.4.1 SEB Allocation on Simple workloads ....................... 30
# LIST OF FIGURES

2.1 The typical organization of the contemporary supercomputers .................. 7
2.2 The overview of supercomputers with a tier of burst buffers .................... 8
2.3 Mira Activity, April 30, 2015 .................................................. 11

3.1 An example of SSD lifetime allocation on jobs ..................................... 14

4.1 The Wall Clock Time Composition ................................................. 20

5.1 SSD Service Ratio, Job Efficiency and System Efficiency vs. Job Size (SEB) .... 30
5.2 SSD Service Ratio, Job Efficiency and System Efficiency vs. Solve Time (SEB) . 30
5.3 SSD Service Ratio, Job Efficiency and System Efficiency vs. Checkpoint Size Ratio (SEB) ......................................................... 30
5.4 SSD Service ratio and Job Efficiency vs. Job Size (SEB) .......................... 32
5.5 SSD Service ratio and Job Efficiency vs. Solve Time (SEB) ....................... 33
5.6 SSD Service ratio and Job Efficiency vs. Checkpoint Size Ratio (SEB) ........... 34
5.7 SSD Service Ratio and Job Efficiency vs. Job Size ............................... 35
5.8 SSD Service Ratio and Job Efficiency vs. Solve Time ............................. 35
5.9 SSD Service Ratio and Job Efficiency vs. Checkpoint Size Ratio ................. 36
5.10 SSD Service Ratio and Job Efficiency on a Mixed Workload (CSR denotes Checkpoint Size Ratio) ...................................................... 36
5.11 System Efficiency vs. SB, JEB, and SEB on varied workloads .................... 37
5.12 SSD Provisioning Ratio to achieve 90%, 95%, 98% System Efficiency, for varied workloads ................................................................. 38
# LIST OF TABLES

2.1 Time to Take a Checkpoint on Systems of Top500 ........................................... 8
2.2 Comparison of DRAM, SSD, and HDD ................................................................. 9

4.1 Notation .................................................................................................................. 19

5.1 Simple System Configuration .................................................................................. 27
5.2 Simple Workloads .................................................................................................... 28
5.3 Trace-Based Realistic Workloads ............................................................................ 29
ACKNOWLEDGMENTS

I would like to take this opportunity to express my gratitude to my advisor Prof. Andrew A. Chien. He gives me patiently guidance all this time and I have learned a lot from him during my graduate study. The most important things I realized are scientific thinking and rigorous working. I will keep them in mind in future study and research.

I also would like to thank the other two committee members, Prof. Haryadi Gunawi and Prof. Shan Lu for their kindness to be on my committee and useful suggestions on my work.

And I want to thank folks from LSSG team, especially Hajime Fujita, Nan Dun, and Zachary Rubenstein. Working with them in the past two years, I gained not only skills but also a lot of fun. Beyond that, I need to thank my friends and my mentor Zhao Zhang for their support.

At last, I want to thank my dear parents for their unconditional love, which supports me all my life.
ABSTRACT

Future extreme-scale (exascale) systems are predicted to have high error rates. Jobs running on such systems will encounter frequent failures. Checkpoint/restart based on non-volatile memories in the form of SSD’s (Solid State Disks) and used as burst buffers is a promising approach to meet resiliency need. However, because of SSD’s high cost and limited lifetime, understanding their effective use and appropriate provisioning is critical.

We explore two problems. First, for a set of jobs, how to allocate limited SSD lifetime to increase their efficiency in the face of failures? (Allocation). Second, given a supercomputer system with a particular error rate, how much SSD lifetime is worth buying to increase resilience and thereby system efficiency? (Provisioning).

We derive a model that captures the characteristics of jobs and systems, and use it to formulate the Allocation problem. We use this model to study the allocation and provisioning questions under a variety of mission-oriented policy scenarios including job size-count, equal job efficiency and maximum system efficiency. We first apply the model on realistic workloads to understand the impact of job characteristics and mix on allocation. Second, we explore properties and performance of three allocation policies on various workloads. Third, we explore appropriate SSD provisioning to achieve acceptable system efficiency.

Our results first show that the SSD lifetime constraint changes the checkpoint interval, and thereby the achievable job and system efficiency. Second, the system efficiency can be increased remarkably by considering a global perspective (workload mix) for SSD lifetime allocation. Finally, further results suggest that underprovisioning SSD lifetime (only 10-20% of requirements without resource constraint) is sufficient to produce 90% system efficiency at failure rates three times that of current systems.
CHAPTER 1
INTRODUCTION

Scientific computations have driven the development of supercomputers. As of November 2014, the supercomputer ranked first in top 500 list [20], Tianhe-2 [19], is comprised of 3,120,000 cores and produces the peak performance of 54,902 TFlop/s (Tera Floating operations per second). To meet the increasing demands of computing power, the supercomputers continues growing towards exascale. However, each component of in a computer system may periodically experience errors coming from various resources such as radiation or circuit noise. Given that supercomputers consist of a considerably number of components, they would experience errors frequently and even stop working in worst cases.

Current supercomputers experience a failure roughly every day. While future exascale systems are projected to have higher error rates, producing MTBF (Mean Time Between Failures) that is less than an hour [28, 29, 54]. On the other hand, many scientific applications utilize hundreds of computing nodes of systems and runs for hours. Such large jobs will encounter frequent failures given the high error rates of future systems. Therefore resilience is identified as one of the four key challenges to achieve exascale computing [48].

For decades, checkpoint/restart (CR) has been the classical HPC (high performance computing) approach to fault-tolerance. CR periodically takes a checkpoint of the system state [1] or important data of application [6] and preserve the checkpoint in reliable storage such as hard disk drive. If an error is detected, applications will read the checkpoints from disk and restart computation. CR helps applications only to rework the lost computations since last checkpoint instead of reworking from beginning.

As failure rates grow up, the checkpoint frequency also needs to increase, otherwise more work would be lost due to frequent failures. However, limited disk-storage bandwidth in exascale system is a critical bottleneck for checkpoint-based resilience. Because checkpointing time depends on the checkpoint size and disk bandwidth, limited bandwidth implies that it takes a long time for applications to write a checkpoint to disks. The lack of ability to
write frequent checkpoints harms resilience. It is anticipated that future system bandwidth as high as 60 TiB/s is needed to drain application data [45]. In contrast current systems can only supply disk bandwidth in magnitude of GiB/s.

A promising approach to bring the gap between disk bandwidth and future bandwidth demands is burst buffers [45, 24, 27]. The burst buffer is a proposed storage layer located between memory and disks and is usually comprised of non-volatile memories (NVM) in the form of SSD (Solid State Disks). First, burst buffers guarantee the persistence of preserved data without power. Meanwhile they provide higher bandwidth than disks, absorbing checkpoint data faster than possible with the disk-based file systems. Emerging CR systems (SCR [46] and FTI [23]) exploit these burst buffers to increase checkpoint frequency, and reduce checkpoint costs. Burst buffers are just now (2016) reaching deployment [12, 21, 37, 4, 25].

Although NVM and SSD have advantages in bandwidth, they are relatively expensive ($/GB) compared to rotating disk-drives. The price of SSDs has continued to decline, however consumer-grade SSDs are still six to seven times more expensive than HDDs with same capacity [15]. In addition, flash-based systems such as SSD have to be erased before being written. And these systems have limited lifetimes, wearing out after a modest number of erase/write cycles [32]. For instance, the densest, cheapest MLC (Multi-Level Cell) flash has write endurance as low as 10,000 cycles. Consequently, utilizing limited SSD lifetime efficiently is an important problem. In this thesis we focus on the efficient utilization of SSD lifetime for checkpointing in supercomputers.

