8-2016

Achieving High Reliability and Efficiency in Maintaining Large-Scale Storage Systems through Optimal Resource Provisioning and Data Placement

Lipeng Wan
University of Tennessee, Knoxville, lwan1@vols.utk.edu

To the Graduate Council:

I am submitting herewith a dissertation written by Lipeng Wan entitled "Achieving High Reliability and Efficiency in Maintaining Large-Scale Storage Systems through Optimal Resource Provisioning and Data Placement." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Qing Cao, Major Professor

We have read this dissertation and recommend its acceptance:

Feiyi Wang, Michael W. Berry, Asad J. Khattak

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Achieving High Reliability and Efficiency in Maintaining Large-Scale Storage Systems through Optimal Resource Provisioning and Data Placement

A Dissertation Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Lipeng Wan

August 2016

This work is dedicated to my parents

Acknowledgements

First of all, I would like to thank Dr. Qing Cao, for serving as my major advisor during my Ph.D. studies. Over the past five years, he continuously provided me support and guidance for my research. Whenever I encountered a difficulty in research, he always encouraged me to face the challenge and helped me understand what would be the best way to solve the problem with great patience. Moreover, Dr. Cao gave me lots of flexibility in choosing my research topic, and I was allowed to work on any research projects I was interested in. With such flexibility, I was able to explore more potential research directions and find the one that I really love to devote myself to.

Second, I would like to thank Dr. Feiyi Wang, who was my mentor when I worked as an intern at the Oak Ridge National Laboratory. During my internship at ORNL, Dr. Wang treated me like a colleague rather than a student. He always encouraged me to come up with my own idea and be confident to talk about it in front of other people. With such encouragement, I eventually became an independent thinker and a qualified researcher in computer science.

Third, I would like to thank Dr. Michael W. Berry, and Dr. Asad J. Khattak for serving on my committee. I really appreciate the insightful comments and suggestions they gave me for helping me revise and refine my dissertation.

Finally, I would like to thank all my colleagues at the University of Tennessee, Knoxville and the Oak Ridge National Laboratory. It was a great pleasure and honor for me to have had the opportunity to work with you.

All of science is nothing more than the refinement of everyday thinking.

— Albert Einstein

Abstract

With the explosive increase in the amount of data being generated by various applications, large-scale distributed and parallel storage systems have become common data storage solutions and been widely deployed and utilized in both industry and academia. While these high performance storage systems significantly accelerate the data storage and retrieval, they also bring some critical issues in system maintenance and management. In this dissertation, I propose three methodologies to address three of these critical issues.

First, I develop an optimal resource management and spare provisioning model to minimize the impact brought by component failures and ensure a highly operational experience in maintaining large-scale storage systems. Second, in order to cost-effectively integrate solid-state drives (SSD) into large-scale storage systems, I design a holistic algorithm which can adaptively predict the popularity of data objects by leveraging the temporal locality in their access patterns and adjust their placement among solid-state drives and regular hard disk drives so that the data access throughput as well as the storage space efficiency of the large-scale heterogeneous storage systems can be improved. Finally, I propose a new checkpoint placement optimization model which can maximize the computation efficiency of large-scale scientific applications while guarantee the endurance requirements of the SSD-based burst buffer in high performance hierarchical storage systems. All these models and algorithms are validated through extensive evaluation using data collected from deployed large-scale storage systems and the evaluation results demonstrate our models and algorithms can significantly improve the reliability and efficiency of large-scale distributed and parallel storage systems.

Table of Contents

Chapter 1

Introduction

With the explosive increase in the amount of data being generated by various applications, large-scale distributed and parallel storage systems have become common data storage solutions and been widely deployed and utilized in both industry and academia. Some examples of such storage systems include Google File System Ghemawat et al. (2003a), Facebook's Haystack object-based storage system Beaver et al. (2010), the Oak Ridge Leadership Computing Facility's (OLCF) Spider I and II storage systems Shipman et al. (2009); Oral et al. (2013), Livermore Computing Center's Sequoia storage system Behlendorf (2012) and Riken Advanced Institute for Computational Science's K-Computer storage system Sakai et al. (2012), etc. While these high performance storage systems significantly accelerate the data storage and retrieval, they also bring some critical issues in system maintenance and management.

First, in order to achieve large capacity and high I/O performance, these large-scale storage systems are usually composed of tens of thousands of physical devices, such as hard disk drives (HDDs), controllers, switches, I/O servers, etc. However, the increase of physical components also leads to high vulnerability, as the failure of any of these components might make partial or entire storage system be out of service. Even worse is that some important data might get lost due to the system failures. Therefore, the reliability issue of large-scale storage systems is always the most critical concern of both system administrators and users. Since the device failures are inevitable, we need to design effective resource management and spare provisioning strategies to minimize the impact brought by failures.

Second, with the development of storage technologies, flash-based storage devices, such as solid-state drives (SSDs), have eventually been exploited by large-scale storage systems SDSC (2015). Though these storage devices can provide much higher I/O performance, they are also much more expensive than traditional hard disk drives of the same capacity. In fact, it is not yet practical to use SSDs to completely replace conventional hard disks drives in large-scale storage systems, which also means only partial of the data can be stored on SSDs. Therefore, how to cost-effectively integrate SSDs into large-scale storage systems and design efficient data placement and replication mechanisms in such heterogeneous storage environments to improve the data access throughput as well as the storage space efficiency is also a challenging task.