In current HPC systems, applications are allowed to independently determine their checkpoint size and frequency without coordination amongst themselves or with the system. Each application then employs a model to pick a checkpoint frequency that maximizes its runtime efficiency [60, 34]. Such independent choice is suitable for rotating disk systems But with limited lifetime, coordinated choices are needed for non-volatile memories. This need for coordination gives rise to several other questions.

1. For a set of jobs, how to allocate SSD lifetime to increase their efficiency in the face of
failures? (Allocation).

2. Given a supercomputer with a particular error rate, how much SSD lifetime is worth buying to increase resilience and thereby system efficiency? (Provisioning).

The corollary of the latter question is how much efficiency is gained for each increment of SSD lifetime.

To answer these questions, we first derive an analytical model that describes job characteristics such as job size, run time, and system characteristics such as system size, memory size, SSD bandwidth. The model captures the behavior of jobs sharing SSD lifetime and taking checkpoints in burst buffers.

We then consider a variety of mission-oriented policy scenarios of allocation.

- **Size-based (SB) Allocation** is to achieve job-size fairness, that is, allocating resource proportional to job size. Large jobs running on hundreds of nodes get more allocation and vice versa.

- **Job-efficiency based (JEB) Allocation** is to equalize job efficiency of jobs in a set. Users of supercomputers desire that their jobs are treated fairly with others.

- **System-efficiency based (SEB) Allocation** is to maximize system efficiency. System administrators would like the property. High system efficiency suggests efficient proceeding of jobs and therefore reduces resource cost.

We formulate these three SSD lifetime allocation problems using the model. They are equivalently transformed into nonlinear systems and optimization problems with constraints. We derive solutions to them respectively. In order to apply the model, we give the jobs properties and system configuration as input, and the model produces SSD lifetime allocation for each job under the designated policy.

For evaluation, we first utilize a simple system configuration and synthetic workloads consisting of two jobs. We further apply the model to realistic systems and workloads. We
describe the supercomputer system of interest and workload properties. We study the impact of job characteristics on allocation and explore properties and performance of three different policies on various workloads. Finally, we explore appropriate SSD provisioning to achieve acceptable system efficiency.

Our results give insight into appropriate policies for SSD usage for resilience in future supercomputers that include burst buffers, and into the cost-effective approach to provisioning burst buffers on such systems.

Specific contributions of the paper include:

- Derive a model to determine optimal SSD lifetime allocation for a variety of objectives, including job-size fairness (size-based allocation), equal job efficiency (job-efficiency based allocation) and maximum system efficiency (system-efficiency based allocation).

- With size-based and system-efficiency based allocation, large size jobs suffer 40% lower job efficiency than small size jobs.

- Checkpoint size ratio has similar but smaller effect with size-based and system-efficiency based allocation, delivering 6 to 10% lower efficiency for large jobs when compared to small.

- Job-efficiency based allocation eliminates job-size unfairness, but must allocate 50% more lifetime (a 3% increase from 6.25% to 9.25%) to large jobs to do so.

- Job-efficiency based allocation’s fair efficiency comes at a cost, decreasing system efficiency by as much as 14% from the best possible. Similarly, size-based allocation also falls short, decreasing system efficiency by as much as 5.5%.

- On cost-effective provisioning, only 10-20% of the optimal lifetime is needed to achieve 90% system efficiency – even at failure rates 3x that of current systems. 95% system efficiency is expensive, requiring a doubling of provisioned lifetime.
The rest of the thesis is organized as follows. Chapter 2 presents the background, and Chapter 3 discuss the two problems we explore in this thesis. Chapter 4 describes our model, and shows how it can be used to derive solutions for varied objectives. Chapter 5 describes workload and system models, and then applies the model to derive insights into SSD allocation policies and consequences. We discuss related work and put our results in context in Chapter 6. At last, we summarize our results and describe future directions in Chapter 7.
CHAPTER 2
BACKGROUND

In this chapter, we provide the background of our work. We first introduce some basic concepts of supercomputers and burst buffer systems. We briefly discuss the important characteristics of SSD and introduce the batch scheduling utilized by supercomputers. Finally, we discuss the resilience challenges in supercomputers and present previous work on the optimal checkpoint interval which brings about our work.

2.1 Supercomputer Basics

2.1.1 Supercomputers

A supercomputer is a computer with a high-level computational capacity [18]. The first supercomputer, the Control Data Corporation (CDC) 6600, was designed by Seymour Cray and released in 1964. The system could effectively operate at 40 MHz and produce a peak performance of 3 million FLOPS (floating point operations per second) [3]. While the original supercomputers were composed of a few processors, systems with thousands of processors began to appear in the 1990s. As of November 2014, Tianhe-2 from National Super Computer Center of China ranked 1st in top 500 supercomputers of the world [19]. It has 3 million cores and produces peak performance of 54,902 tera-FLOPS.

Scientific and engineering computations have driven the demand for large-scale computing power. Supercomputers have been used for various scientific researches including computational chemistry, molecular dynamics modeling, earthquake simulation and nuclear testing. The parallelism demand of these scientific applications scales up to thousands of nodes to achieve high-performance computing. For example, molecular dynamics (MD) codes are an important computational method in a wide variety of areas of biology, chemistry, and physics, which can simulate the physical movements of millions of atoms. Such MD codes highly desire supercomputers’ computing power for parallel simulation. One popular MD
codes, ddcMD (domain decomposition molecular dynamics), developed by Lawrence Livermore National Laboratory, achieved 101.7 TFlops over a 7 hour run on 131,072 processors of IBM BlueGene/L computer. The 2005 Gordon Bell Prize was awarded for this accomplishment [11]. As higher computing power is desired, the development of exascale systems is one the way.

2.1.2 Burst Buffer System

Figure 2.1 shows a typical organization of the contemporary supercomputers. Compute nodes are composed of processors and memory, performing float point and numeric calculations. Usually there is no local permanent storage attached with each node. Instead the shared massive external storage devices (i.e. HDD) are controlled by I/O nodes, which provides I/O services to compute nodes. The data generated by applications on compute nodes are written to external storage by I/O nodes. Parallel file systems (PFS) are running on I/O nodes and HDDs to manage the massive storage.

![Diagram of supercomputer organization](image)

Figure 2.1: The typical organization of the contemporary supercomputers

However, the limited data transfer bandwidth of HDD is a primary obstacle that prevents performance scaling. Prior studies [42, 51, 49] have observed bursty I/O behavior of HPC applications, that is, applications require to write massive data to disk in a short time. One example of bursty I/O behavior is taking checkpoints. Applications write a copy of
data or their memory states to disks for backup in case failures occur. The checkpointing time becomes the bottleneck of performance as the size of checkpoints grow up and further saturate the bandwidth of disk. It takes as long as 30 minutes and up to hours to write one checkpoint. Table 2.1 shows the checkpoint time in several popular systems reported by previous study [28]. As this trend continues, the checkpoint time will finally exceed MTBF, implying that there is no chance for the job to successfully complete since failures occur faster than checkpointing.

Table 2.1: Time to Take a Checkpoint on Systems of Top500

<table>
<thead>
<tr>
<th>Systems</th>
<th>Peak Performance</th>
<th>Checkpoint Time (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLNL Zeus</td>
<td>11 teraFLOPS</td>
<td>26</td>
</tr>
<tr>
<td>LLNL BlueGene/L</td>
<td>500 teraFLOPS</td>
<td>20</td>
</tr>
<tr>
<td>Argonne BlueGene/P</td>
<td>500 teraFLOPS</td>
<td>30</td>
</tr>
<tr>
<td>LANL RoadRunner</td>
<td>1 petaFLOPS</td>
<td>20</td>
</tr>
</tbody>
</table>

Burst buffers have been proposed as a high-bandwidth, storage tier between compute nodes and disk storage to reduce checkpointing overhead without requiring large increases in the parallel file system bandwidth, as shown in Figure 2.2. Examples including Bent [24], Brown [27], Liu [45], and other works [53, 50, 41] have explored utilizing burst buffers to improve the resilience.
The burst buffer such as SSD has higher bandwidth and lower latency compared to PFS. It serves as I/O offloading layer that absorbs bulk data produced by applications, while seamlessly draining the data to the PFS in the background. With this layer of burst buffers, applications can dump data quickly and return to computation without waiting data to be moved to disk.

The advanced architecture features of burst buffer systems have been recognized. Several next generation of supercomputers are going to deploy burst buffers. Cori [12] is National Energy Research Scientific Computing (NERSC) Center’s next supercomputer system (NERSC-8). The phase 1 system will provide approximately 750TB of burst buffer storage. Summit [17] is Oak Ridge National Laboratory’s next high performance supercomputer. It will consist of approximately 3,400 nodes, each with 512 GB memory and 800 GB NVRAM as burst buffers.

2.1.3 Solid State Drives

A solid-state drive (SSD) or solid-state disk is a data storage device that stores data persistently [15]. As emerging nonvolatile memory technologies, SSD have some attractive features: relative low read/write latency, high read/write bandwidth and ability to retain data without power. Table 2.2 lists read latency, write latency, data transfer rate and endurance of DRAM, SSD and HDD.

<table>
<thead>
<tr>
<th></th>
<th>DRAM</th>
<th>SSD</th>
<th>HDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Latency</td>
<td>~10ns</td>
<td>5\mu s – 50\mu s</td>
<td>~4ms</td>
</tr>
<tr>
<td>Write Latency</td>
<td>~10ns</td>
<td>2 – 3ms</td>
<td>~4ms</td>
</tr>
<tr>
<td>Data Transfer Rate</td>
<td>1 - 10 GB/s</td>
<td>200 - 600 MB/s</td>
<td>50 - 120 MB/s</td>
</tr>
<tr>
<td>Endurance (erase/write cycles)</td>
<td>$10^{15}$</td>
<td>$10^4 – 10^5$</td>
<td>$10^{15}$</td>
</tr>
</tbody>
</table>

On the other hand, both SSD’s and raw non-volatile memories such as NAND flash are more expensive per unit capacity than HDD (Hard Disk Drive), roughly six to seven times. In addition, these systems have lifetimes of 10,000 to 100,000 write cycles [39, 32]. We take
Summit system [17] as an example. Each node of Summit system has 512 GB memory and additional 800 NVRAM as burst buffer. Suppose the number of write cycle is $10^4$. If the system writes a snapshot of memory every hour to burst buffers, optimistically the lifetime of burst buffer can only last for less than two years. Therefore the effective use of SSD lifetime is an important consideration in deciding checkpoint frequency.

2.1.4 Batch Scheduling

In supercomputers, users submit their jobs with resource requests such as number of cores/nodes and machine hours needed. Due to resource limitation, jobs usually cannot start immediately after job submission. Instead they are placed in the waiting queue controlled by scheduler. The scheduler decides which jobs to run on the basis of request run time, how long a job has been waiting, and fair sharing of resources among different users.

Figure 2.3a shows the activity of Mira (Blue Gene/Q) system [9] of Argonne Leadership Computing Facility [10]. The squares represent partitions of Mira, the colored ones among which are occupied by different jobs. The concurrent running jobs are listed in Figure 2.3b. The system is fully occupied so that jobs are placed in the queue as shown in Figure 2.3c.

Batch scheduling is widely deployed in supercomputer systems. Batch scheduling of jobs in the queue consists of two activities [40]. First the scheduler selects a job or a set of jobs from the queue to execute. Second the scheduler allocate resources such as memory, CPU, power and bandwidth among the set of jobs that have be selected for execution.

In our work we focus on resource allocation across jobs within a workload with the objective to optimize system and job performance.

**Workload:** A workload is composed of a stream of jobs which share the supercomputer resources. The scheduler makes resource allocation decision among jobs within a workload. Popular schedulers include SLURM workload Manager [14] and PBS (Portable Batch System) [13].
Figure 2.3: Mira Activity, April 30, 2015
2.2 Resilience in Supercomputers

Supercomputer systems are growing towards exascale to meet the need of scientific computation. Today these large-scale systems have already been comprised of millions of components, leading to higher error rates.

Reported by previous study [36] the Blue Waters system [2] experienced 1025 failures which result in job failures in 261 days. The system MTTI (mean time to interrupt) is roughly 6.11 hour. The Blue Waters system has 26,864 compute nodes. That gives the single node failure rate as 6,100 FIT (Failure In Time) per node. FIT is used to quantify the number of failures in billion machine hours, which serves as a metric to measure failure rates.

Snir et al. [54] predict the future failure rate range as 500 to 20,000 FIT per node. 20,000 FIT per node failure rate gives that the estimated MTBF is 2 days for a 1,000-node system, 5 hours for a 10,000-node system, and 30 minutes for a 100,000-node system. Resilience is identified as one key challenges towards exascale computing.

2.2.1 Checkpoint/Rerstart and Derivatives

The broadly used approach to achieve resilience is Checkpoint/Rerstart (CR). Applications take snapshots of states and write checkpoints to stable storage such as hard disk drive (HDD). Upon a failure occur, applications read checkpoints from disk and restart computation.

As we discussed in previous section, the checkpointing time is the critical bottleneck of performance as checkpoints size grow up and further saturate the bandwidth of disk. To meet exascale resilience challenges, numerous researchers have proposed multi-level checkpointing systems [38, 57, 46, 23]. The primary idea of multi-level checkpoint is to capture inexpensive but less-resilient checkpoints in memory and intermediate fast storage such as burst buffer, and finally expensive but most-resilient checkpoints in the parallel file system.
These approaches reduce resilience overhead by avoiding frequent writes to the parallel file system.

2.2.2 Optimal Checkpoint Interval

Frequent checkpointing increases performance overhead and in our case, the consumption of SSD lifetime, but infrequent checkpoints increase work loss due to failures. So, the optimization objective is to minimize the total checkpointing overhead and rework. Young [60] used a first order approximation to derive the optimum checkpoint interval, producing the optimum checkpoint interval $\tau$, as shown in Equation (2.1), where $\delta$ denotes the time to write one checkpoint to storage, and $M$ denotes MTBF.

$$Optimal \; Interval = \sqrt{2\delta M}.$$ (2.1)

This simple formula is easy to apply, and typically applications use this interval or slightly shorter, to take into slight inaccuracies in Young’s assumptions such as failure distribution to simplify the model.