Moreover, in some scenarios, the write workloads issued to large-scale storage systems can be extremely intensive. For instance, the large-scale storage systems used in high performance computing (HPC) environments often need to store checkpoint data generated by scientific applications running on supercomputers. Due to the high bandwidth required by such checkpointing operations, SSDs are often used as the burst buffers to absorb the checkpoint workloads. However, the amount of data that can be written to SSDs is limited since only a finite number of program-erase (P/E) cycles are possible before the bit error of SSD becomes unacceptable high. Such intensive write workloads generated by high-frequency checkpointing operations of scientific applications will dramatically consume the allowed P/E cycles of each SSD and wear out the devices quickly. Therefore, how to efficiently utilize SSDs while guarantee their endurance requirements under intensive write workloads is another common issue needs to be solved in large-scale storage systems, especially in HPC storage environments.

In fact, the above issues cover three critical aspects of maintaining large-scale storage systems, including how to fulfill system construction and provisioning, how to improve system performance and how to prolong system lifespan. In this dissertation, I plan to address these issues through the following methodologies:

- **Optimal resource management and spare provisioning:** Designing and Building a large-scale storage system needs to manage all kinds of hardware resources by factoring in a variety of goals such as capacity, performance and availability, while adhering to a fixed price point. In my research, I propose a two-phased design approach, namely initial and continuous provisioning that can help alleviate this situation. Initial provisioning addresses the early stage of the procurement, and explores the optimal tradeoffs between cost, performance and availability. Continuous provisioning provides an optimal spare part provisioning model to ensure a highly available system operational experience. Both approaches leverage the insights gained from a detailed analysis of field-collected operational data (including device replacement logs, vendor-provided failure rates, etc.) and a well-designed system-agnostic simulation tool.

- **Optimal workload-adaptive data placement:** Existing solutions to build a heterogeneous storage system utilizing both HDDs and SSDs are largely based on heuristic algorithms that are either developed in isolation with the runtime workload, or are based on static assumptions on the workload patterns, making them unsuitable when the underlying workloads and demands change over time. In my research, I propose a holistic optimization algorithm which can adaptively predict the popularity of data objects by leveraging temporal locality in their access pattern, and adjust their placement and replication among storage tiers to improve the data access throughput and the storage space efficiency.

- **Optimal checkpoint placement with guaranteed burst buffer endurance:** In order to provide large-scale scientific applications running on HPC systems enough bandwidth and IOPS to write checkpoints, SSDs are often used to build a burst buffer layer between the compute nodes and parallel file systems so that the checkpoint data can be temporarily written into the burst buffer first and then drained to the underlying parallel file systems asynchronously. In existing large-scale storage systems, the burst buffers were designed to absorb all I/O workloads. However, in reality the intensive write workloads generated by large-scale long-running scientific applications through checkpointing could degrade the endurance of SSD devices and the reliability of the burst buffer significantly. In my research, I propose a new checkpoint placement optimization model which can maximize the computation efficiency of large-scale scientific applications while guarantee the endurance requirements of the SSD-based burst buffer in HPC storage environments.

Chapter 2

Related Work

According to the three methodologies I proposed in this dissertation, I summarize the related work into the following three aspects:

- **Reliability and spare provisioning of storage systems:** Storage system reliability and data availability have been studied on several fronts. Analytical modeling, coupled with field data analysis and fitting are among the most common approaches. A large body of existing work focusses on building probability models for failures of disk drives and data loss in RAID groups Gibson and Patterson (1993); Chen et al. (1994); Schulze et al. (1989); Patterson et al. (1988); Xin et al. (2003); Rao et al. (2006). A few existing studies have also tried to estimate the reliability of a storage system through simulation Greenan (2009); Elerath and Pecht (2007); Elerath and Schindler (2014). In particular, Elerath and Pecht Elerath and Pecht (2007) have implemented a Monte Carlo simulation for RAID 4 groups to evaluate how time dependent failure and repair rates impact the average number of data loss events that could occur during a given mission time. Greenan developed a high-fidelity reliability simulator for erasure-coded storage systems Greenan (2009). All of the simulation-based approaches focus at the component-level, i.e., disk or RAID group failures. None of them takes an end-to-end approach to study reliability and its impact on the provisioning of a large-scale storage system.

Spare provisioning optimization has been extensively studied in the industrial engineering area. Different optimization models have been proposed by a number of researchers. For instance, in order to guarantee a specific availability metric for the system, queuing theory based approaches have been frequently used to determine the number of spare parts that should be prepared Jardine and Tsang (2005); Mani and Sarma (1984); Alam and Mani (1988); Lewis and Cochran (1995). Besides queuing theory, some optimization-based models Vaughan (2005); Ghodrati et al. (2012) were also proposed in the operations research (OR) area. However, due to the complexity of the extreme-scale distributed storage system, existing OR related spare provisioning models cannot be directly applied or non-trivially extended.

- **Data placement optimization in heterogeneous storage systems:** As large-scale distributed storage systems have been widely used in both industry and academia, the problem of distributing several petabytes of data among hundreds or thousands of storage devices becomes more and more critical. To address this problem, many data placement algorithms have been proposed. For instance, Distributed Hash Tables (DHTs) have been used to place and locate the data objects in P2P systems Stoica et al. (2001); Ratnasamy et al. (2001); Cai et al. (2004). Another replica placement scheme called chain placement was also proposed and applied to some P2P and LAN storage systems Rowstron and Druschel (2001); Lee and Thekkath (1996); MacCormick et al. (2004). Honicky and Miller presented a family of algorithms named RUSH Honicky and Miller (2004) that utilizes a mapping function to evenly map replicated objects to a scalable collection of storage devices, so that it can support efficient additions and removals of weighted devices.