Daly [33, 34] refined Young’s work, using a higher order approximation to compute optimum checkpoint interval. Daly’s approach minimizes overall execution time (wall clock time) which is computed as shown below:

$$Wall \; clock \; time = solve \; time + dump \; time + rework \; time + restart \; time.$$ (2.2)

Where solve time is the application work time, and dump time is the time to write checkpoints to storage. Rework time is the work lost due to a failure, and restart time is the time to load a checkpoint and for the job to resume work. Our models build on Young and Daly’s work.
CHAPTER 3
PROBLEM

In previous chapter, we explain the idea of burst buffer as a solution for checkpoint/restart based resilience. We briefly discuss the characteristics of SSD, especially its limited lifetime. Based on this background, we describe the problems we explore.

3.1 SSD Lifetime Allocation Problem

The utilization of SSD lifetime we consider is taking checkpoints. Although other purposes of using burst buffers exist, they are not in the scope of this work.

We first show a simple example, as shown in Figure 3.1. There are some SSD lifetime available in system and two jobs need to share them for checkpointing. Job A and B run on 4 nodes and 8 nodes respectively. They have different runtime and checkpoint size. With SSD lifetime allocated, Job A and B are restricted to take certain number of checkpoints. Therefore their checkpoint intervals are determined.

![Figure 3.1: An example of SSD lifetime allocation on jobs](image)

This example gives rise to several questions. First, how to allocate the given amount of SSD lifetime to these two jobs. Second, how to guarantee job fairness. Third, can the job
survive failures and how fast can it complete. Fourth, in addition to job performance, how to improve the system performance, that is, the system can proceed both jobs quickly.

There are numbers of potential choices. For instance, allocating more SSD lifetime to Job B implies it can write frequent checkpoints on burst buffers to guarantee resilience. However, as suggested by Young and Daly’s work, the optimal checkpoint interval to achieve maximum job performance depends on failure rate and checkpoint size. More frequent checkpointing will not bring any benefit to performance instead it wastes time and slows down the progress of jobs.

On the other side, given that the available SSD lifetime is limited, allocating more to Job B indicates allocating less to Job A. Job A lacking SSD lifetime resource is limited to taking checkpoints. As a result, it may suffer large amount of work loss due to failures. The work loss due to failure is proportional to checkpoint interval. The checkpoint interval $\tau_1$ of Job A is more expanded than $\tau_2$ of Job B. In the worst case Job A would fail to complete due to lack of checkpoints.

We conclude from the above example that inappropriate SSD lifetime allocation will lead to bad performance both for jobs and supercomputers. Thus it is interesting to learn what factors should be taken into consideration for appropriate SSD lifetime allocation. As we discussed in Chapter 2, supercomputers exploit the batch scheduling systems to decide which jobs to execute and then allocate resource among them. Those allocation choices are made by considering both job properties such as requested nodes and run time and available system resources. In this thesis, we also want to explore how job and system characteristics affect SSD lifetime allocation.

Based on the above discussion, we define the first problem as follows.

**Allocation**: given limited SSD lifetime and a set of jobs in a workload, how should we allocate SSD lifetime to increase their efficiency in face of failures?
3.2 SSD Lifetime Provisioning Problem

In Allocation problem, the available amount of SSD lifetime is given as a known constraint. In practice, provisioning is carefully decided by system administrators in order to minimize the cost while providing satisfactory services.

We first consider provisioning SSD lifetime for only one job. Without resource constraint, a job would take checkpoints in the frequency suggested by Young’s Formula 2.1. Therefore given its estimated runtime and checkpoint size, we can calculate the number of checkpoints it needs to take and further the amount of SSD lifetime required. We define this amount derived from Young’s formula as the optimum SSD lifetime because it delivers best job performance as proved by Young. We further define overprovisioning as provisioning more than the optimum amount and underprovisioning as provisioning less than the optimum amount.

Overprovisioning will not bring any benefit as jobs can not further improve their performance by taking more checkpoints. Underprovisioning may reduce jobs’ performance and resilience. However, the degree of degradation remains unknown. There may be opportunities to sacrifice little performance to save SSD lifetime.

In addition, in supercomputers provisioning are made for a collection of jobs or workloads, which requires coordination among jobs and also systems. It is interesting to explore how a increment of SSD lifetime affects the system efficiency rather than a single job.

We define the second problem as follows.

**Provisioning:** given a supercomputer system with a particular failure rate, how much SSD lifetime is worth buying to increase resilience and thereby system efficiency?

3.3 Summary

In this chapter, we have introduced the two problems we explore in this thesis. First is how to allocate SSD lifetime among jobs within a workload with objectives to maximize job
and system efficiency. Second is how to provision supercomputers such that it meets the resilience needs and increases system efficiency.
CHAPTER 4
MODEL

To solve the Allocation and Provisioning problem, we build a mathematical model to formulate them systematically. In this section we first explain the assumptions we have made for the model and then establish the model. For the Allocation problem, we propose three allocation objectives and then derive solutions to each allocation policy respectively.

4.1 Assumptions

We capture the essential characteristics of systems and workloads and simplify the model with following assumptions.

- Failures are detected at the instant of their occurrence. In reality, there may be a latency between failure occurrence and detection. In some cases, failures escape detection and leads to incorrect results which are referred as SDC (silent data corruption). In our model, we focus on failures that are detected immediately.

- Failures arrive randomly, that is, they follow a Poisson distribution with constant failure rate. This assumes the events of failures are memoryless, while some studies show that failures may be correlated, revealing locality. We deploy the classical failure model as in Young’s work.

- There are no failures during checkpointing and recovery. Daly [34] and Tantawi [55] have considered models allowing failures during checkpointing and recovery. However, the derived solutions are intractable and non-trivial to realize in practice. We argue that in future burst buffer systems, the fraction of checkpointing and recovery is trivial compared to real work time of applications. Thus it is reasonable to make this assumption.
• The job interrupts arise from hardware failures, and thus are proportional to the number of nodes used by the job. We do not consider other possible error sources, such as software errors.

• Restart time is zero. The restart time includes reinitializing applications and probably underlying libraries utilized by applications. However it is a small fraction of runtime and the probability of failing is relative small. Daly’s work [34] found that the restart time has no contribution for optimum checkpoint interval under higher order model. For simplicity, we exclude restart time in our model.

4.2 Model

Table 4.1: Notation

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>Number of nodes of the system</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>Node failure rate</td>
</tr>
<tr>
<td>$L$</td>
<td>Available SSD lifetime</td>
</tr>
<tr>
<td>$M$</td>
<td>Number of jobs in total</td>
</tr>
<tr>
<td>$n_i$</td>
<td>Number of nodes used by $i$-th job</td>
</tr>
<tr>
<td>$T_{w,i}$</td>
<td>Wall clock time</td>
</tr>
<tr>
<td>$T_{s,i}$</td>
<td>Solve time, time spent on real work</td>
</tr>
<tr>
<td>$\tau_i$</td>
<td>Checkpoint interval</td>
</tr>
<tr>
<td>$\tau_{i,opt}$</td>
<td>Optimum checkpoint interval by Young</td>
</tr>
<tr>
<td>$\delta_i$</td>
<td>Dump time, time of writing one checkpoint</td>
</tr>
<tr>
<td>$\alpha_i$</td>
<td>Job interrupt rate</td>
</tr>
<tr>
<td>$s_i$</td>
<td>Checkpoint size</td>
</tr>
<tr>
<td>$l_i$</td>
<td>SSD lifetime allocated to $i$-th job</td>
</tr>
<tr>
<td>$l_{i,opt}$</td>
<td>Optimum SSD lifetime</td>
</tr>
</tbody>
</table>

Table 4.1 summarizes notation and variable definitions we use to build the model. Based on the cost function defined by Daly’s Equation (2.2), we optimize the wall clock time with given limited SSD lifetime as a constraint.