To address the reliability and replication issues of the RUSH algorithm, Weil et al. proposed a scalable pseudo-random data distribution algorithm named CRUSH Weil et al. (2006b). Besides optimally distributing data to available resources and efficiently reorganizing data after adding or removing storage devices, CRUSH exploits flexible constraints on replica placement to maximize data safety in the case of hardware failures. Specifically, CRUSH allows the administrator to assign different weights to storage devices so that the administrator can control the relative share of data each device is responsible for. However, the device weights used in the CRUSH algorithm only reflect the capacities of storage devices, therefore, the CRUSH algorithm may not be effective anymore for hybrid storage systems consisting of both SSD and HDD devices, as these two kinds of storage devices have completely different performance characteristics.

Recently, efforts have been made to combine SSD and HDD drives together to construct hybrid storage systems. In such systems, SSDs are either used for caching purposes, or used as more independent storage devices. For example, Srinivasan et al. designed a block-level cache named Flashcache Srinivasan et al. (2015) between DRAM and hard disks using SSD devices. Zhang et al. proposed iTransformer Zhang et al. (2012) which exploits a small SSD to schedule requests for the data on disks so that high disk efficiency can be achieved. SieveStore Pritchett and Thottethodi (2010) adopts a selective caching approach in which the accesses of each block are tracked and the most popular block is cached in SSD device. In the second approach, SSDs are more independently used. Chen et al. designed and implemented a high performance hybrid storage system named Hystor Chen et al. (2011a), which identifies data blocks that either can result in long latencies or are semantically critical on hard disks, and store them in SSDs for future accesses. In order to prolong the service life of SSDs devices, Ren et al. proposed I_CASH Yang and Ren (2011) to reduce random write traffic to SSDs. Specifically, I_CASH is an approach that exploits the spacial locality of data accesses, and only store those seldom-changed data blocks on SSDs. Finally, ComboDrive Payer et al. (2009) concatenates SSD and HDD into one address space via a hardware-based solution, so that certain data on HDD can be moved into the faster SSD space.

There are two major drawbacks in existing studies: first, most existing studies on hybrid storage systems only consider how to improve the utilization of SSD drives, but they have ignored the reliability and replication issues; second, existing studies have not considered the dynamic nature of the I/O workloads, a nature that makes continuous training and learning necessary.

- **Checkpoint placement and SSD lifetime optimization:** The idea of utilizing SSD devices to build a burst buffer layer between HPC systems and parallel file systems to temporarily absorb checkpoint I/O workloads has been researched over the last few yearsLiu et al. (2012); Sato et al. (2012); Bent et al. (2012). In fact, several recent high performance computing system deployments have already included burst buffer layers Xu et al. (2014); Sato et al. (2014). Existing studies in this area mainly focus on how to maximize the I/O performance of the checkpointing operations through the burst buffer. For example, Scalable Checkpoint/Restart library (SCR) Moody et al. (2010) provides an interface that allows scientific applications to periodically do checkpointing to SSDs, and asynchronously flush these checkpoints from SSDs to the underlying parallel file systems without interfering with applications' computation phase. In Wang et al. (2014), a new design of the burst buffer system named BurstMem is proposed which implements functionalities similar to SCR but provides better I/O performance through efficient storage and communication management strategies. Park and Shen (2009a) presents a trace-driven performance evaluation of scientific I/O workloads on SSDs, which shows the concurrent I/O might significantly affect the SSD performance. However, none of these studies considered the endurance issue of the burst buffer under scientific I/O workloads, though such issue has emerged and will become extremely critical in next-generation HPC computing and storage systems.

For single SSD device under common I/O workload, its endurance and reliability have been extensively studied. The existing body of work in this area can be classified into three categories. The first category focuses on improving the internal design of SSD devices Agrawal et al. (2008); Chen et al. (2011b); Wu and He (2012), including designing better flash translation layers (FTL), more efficient wear-leveling and garbage collection algorithms. The second category mainly concentrates on OS-level optimizations Wu et al. (2009); Lu et al. (2013), including utilizing TRIM commands from OS, designing filesystem-aware garbage collection algorithms. Finally, the third category reshapes the I/O workloads Soundararajan et al. (2010); Chen et al. (2011a); Yang and Ren (2011), including reducing the write workloads and the randomness of the access pattern, so that the write-amplification can be reduced. All of these techniques are effective if the amount of data written to SSDs is within some boundaries. However, for I/O workloads generated by large-scale long-running scientific applications, these techniques might not be efficient, given the checkpoint frequency and amount of data written at each checkpoint step, consuming the allowed program-erase (P/E) cycles of underlying burst buffer SSDs and quickly wearing them out.

In order to better utilize SSD devices under scientific I/O workloads, Fang and Chien (2015) presents a checkpoint interval optimization model for large-scale scientific applications. This model is essentially same as those developed by Young (1974); Vaidya (1997); Daly (2006) whose objective is to maximize the computational efficiency of HPC systems, but it also puts the constraint of burst buffer capacity into consideration. By using this model, optimal SSD capacity allocation among all scientific applications can be determined. Since the SSD-based burst buffers of the HPC systems are assumed to absorb all checkpoint data of the scientific applications, this model intends to reduce the checkpoint frequency of some write-heavy jobs if a rigorous constraint of burst buffer capacity is given. However, such reduction in checkpoint frequency also significantly increases the potential wasted computation time caused by system failures, especially for large computation jobs.

Chapter 3

Optimal Resource Management and Spare Provisioning

3.1 The Overview

The design and procurement of large-scale storage systems are complex in nature. When faced with multi-faceted considerations, system designers usually cope with the challenges by adopting an ad hoc process that is a combination of back of the envelope calculations and the reliance on past experiences. The end result may *make sense*, but they are difficult to reason, with little or no quantifiable justification. Therefore, we take a more systematic approach by focusing on three key issues in designing such a system, namely availability, capability (performance) and capacity, under a fixed cost constraint. In particular, we divide the provisioning process into two phases, namely **initial provisioning** and **continuous provisioning**. The operational experience from system administrators suggests that designing for the second phase should receive equal, if not more attention since the shelf life of an extreme-scale storage system tends to be five years or even longer. As can be seen in Table 3.1, the two phases also place different emphasis regarding the aforementioned key characteristics

(Check marks indicate the metrics that will be optimized in each phase, while "fixed" indicates the constraints during each phase).

Table 3.1: Provisioning approaches

	Performance	Capacity	Availability	Spare Parts
Initial Provisioning	✓	✓	✓	Fixed
Continuous Provisioning	Fixed	Fixed	✓	✓

The provisioning of the initial system deployment is primarily based on the understanding of the trade-offs among cost, performance and capacity. While cost remains the primary constraint, it is not the case that simply buying faster disks will yield the best performance for a given budget. This is because of the complex building structures of an HPC storage system, as well as how different components affect the performance, cost and reliability of the whole system.

Since the component characteristics in storage systems change over time, achieving high data availability requires continuous provisioning and deployment. For instance, when an extreme-scale storage system is initially deployed, all components are new, but as time goes by, some components fail and get replaced, which changes their performance or reliability characteristics. If spare parts have been provisioned, and readily available before the failure, the replacement and repair of these faulty parts could be completed quickly, significantly reducing the possibility of data unavailability. Moreover, as failed components are replaced by spare parts, the system contains both new and aging components. Thus, the reliability status of the system during operations is different from the one at the time of initial deployment. Therefore, the spare provisioning policies for continuous operations should also be different.

In this chapter, I concentrate on these resource provisioning problems during the construction and maintenance of large-scale storage systems. My study are primarily intended for storage system architects, administrators and procurement teams, and can help them answer the following critical questions: How many units should be purchased for each type of hardware component in order to achieve the desired capacity and performance under a fixed price envelope? What kind of disk drive should be purchased? What is the impact on cost, performance, and footprint, assuming a smaller sized drive can achieve the desired capacity? How will the drive size impact availability and rebuild times of the RAID group, and consequently, the performance of the RAID array? What are the expected failure rates of the storage subsystem components, and how do they impact data availability? How to take advantage of the above fact to better provision spares to improve data availability, rather than blindly allocating funds? The answers to these questions will be used to make better procurement plans and provisioning policies.

3.2 Factors Affecting the Reliability of Large-Scale Storage Systems

In order to design optimal initial and continuous resource management and provisioning policies, we need to have a comprehensive understanding on system architectures, device failures, failure dependencies and propagation. These are the factors that have most significant impact on reliability of large-scale storage systems.

3.2.1 System architectures

System architecture plays an important role in the reliability of large-scale storage systems. For the convenience of replacing failed components in the system, hardware devices are usually encapsulated as *field replaceable unit* (FRU). A set of FRUs that implement a particular functionality will be further integrated as an *scalable system unit* (SSU). Large-scale storage systems are built using SSU for ease of design, procurement, deployment, management and maintenance. An SSU consists of all required components to build a stand alone file system. In order to reach the design targets, multiple SSUs are acquired and deployed. SSU examples include block-level storage systems (e.g. DDN SFA series DataDirect Networks, Inc. (2014), IBM DS series IBM DS8000 Series (2014), NetApp FAS series NetApp, Inc. (2014)) or file-system level appliances (e.g. Seagate ClusterStor9000 Seagate Technology (2014) or Panasas ActiveStor Panasas, Inc. (2014)).

The architecture of OLCF's Spider I is presented here as an example of a large-scale storage system. The design targets and specifications of Spider I are well documented Shipman et al. (2009). Spider I was deployed in 2008, and remained operational until 2013, serving the Jaguar supercomputer which was No. 1 on the Top500 list of machines in June 2010. At the time of deployment, Spider I was announced as the fastest and largest known Lustre parallel file system in the world. Spider I offered an aggregate system performance of 240 GB/s, and provided over 10 PB of RAID 6 formatted capacity, using 13,440 SATA disks and 192 file system servers. It served more than 26,000 file system clients from several clusters and the Jaguar supercomputer.

Spider I was built using 48 SSUs, each one consisted of a DDN S2A9900 controller couplet DataDirect Networks, Inc. (2011), with 280 1 TB SATA disks configured in 5 disk enclosures. Each Spider I DDN couplet was composed of two singlets and connected to 4 file system servers. Host-side interfaces in each singlet was populated with two dual-port 4x DDR IB HCAs. The back-end disks were connected via ten SAS links on each singlet. For a SATA based system, these SAS links connected to expander modules within each disk shelf. The expanders then connected to SAS-to-SATA adapters on each drive. All components had redundant paths. Each singlet and disk tray had dual power-supplies where one power supply was powered by the house power and the other by the UPS. Figure 3.1 illustrates the internal architecture of a Spider I DDN S2A9900 couplet.

In the following sections, Spider I is used as a case study since its field failure data is publicly available Wan et al. (2014b, 2015).