The composition of wall clock time is shown in Figure 4.1.
We study a large-scale system with $N$ nodes, and node failure rate of $\lambda$ failures per hour (the inverse of MTBF, i.e. $MTBF = 1/\lambda$). The system has a limited SSD lifetime of $L$ gigabytes. In a workload $M$ jobs of varying characteristics run concurrently on the system. In the $M$ jobs, the $i$-th runs on $n_i$ nodes, where $\sum_{i=1}^{M} n_i \leq N$. The wall clock time of a job is $T_{w,i}$, and the solve time is $T_{s,i}$. Denoting the checkpoint interval as $\tau_i$, the job takes $(\frac{T_{s,i}}{\tau_i} - 1)$ total checkpoints. Including a checkpoint for the initial computation state, then combining the total number of checkpoints and checkpointing time, total checkpoint time of a job is (where $\delta_i$ is the dump time, i.e., the time to write a checkpoint)

$$\frac{T_{s,i}}{\tau_i} \delta_i. \quad (4.1)$$

Failures occur as Poisson arrivals during real work time $\tau_i$, and recovery is achieved by resuming computation from the previous checkpoint. The lost work due to a failure is a fraction of $\tau_i$. With Poisson arrivals, a good approximation for lost work is $\frac{1}{2} \tau_i$ as in Daly’s first order model [34].

The job interrupt rate, $\alpha_i$, is proportional to the number of nodes, the total number of failures is

$$T_{s,i} \alpha_i. \quad (4.2)$$
Which gives a total rework time of

\[ \frac{1}{2} \tau_i T_{s,i} \alpha_i. \] (4.3)

Substituting all of these items in Equation (2.2), and approximating the restart time as zero, gives us the overall wall clock time (cost function to be optimized).

\[ T_{w,i} = T_{s,i} + \frac{T_{s,i}}{\tau_i} \delta_i + \frac{1}{2} \tau_i T_{s,i} \alpha_i \] (4.4)

Next, we consider usage of SSD lifetime. For Job \( i \), each checkpoint consumes \( s_i \) gigabytes of SSD lifetime. Combined with the checkpoint interval \( \tau_i \), the total amount of SSD lifetime required for Job \( i \) is \( l_i \).

\[ l_i = \frac{T_{s,i}}{\tau_i} s_i \] (4.5)

We use the Young’s formula (2.1) as the baseline for optimum checkpoint interval.

\[ \tau_{i,\text{opt}} = \sqrt{\frac{2\delta_i}{\alpha_i}} \] (4.6)

Using it, we can compute the optimum SSD lifetime for a job. Allocating more SSD lifetime than the optimum gives no performance benefits to job or system. The optimum SSD lifetime \( l_{i,\text{opt}} \) is given by

\[ l_{i,\text{opt}} = \frac{T_{s,i}}{\tau_{i,\text{opt}}} s_i. \] (4.7)

Of course, SSD lifetime is a limited resource, so we constrain the total use by the \( M \) jobs to be the provisioned SSD lifetime, \( L \), as below.

\[ \sum_{i=1}^{M} l_i \leq L \] (4.8)

We define several characteristics of jobs and system.
• *Job Efficiency* is the useful work in wall clock time. It is defined as

\[
\text{Job Efficiency} = \frac{\text{Solve Time}}{\text{Wall Clock Time}} = \frac{T_{s,i}}{T_{w,i}}.
\] (4.9)

• *SSD Service Ratio* measures the fraction of useful SSD lifetime received, the fraction of SSD lifetime in the *optimum SSD lifetime*, given by (4.10).

\[
\text{SSD Service Ratio} = \frac{\text{Allocated SSD lifetime}}{\text{optimum SSD lifetime}} = \frac{l_i}{l_{i,\text{opt}}}.
\] (4.10)

• *System Efficiency* is the total useful work across all jobs as a fraction of total wall clock time, defined as

\[
\text{System Efficiency} = \frac{\text{Total Solve Time}}{\text{Total Wall Clock Time}} = \frac{\sum_{i=1}^{M} T_{s,i}}{\sum_{i=1}^{M} T_{w,i}}.
\] (4.11)

• *SSD Provisioning Ratio* is the fraction of available SSD lifetime in total *optimum SSD lifetime*.

\[
\text{SSD Provisioning Ratio} = \frac{L}{\sum_{i=1}^{M} l_{i,\text{opt}}}.
\] (4.12)

4.3 Allocation Problem

4.3.1 Allocation Objectives

We want to understand how to allocate the limited SSD lifetime to jobs in a workload on systems with a variety of different characteristics. We consider three allocation goals:

• **Size Based (SB) Allocation**

  A simple allocation method for job-size fairness. We allocate SSD lifetime in proportion to job size (number of nodes).
• **Job Efficiency Based (JEB) Allocation**

Allocate so as to ensure equal job efficiency across jobs. Users like this property. The allocation for each job depends on the overall workload composition.

• **System Efficiency Based (SEB) Allocation**

Allocate so as to maximize overall system efficiency. Supercomputer center administrators like this property. The allocation for each job depends on the overall workload composition.

SB can be achieved without complex optimization. We formulate JEB and SEB allocation computations mathematically in the next section.

### 4.3.2 Formulation and Solutions

We formulate three allocation objectives and derive corresponding solutions.

**A.** SB supplies SSD lifetime proportional to the job size. The definition simply gives

\[
l_i = \frac{n_i}{N} L.
\] (4.13)

**B.** JEB requires equalizing job efficiency for all jobs, as below (Equation 4.14)

\[
\frac{T_{s,1}}{T_{w,1}} = \ldots = \frac{T_{s,i}}{T_{w,i}}, \text{ for } i = 1..M
\]

subject to \( \sum_{i=1}^{M} l_i \leq L. \)

Substituting the items for \( T_{s,i} \) and \( T_{w,i} \) in (4.14), we have

\[
\begin{cases}
\frac{1}{\frac{\delta_1}{\tau_1} + \frac{1}{2} \tau_1 \alpha_1} = \ldots = \frac{1}{\frac{\delta_M}{\tau_M} + \frac{1}{2} \tau_M \alpha_M} \\
\frac{T_{s,1}}{\tau_1} s_1 + \frac{T_{s,2}}{\tau_2} s_2 + \ldots + \frac{T_{s,M}}{\tau_M} s_M = L
\end{cases}
\] (4.15)
Equation 4.15 characterizes the JEB allocation problem as a nonlinear system with variables \( \tau_i \). We used iterative search methods in the Matlab optimization toolbox to solve this system of equations for SSD allocation.

\( C \). SEB objective allocates SSD lifetime to each job in the workload such that the system efficiency is maximized.

\[
\begin{align*}
\max_{\tau_i} \text{ System Efficiency} &= \frac{\sum_{i=1}^{M} T_{s,i}}{\sum_{i=1}^{M} T_{w,i}} \\
\text{subject to } \sum_{i=1}^{M} l_i &\leq L
\end{align*}
\]

(4.16)

Since the solve time \( T_{s,i} \) are constants given by the workload, the maximization problem (4.16) is equivalent to the wall clock time minimization problem (4.17).

\[
\begin{align*}
\min_{\tau_i} \sum_{i=1}^{M} T_{w,i} \\
\text{subject to } \sum_{i=1}^{M} l_i &\leq L
\end{align*}
\]

(4.17)

The problem stated in (4.17) is a minimization problem with nonlinear constraints. Without constraints, there always exists an unique minimum solution to minimization of \( \sum_{i=1}^{M} T_{w,i} \), since \( \forall i, \frac{\partial^2 T_{w,i}}{\partial \tau_i^2} > 0 \). However with constraints, we cannot solve problem (4.17) directly or explicitly [43]. We use the Lagrange multiplier method to solve minimization problem (4.17), formulated as in Equation (4.18), where \( \mu \) is the corresponding Lagrange multiplier and \( S = \sum_{i=1}^{M} l_i \).

\[
\begin{align*}
\frac{\partial T_{w,i}}{\partial \tau_i} &= \mu \frac{\partial S}{\partial \tau_i} \text{, for } i \text{ in } 1..M \\
S &= L
\end{align*}
\]

(4.18)

Substituting the items for \( T_{w,i} \) and \( S \) in (4.18), we derive (4.19) and the form of checkpoint
interval (4.20).

\[- \frac{T_{s,i} \delta_i}{\tau_i^2} + \frac{1}{2} T_{s,i} \alpha_i = -\mu \frac{T_{s,i} s_i}{\tau_i^2} \]  

\[\tau_i = \sqrt{\frac{2(\delta_i - \mu s_i)}{\alpha_i}} \]  

(4.19)  

(4.20)

\(\tau_i\) is a function of \(\mu\), which can be obtained by solving the constraint equation (4.21).

\[\sum_{i=1}^{M} \frac{T_{s,i}}{2(\delta_i - \mu s_i)} s_i = L \]  

(4.21)

As a sanity check, note that the solution in (4.20) matches Young’s formula \(\tau = \sqrt{\frac{2\delta_i}{\alpha_i}}\) if the limited SSD lifetime constraints are eliminated. In Equation (4.20), the effect of SSD lifetime is reflected in term \(-\mu s_i\).

An explicit solution to Equation (4.21) should be given in the form \(\mu = \mu(T_{s,i}, \delta_i, s_i, \alpha_i, L)\). However, Equation (4.21) is nonlinear and contains a set of items. Thus we employ MATLAB’s equation solver [16] to obtain the numeric solutions of \(\mu\) and \(\tau_i\), given other parameter values. After obtaining \(\tau_i\), we can substitute \(\tau_i\) in equation (4.5) to get the allocation number \(l_i\) for SEB allocation objective.

### 4.4 Summary

In this chapter, we first present the assumptions of our model and then derive the analytical model. We discuss three allocation policies: size-based (SB) allocation, job-efficiency based (JEB) allocation and system-efficiency based (SEB) allocation, each designed from different perspective. We formulate these three allocation problem and provide solutions to them respectively.
CHAPTER 5
APPLYING THE MODEL

To apply the model, we first describe the computer system of interest, and then workload properties. We first apply the system-efficiency based allocation both to simple workloads and realistic workloads to understand the impact of job characteristics and mix on SEB allocation. Second, we explore properties and performance of SB, JEB, and SEB on various workloads. Finally, we explore appropriate SSD provisioning, how much is required to achieve acceptable system efficiency?

We use one-month IBM BG/P Intrepid [8] job traces containing 1,970 job records\(^1\), but that system has no SSD’s, so we use Gordon [7], one of the first supercomputer systems to incorporate SSD’s to size these resources.

5.1 System Model

The key system characteristics are nodes, node failure rate, memory size, SSD bandwidth, and SSD provisioning ratio. A small-scale simple system model is described for two-job workload study. We further define the range of each characteristic of real systems.

5.1.1 Simple System

We consider a small-scale system with medium failure rate for simple workloads study. The system has 1024 nodes. Projection of future failure rates is 500 to 20,000 FIT per node, therefore we choose medium failure rate of 10,000 FIT per node. Table 5.1 summarizes the value of each parameter.

\(^1\) The traces provide information of job size and solve time but not checkpoint size ratio and job interrupt rate.
Table 5.1: Simple System Configuration

<table>
<thead>
<tr>
<th>System Characteristic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num of Nodes</td>
<td>1024</td>
</tr>
<tr>
<td>Node Failure Rate (FIT)</td>
<td>10,000</td>
</tr>
<tr>
<td>Memory Size (GB per node)</td>
<td>8</td>
</tr>
<tr>
<td>SSD Bandwidth (GB/s)</td>
<td>32</td>
</tr>
<tr>
<td>SSD Provisioning Ratio</td>
<td>25%</td>
</tr>
</tbody>
</table>

5.1.2 Realistic System

We consider a hybrid model of Intrepid and Gordon. The range of system properties are defined as follows.

- **Nodes**: 40,960 (Intrepid).

- **Node Failure Rate**: 130-20,000 FIT. (Intrepid MTTI of 7.5 days (130 FIT) [54, 36] and Blue Waters MTTI of 6.11 hours (6,100 FIT), projections from 500 - 20,000 FIT per processor [54].)

- **Memory Size**: 2GB (Intrepid).

- **SSD Bandwidth**: 320GB/s (Gordon system [7] has 64TB memory and 256GB/s aggregate SSD bandwidth, projected to Intrepid based on memory size, Gordon-like SSD bandwidth is 320GB/s.)

- **SSD Provisioning Ratio**: 100%, 25%, and 6.25% (reducing 4x).

5.2 Workload Model

Important dimensions of jobs for supercomputers include job size, solve time, checkpoint size ratio and job interrupt rate. First we compose simple workloads consisting of two jobs to isolate the impact of job characteristics on allocation. Then we discuss how to synthesize realistic workloads.
Table 5.2: Simple Workloads

<table>
<thead>
<tr>
<th>Workload Name</th>
<th>Job Composition</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_Job_Size_WL</td>
<td>Two types of jobs which have different job size</td>
</tr>
<tr>
<td>S_Solve_Time_WL</td>
<td>Two types of jobs which have different solve time</td>
</tr>
<tr>
<td>S_Ckpt_Size_Ratio_WL</td>
<td>Two types of jobs which have different checkpoint size ratio</td>
</tr>
</tbody>
</table>

5.2.1 Simple Workloads

To isolate the impact of job characteristics on allocation, we compose simple workloads comprised of two jobs, which only differentiate in one job property. Three simple workloads are shown in Table 5.2.

5.2.2 Realistic Workloads

We discuss the range of realistic workloads properties as follows.

- **Job Size:** 512 - 16384 nodes.
- **Solve Time:** 1 to 17 hours.
- **Checkpoint Size Ratio:** 10% to 90% of node memory.
- **Job Interrupt Rate:** The job interrupt is determined by node failure rate, i.e. $\alpha_i = n_i \lambda$.

Having established the dynamic ranges for the job characteristics, we next discuss how to design workloads.