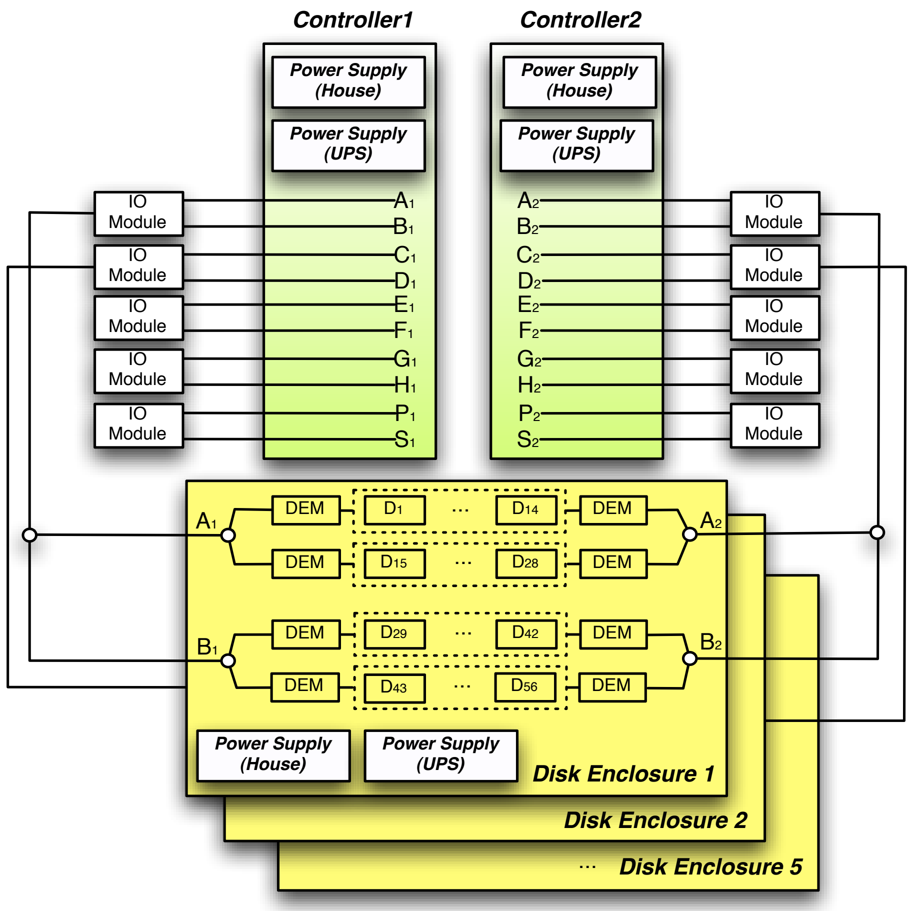


Figure 3.1: Spider I S2A 9900 architecture

3.2.2 Device failures

Different types of FRUs have different failure patterns, some devices might fail more often than others during the operations. Therefore it is critical to understand the reliability characteristics of each type of FRU. Usually, such information can be obtained from vendor-provided reliability metrics and field-collected failure data.

Vendor-provided reliability metrics

System vendors often provide AFR (annualized failure rate) or MTTF (mean time to failure) of each type of FRU. As stated earlier, Spider I consists of 48 SSUs in total, and each SSU is built with 9 types of FRUs. The vendor-provided AFRs of all types of

Table 3.2: FRUs in one scalable storage unit

FRU Type	Number per SSU	Total Number	Unit Cost ($)	Vendor AFR	Actual AFR
Controller	2	96	10,000	4.64%	16.25%
House Power Supply (Controller)	2	96	2,000	0.83%	4.38%
Disk Enclosure	5	240	15,000	0.23%	1.17%
House Power Supply (Disk Enclosure)	5	240	2,000	0.08%	8.50%
UPS Power Supply*	7	336	1,000	3.85%	NA
I/O Module	10	480	1,500	0.38%	0.92%
Disk Expansion Module (DEM)	40	1,920	500	0.23%	0.29%
Baseboard*	20	960	800	0.23%	NA
Disk Drive	280	13,440	100	0.88%	0.39%

*Field data missing, actual AFR is unavailable.

FRUs are listed in Table 3.2. Vendor-provided reliability metrics can be used to derive a coarse-grained estimation of a storage subsystem's reliability. As an example, one model that has been widely used to estimate the data availability of disk redundancy groups is continuous Markov chain, which has an underlying assumption that the failure rates of disk drives are constant (time independent) Gibson and Patterson (1993); Chen et al. (1994); Schulze et al. (1989); Patterson et al. (1988). With such a model, the vendor-provided metrics, AFRs and MTTF, can be used to establish the failure model of each disk drive, which assumes that the time to failure of disk drives is an exponential distribution.

Field-collected failure data

Besides the vendor-provided metrics, system administrators typically maintain field-gathered failure and replacement data. Such information is much closer to the reality than vendor provided reliability metrics. In fact, by analyzing the field-gathered failure data of storage systems, several existing studies have shown that the failure rates of disk drives and other hardware components can vary over time Greenan (2009); Pinheiro et al. (2007).

The failure and replacement data for Spider I was collected from all of the 48 SSUs during its 5-year operational period. The dataset contains timestamps when device replacement was needed. We first calculate the actual AFR for each type of FRU using the following formula and the calculation results are also listed in Table 3.2.

$$\text{A type of FRU's actual AFR} = \frac{\text{Total \# of replacement of such FRU}}{\text{Total \# of such FRU} \times \text{Years}} \times 100\% \quad (3.1)$$

The key findings from the actual AFR calculation can be summarized as follows:

- *The actual annualized failure rate (AFR) of Spider I disks is only 0.39% – much smaller than what has been reported in previous studies Schroeder and Gibson (2007). It is hard to generalize this as the environment, testing conditions and vendors are quite different. Efficient facilities support, e.g., better power and cooling infrastructure, might be a factor here. However, it is not possible to quantitatively establish a causal relationship between operating conditions and disk drive failure rate.*

- *Aggressive burn-out tests at the time of system deployment help eliminate potential problematic or slower disks early on, which improves the overall aggregate parallel performance. It also keeps the disk AFR low by removing potential problematic disks from the population. There is no community standard for stress testing and slow disk identification. The method adopted by OLCF involved individually stressing each SSU, and identifying the slowest disk RAID groups. Then, those groups were exercised separately, and latency statistics on the disks were collected individually. This process should be performed during initial deployment, and repeated periodically to keep a healthy and uniformly performing disk population. OLCF's records indicate that the AFR before the acceptance of the Spider I system was much higher (2.2%).*

Their early testing helped remove close to 200 slow or bad disks. This resulted in a much lower AFR during production (0.39%).