First, to isolate the impact each type of job characteristic, we design three workloads each comprised of jobs varying in just one characteristic (shown in Table 5.3). Next, we consider a mixed workload to understand their aggregate effects on realistic workloads. Finally, we design job size representative workloads – small job heavy, medium job heavy, and large job heavy.
Table 5.3: Trace-Based Realistic Workloads

<table>
<thead>
<tr>
<th>Workload Name</th>
<th>Key Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job Size WL</td>
<td>Vary job size from 512 to 16384</td>
</tr>
<tr>
<td>Solve Time WL</td>
<td>Vary solve time from 1 hour to 17 hours</td>
</tr>
<tr>
<td>Checkpoint Size Ratio WL</td>
<td>Vary checkpoint size ratio from 0.1 ∼ 0.9</td>
</tr>
<tr>
<td>Mixed WL</td>
<td>Vary job size from 512 to 8192, and checkpoint size ratio from 0.1 to 0.9</td>
</tr>
<tr>
<td>Small Job Heavy WL</td>
<td>Num of small jobs (⩽ 512 proc) is larger than 60% of WL</td>
</tr>
<tr>
<td>Medium Job Heavy WL</td>
<td>Num of medium jobs (1024-4096 proc) is larger than 60% of WL</td>
</tr>
<tr>
<td>Large Job Heavy WL</td>
<td>Num of large jobs (⩾ 8096 proc) larger than 60% of WL.</td>
</tr>
</tbody>
</table>

5.3 Metrics

We utilize the following three metrics to evaluate the resource allocation, job performance and system performance respectively (detail definitions are shown in Section 4).

- **SSD Service Ratio** quantifies the SSD lifetime allocation compared to optimum SSD lifetime indicated by Young’s formula.

- **Job Efficiency** represents the performance of a job. If jobs are allocated enough SSD lifetime to take checkpoints in interval suggested by Young’s formula, the job efficiency is maximized.

- **System Efficiency** is a typical metric for evaluating the performance of the system. High system efficiency indicates quick proceeding of jobs, which is desired by system administrators.
5.4 Studies

5.4.1 SEB Allocation on Simple workloads

To understand the affect of job characteristics (job size, solve time, checkpoint size ratio) on SEB allocation. We apply SEB Allocation on simple systems and simple workloads described above section. We show the comparison of SSD service ratio, job efficiency, and system efficiency of two jobs within each workload.

![Figure 5.1: SSD Service Ratio, Job Efficiency and System Efficiency vs. Job Size (SEB)](image1)

![Figure 5.2: SSD Service Ratio, Job Efficiency and System Efficiency vs. Solve Time (SEB)](image2)

![Figure 5.3: SSD Service Ratio, Job Efficiency and System Efficiency vs. Checkpoint Size Ratio (SEB)](image3)
Figure 5.1 presents the results of System Efficiency Based Allocation on workload "S_Job_Size_WL". We observe that given 25% SSD provisioning ratio, the SSD service ratio of Job 0 and Job 1 are identical as the ratio of their job size $n_1/n_0$ increases, both receiving 25% SSD service. In terms of job efficiency, Job 0 outperforms Job 1. The differences in job efficiency are exacerbated as job size ratio grows. This is because the job size increases both the possible number of failures and dump time. These two factors contribute to the performance degradation of Job 1 as job size grows. As a result, the system efficiency which is weighted average of job efficiency is decreasing correspondingly.

Figure 5.2 shows the results on workload "S_Solve_Time_WL". Comparing Job 2 (long running job) and Job 0 (short running job), we observe no difference in SSD service ratio, job efficiency, and system efficiency. The results conclude that the solve time has no impact on System Efficiency Based Allocation.

Figure 5.3 compares Job 3 (large checkpoint size job) and Job 0 (small checkpoint size job). It turns out that Job 3 is awarded same SSD service ratio as Job 0, which means the total number of checkpoints been taken are same for both applications. However larger checkpoint size of Job 3 increases dump time, and further reduce the job efficiency. The difference of job efficiency between Job 3 and Job 0 is increasing as the difference of their checkpoint size scales.

It can be concluded from simple workload results that (1) job size, solve time and checkpoint size ratio do not impact SSD service ratio under System Efficiency Based Allocation; (2) Large job size and checkpoint size ratio jobs suffer in job efficiency and further reduce the system efficiency.

5.4.2 SEB Allocation on Realistic Workloads

To understand how workload affects system behavior and therefore best system-efficiency based (SEB) allocation, we consider three workloads (Job Size WL, Solve Time WL, and Checkpoint Size Ratio WL described in Section 5.2 and summarized in Table 5.3).
Across a range of failure rates, workload characteristics and SSD provisioning levels, the SSD service ratio of SEB allocation is constant (see Figures 5.4a, 5.5a and 5.6a). That is, under SEB, SSD lifetime allocation is not affected by job size, solve time, and checkpoint size ratio.

The last three graphs in Figures 5.4, 5.5 and 5.6 explore the effect of job size, solve time and checkpoint size ratio on job efficiency as SSD provisioning ratio and node failure rate are varied. As job size increases, the job efficiency degrades because large jobs experience more failures, increasing rework time. However, as solve time is increased, there is no effect on job efficiency. Finally, as checkpoint size ratio increases, we see a smaller drop in job efficiency, due to increasing checkpoint dump time. Varying SSD lifetime (from 100% to 6.25% provisioning ratio) and failure rate (from 130 to 20,000 FIT) both increase the pressure on the checkpoint system, causing greater drops in job efficiency.

Overall, we make several observations for SEB allocation: (1) the resulting SSD service
ratios are independent of job size, solve time and checkpoint size ratio, and (2) jobs with larger size and checkpoint size ratio experience decreased job efficiency.
5.4.3 Comparing SB, JEB and SEB SSD Lifetime Allocation

To understand SB, JEB, and SEB allocation and its effects on jobs and systems we compare them on four workloads, using an extreme system configuration with 20,000 FIT per node (high) and 6.25% SSD provisioning ratio (low).

First, SB and SEB produce similar SSD service ratio as job size is varied (see Figure 5.7). JEB exhibits a preference for large jobs, giving them an even larger allocation of SSD lifetime than SB and SEB. Specifically, an increase from 6.25% to 9.25%, or 50% more lifetime is required to produce equal job efficiency, as shown to the right. In contrast, under SB and SEB allocation, large jobs suffer significantly lower efficiency (40% for jobs with 16K nodes compared to small jobs with 512 nodes).

Second, JEB and SEB produce the same SSD service ratios, i.e. 6.25%, independent of their solve time (see Figure 5.8). Because SB gives a fixed allocation based on size, short solve time jobs receive higher service ratios, and long solve time jobs receive lower service.
ratio. Consequently, under SB long solve time jobs suffer degraded job efficiency.

Third, we vary checkpoint size ratio, SB prefers small checkpoint size ratio jobs, but JEB prefers large. SEB is neutral (see Figure 5.9). As expected in each case, the preferred job types exhibit higher efficiency. Comparing Figure 5.7 and 5.9, checkpoint size ratio shows similar but smaller effect with SB and SEB. For both, large jobs have 6 to 10% lower job efficiency than small jobs.

Fourth, we vary both job size and checkpoint size ratio. SB produces the equal SSD service ratio for same checkpoint size ratio jobs as job size is varied. For a given job size, low checkpoint size ratio jobs have higher service ratio due to fixed allocation by SB. SEB produces same SSD service ratio independent of job size and checkpoint size ratio (see Figure 5.10). While, JEB prefers large jobs with higher checkpoint size ratio (i.e. 8k node,
checkpoint size ratio = 0.9), the service ratio of which is 3x of SB and 1.6x of SEB. As a result, JEB improves the job efficiency of large jobs by 24% compared to SB and 8% compared to SEB.

In Figure 5.11, we consider the system efficiency produced by SB, JEB, and SEB on these varied workloads. As expected, SEB always produces the best system efficiency. JEB can produce much lower system efficiency, with a dramatic 14% drop for the Job Size WL. SB produces as low as 5.5% lower, in the Mixed WL. Smaller differences are observed for the Solve Time WL and Checkpoint Size Ratio WL.

Overall, we observe that there are significant differences in achieved system efficiency and job efficiency as a function of the SSD lifetime allocation approach. With differences as
large as 5-14\%, careful choices are required.
5.4.4 Exploring SSD Provisioning

To understand how much SSD lifetime is worth buying, we consider the effect of SSD provisioning ratio on achievable system efficiency. In Figure 5.12, we plot the SSD Provisioning Ratio to achieve a given system efficiency (90%, 95% and 98%) as a function of the node failure rate. Each plot considers a different workload, ranging from small-job heavy to medium to large-job heavy. In all, for low failure rates, 10% SSD provisioning is sufficient to achieve 90% system efficiency. For workload with more large jobs, more SSD is needed to reach 95% system efficiency because failures affect the entire large jobs, more frequent checkpoints are needed. Thus, the SSD provisioning requirement grows. However, underprovisioning as low as 37% can achieve 95% system efficiency for the large job heavy workloads. In short, there’s a large benefit for a small amount of burst-buffer lifetime, but a high cost to reach extreme levels of efficiency.

Specifically, at high error rates (20K FIT/node), moving from 90% to 95% increases the required SSD lifetime by 2-2.5x. At low error rates close to those in systems today (2.5K FIT/node), the difference is similar, 2x to move from 90% to 95%. In both cases, even larger provisioning increases are needed to reach 98% system efficiency.

These results suggest that significant underprovisioning (10-20% of the optimum SSD lifetime) may be cost-effective for supercomputing systems.
5.5 Summary

In this chapter, we apply our model in practice. First we present the key system and workload parameters and discuss the range of values we explore. We deploy a Intrepid BG/P like system model and design workloads composed from Intrepid job traces. Second we discuss three metrics for evaluation, namely, SSD service ratio, job efficiency and system efficiency. Third, we conduct three studies.

- Apply SEB SSD lifetime allocation both on simple workloads and realistic workloads. We explore the effect of job characteristics on allocation.

- Apply SB, JEB and SEB allocation policies on varied workloads. We compare their properties and performance.

- Explore SSD provisioning ratio required to achieve 90%, 95%, and 98% system efficiency. Our results reveal that underprovisioning is desirable.
CHAPTER 6
DISCUSSION AND RELATED WORK

The problem of determining optimal checkpoint interval has been studied extensively [60, 31, 26, 47, 44, 34] with varied objectives and approaches. Young [60] first developed a first order model to estimate the optimum checkpoint interval that minimizes the time loss because of job failures due to random errors. Chandy [31] considered a model to place checkpoints such that the system availability is maximized. Daly [34] extended Young’s works and derived a higher order model with the minimization goal of total wall clock time. Aupy et al. [22] focused on the challenge of minimizing energy consumption and refined an analytical model to approximate the optimal checkpointing period for energy. Ling et al. [44] formulated the problem using the calculus approach to minimize the total expected cost. However, this body of research address the problem of optimizing the performance each job individually, without any consideration of shared resource constraints. Our work considers global optimization of the use of a shared resource – SSD lifetime – across jobs within workloads.

Considerable research explores resource-constrained optimization in cloud computing and high performance computing. Sarood [52] proposed a model to distribute available nodes and power amongst the queued jobs such that the throughput of HPC data centers is maximized under a given power budget. Numerous job scheduling algorithms [5, 59, 58, 56] have been proposed with various optimization objectives including system efficiency and throughput. These approaches assume that once scheduled, all jobs run at 100% efficiency. This is because there is no resource-sharing interaction. In contrast, the allocation of SSD lifetime affects the efficiency of each job; so the complex interaction amongst the jobs, their failure rates, and SSD lifetime is the distinct problem we explore. And, as shown by our results, this is a significantly different problem.

Other cloud research attempts to minimize “application cost”. Di et al. [35] developed a virtual machine (VM) resource allocation algorithm for cloud systems that minimizes user payment. Chaisiri et al. [30] considered VM provisioning in Amazon Elastic Compute
Cloud (EC2) system to minimize user cost. Such problems are fundamentally different from considerations of job and system efficiency which are the focus of our work. For example, our work explores how much SSD lifetime is worth buying for resilience – from a system operator’s point of view.

Our work focuses on a different problem, both demonstrating the importance of SSD lifetime allocation for job and system efficiency, as well as suggesting effective techniques for allocation. We go one step further, showing that only 10-20% of what many would expect is the proper SSD lifetime is needed to achieve 90% system efficiency.

To apply it, based on an expected workload, the model can be used to pre-compute SSD lifetime allocations based on job mix properties. A more ambitious approach might apply the model periodically to the system history or even dynamically.
CHAPTER 7
SUMMARY AND FUTURE WORK

7.1 Summary

As supercomputers are growing towards exascale, the MTBF is expected to be in the order of minutes. Checkpoint/restart approach based on burst buffers (SSD) is proved to meet resiliency need in supercomputers. However, because of the relative high cost and limited lifetime of SSD, it is important to understand how to efficiently utilize SSD and how to make provision for supercomputers.

In this thesis, we explore two problems. First is to understand how to allocate SSD lifetime among jobs in a workload such that their efficiency is maximized with failures taken into consideration. Second is to explore how to provision supercomputer systems given their failure rate with objectives to guarantee resilience and maximize system efficiency.

we derive a model that captures system and job characteristics, and use it to formulate the SSD lifetime allocation problem. Exploring three objectives – size based, job efficiency based and system efficiency based allocation, we show that a central lifetime constraint changes checkpoint intervals, and thereby the achievable job and system efficiency by significant percentages. These results suggest the careful management of SSD lifetime in burst buffers is important. Study of SSD provisioning reveals that low provisioning is sufficient to achieve 90% and 95% system efficiency, so low provisioning may be more cost-effective.

Our results give insight into appropriate policies for SSD usage for resilience in future supercomputers that include burst buffers, and into the cost-effective approach to provisioning burst buffers on such systems.
7.2 Future Work

This study used a relatively restrictive model to explore SSD lifetime allocation problem and provisioning problem. To simplify the model, we made several assumptions that are not accurate in practice. There are considerably number of future directions.

- First we think it is interesting to study a broader variety of workloads and system parameters. In this thesis we considered four job and five system characteristics and explored certain range of several characteristics. For model evaluation we only considered a hybrid model of Intrepid and Gordon system. However, future systems will change rapidly in terms of number of nodes and SSD bandwidth. We can project a possible range of these parameters and take them into account to apply the model. For example, we can learn the impact of changing SSD bandwidth on allocation. Such experiments can lead to more insightful conclusions for future systems.

- An possible extension of the model is to capture burst buffer bandwidth contention. In this study, we did not consider the possible contention of bandwidth. However, it is likely the bursty I/O of jobs occur concurrently. In such a case, the checkpointing time of jobs may vary and further affect the allocation.

- In this work we explored the isolated affect of each job characteristic on allocation. We feel it is interesting to study simultaneous variation of system and workload parameters. For example, when SSD bandwidth scales up and failures rate increase, how does SEB allocation change for a given workload.

- Finally we consider the implementation of the model for practical usage. One possibility is to implement the allocation policy in job scheduler. For the jobs in waiting queue, the job scheduler collects job mix properties and utilizes the model to decide the SSD lifetime allocation. Once the allocation is given, the checkpoint interval of each job stay constant. More ambitiously, the allocation model can be used online, that
is, when a new job arrives, the checkpointing intervals of all current jobs are updated dynamically.
REFERENCES