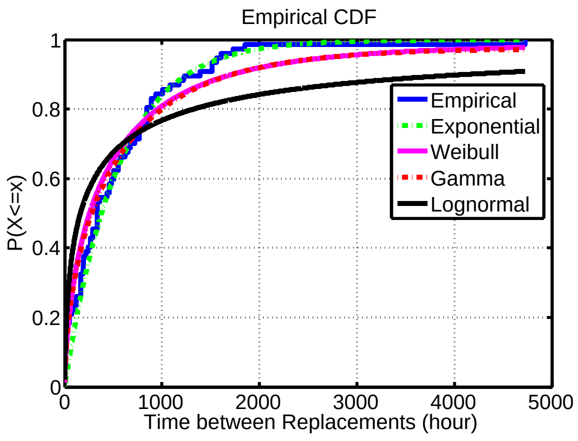
- *Non-disk components of Spider I have higher AFRs than vendor provided metrics. While this comes as a surprise, it also suggests that future studies should carefully model and account for the reliability of non-disk components as they contribute heavily towards the overall reliability of the system.*

With the failure dataset, we also derive the empirical, cumulative distribution function (CDF) of the time between device replacements for different types of FRUs (Figure 3.2). An interesting fact that needs to note is the failures of disk drives can be more accurately modeled by joining two different distributions. For example, as shown in Figure 3.2(d), when the time between disk replacements is relatively small, a Weibull distribution with decreasing failure rate is a better fit; with increasing time between disk replacements, the failure rate is stable, and an exponential distribution is a better fit. This observation indicates that in reality the failure rate of disk drives could be neither constant nor monotonically increasing or decreasing, which differs from what is usually assumed by many existing studies Schwarz et al. (2004); Elerath and Pecht (2007); Greenan (2009); Elerath and Schindler (2014).

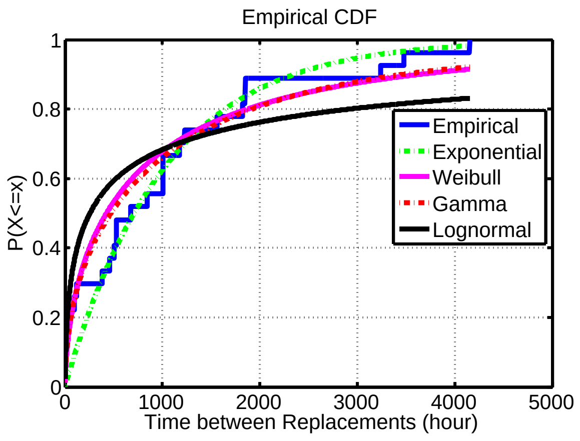
Time spent on FRU replacement in Spider I have not been recorded or shared with the public. However, it was stated that most of these replacements were completed within 24 hours, if spare parts were available on-site. If there was no spare part on-site, a replacement was awaited, and usually took at least 7 days Hill (2014).

3.2.3 Failure dependencies and propagation

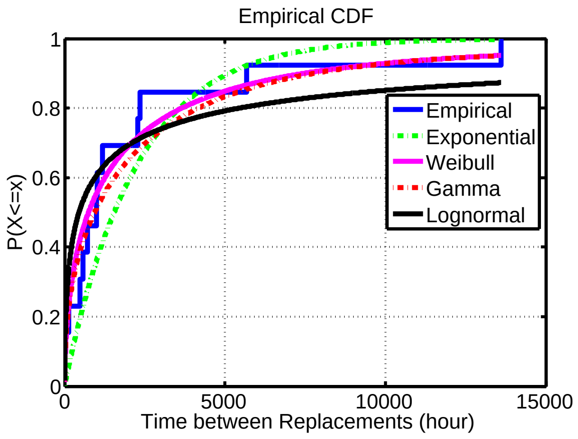
A large-scale storage system is often composed of thousands of FRUs, and the failure dependencies between them are complex. Specifically, one FRU's failure might have a cascading effect on other FRUs, as there is often a correlation between the failures of closely-coupled hardware components in a storage system. For example, a disk enclosure's failure might lead to the unavailability of hundreds of disk drives.



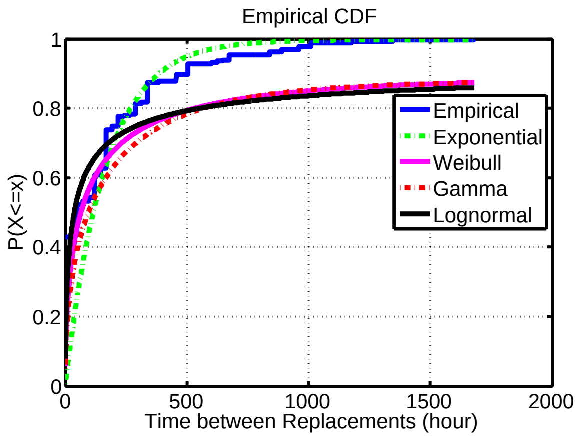
(a) Controllers



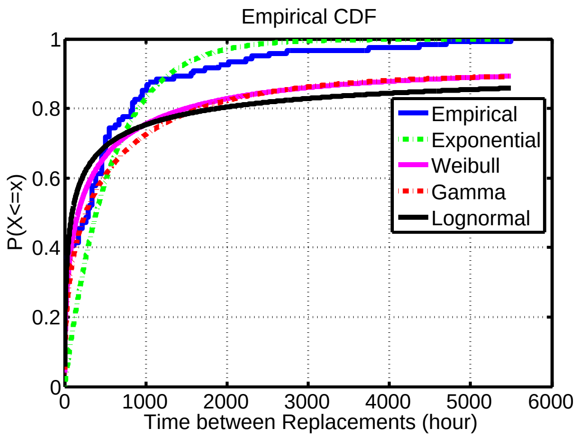
(b) Disk Expansion Modules



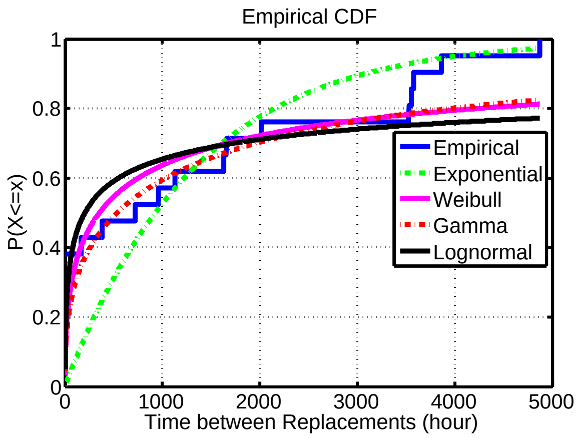
(c) Disk Enclosures



(d) Disk Drives



(e) House Power Supply



(f) I/O Modules

Figure 3.2: Distribution of time between device replacements for different types of FRUs in Spider I

Therefore, the failure dependencies between FRUs, or more specifically, how the failures propagate in the system, is another important factor needs to be considered in maintaining large-scale storage systems.

Based on the understanding of the architecture of large-scale storage systems and failure data obtained from the field and vendors for Spider I, we built a generic simulation tool to study how failure dependencies and propagation affect the reliability of the entire storage system. This tool will also be further used to evaluate different provisioning policies in next two sections.

Design considerations

The design of the simulation tool was inspired by a conventional diagrammatic method for modeling the reliability of complex systems called the reliability block diagram (RBD) Rausand and Hoyland (2003). It can estimate not only the number of failures during a certain period of operation for each type of FRU, but also the system-level reliability by analyzing the failure propagation.

As shown in Figure 3.3, in phase 1, the failure events of each type of FRU are randomly generated based on the reliability characteristics, which are determined by vendor-provided metrics, historical failure data and the provisioning policies used. Thereafter, the failure events are randomly allocated to FRUs that belong to the same type, and logged throughout each FRU's life cycle.

In phase 2, the tool extracts the failure dependencies from all FRUs, and builds an RBD based on the topology of the storage system. For example, the RBD of the SSU in Table 3.2 is shown in Figure 3.4, where each block is assigned a unique ID to represent an FRU (for the convenience of using graph algorithms, a dummy block is created that does not represent any real FRU as the root (block 0) of all blocks in the RBD). In the RBD, the reliability of each block depends on its parents, while determining that of its children. Given all such extracted failure dependencies, the simulation tool synthesizes the results across all components, and provides detailed information on the estimates of the various metrics of interest, e.g., the average

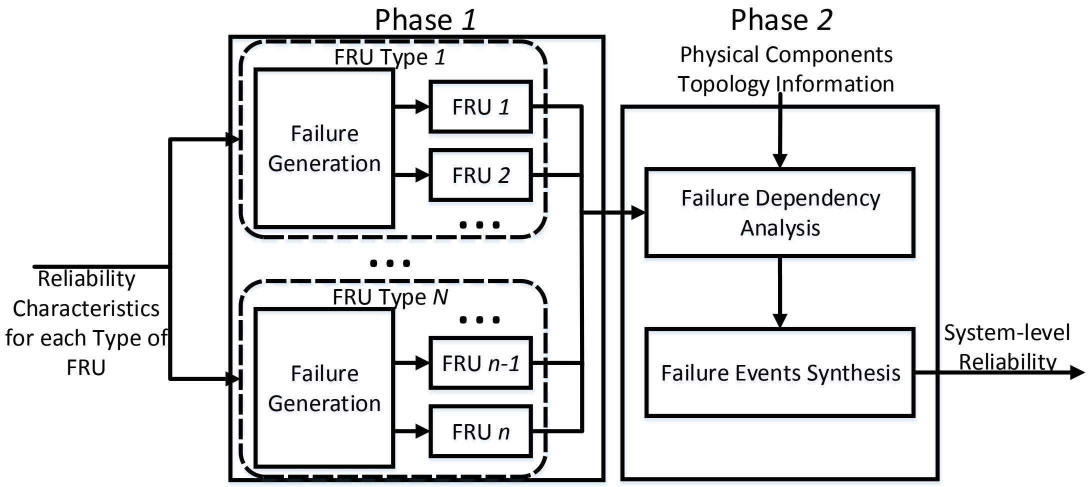


Figure 3.3: Framework of the provisioning tool

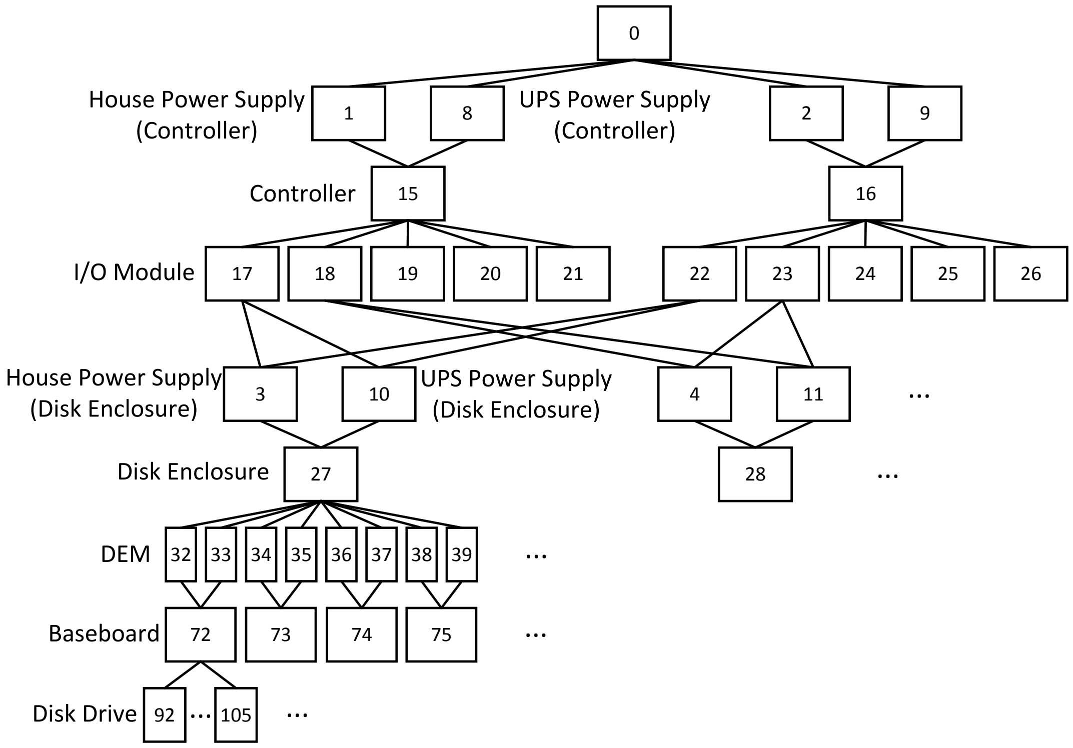

House Power Supply (Controller)
UPS Power Supply (Controller)
Controller
I/O Module
House Power Supply (Disk Enclosure)
UPS Power Supply (Disk Enclosure)
Disk Enclosure
DEM
Baseboard
Disk Drive

Figure 3.4: RBD of a scalable storage unit

number of failed FRUs, events leading to data unavailability or data loss and for how long.

Table 3.3: Parameter settings of the simulation tool

FRU Type	Time between Failure		Time to Repair		Time to Repair (no spare part)	
	Distribution	Parameters	Distribution	Parameters	Distribution	Parameters
Controller	Exponential	rate = 0.0018289	Exponential	rate = 0.04167	Shifted exponential	rate = 0.04167, offset = 168
House Power Supply (Controller)	Weibull	shape = 0.2982, scale = 267.7910	Exponential	rate = 0.04167	Shifted exponential	rate = 0.04167, offset = 168
Disk Enclosure	Weibull	shape = 0.5328, scale = 1373.2	Exponential	rate = 0.04167	Shifted exponential	rate = 0.04167, offset = 168
House Power Supply (Disk Enclosure)	Exponential	rate = 0.0024351	Exponential	rate = 0.04167	Shifted exponential	rate = 0.04167, offset = 168
UPS Power Supply [*]	Exponential	rate = 0.001469	Exponential	rate = 0.04167	Shifted exponential	rate = 0.04167, offset = 168
I/O Module	Weibull	shape=0.3604, scale =523.8064	Exponential	rate = 0.04167	Shifted exponential	rate = 0.04167, offset = 168
Disk Expansion Module (DEM)	Exponential	rate = 0.000979	Exponential	rate = 0.04167	Shifted exponential	rate = 0.04167, offset = 168
Baseboard [*]	Exponential	rate = 0.000252	Exponential	rate = 0.04167	Shifted exponential	rate = 0.04167, offset = 168
Disk Drive	[0, 200], Weibull	shape = 0.4418, scale = 76.1288	Exponential	rate = 0.04167	Shifted exponential	rate = 0.04167, offset = 168
	[200, ∞], Exponential	rate = 0.006031				

[*]Field data missing, vendor-provided AFRs are used.

Implementation and validation

As stated in Section 3.2.2, for each type of FRU, the empirical data of the time between device replacements in Spider I is fitted to four different distributions. In order to choose the best parameter settings for the simulation tool, the Chi-squared test Greenwood and Nikulin (1996) is adopted to determine the probability distribution and corresponding parameters that are more realistic to generate the failure events. To generate the repair time, the exponential distribution with two different mean values, 24 hours for FRUs have a spare part and 168 hours (7 days) for those that do not, is used. In certain cases, when the repair time is much longer, the resource provisioning will be even more critical to improve the overall reliability, as an unoptimized provisioning strategy could lead to a much longer window of vulnerability and higher probability of data unavailability. The chosen distributions and parameters are listed in Table 5.2.

An interesting fact worth noting in Table 5.2 is the distribution parameter settings for generating disk drive failure events. As the analysis of disk failure data reveals (see Figure 3.2(d)), when the time between disk replacements is relatively small (less than 200 hours), a Weibull distribution with decreasing failure rate is a better fit; with the time between disk replacements increasing, the failure rate becomes stable, and an exponential distribution fits the empirical data better. Therefore, we use a method called inverse transform sampling Devroye (1986) to generate disk failure events so that the time between failures fits a crafted distribution, which is actually a join of a Weibull distribution with decreasing failure rate and an exponential distribution with constant failure rate.

After a failure event of a specific FRU type is generated, it will be randomly allocated to an attribute device belonging to that FRU type in the system, and logged as a failure of that device. A random repair time will then be generated and logged for that device so that we can derive all its failure time intervals for the operational period. Once the failure logs of all devices for the operational period are obtained, the tool will synthesize them based on the RBD to derive the duration of temporary data unavailability and permanent data loss. For example, in the RBD shown in Figure 3.4, if the execution results indicate all parents of an FRU are down during a time period, the FRU is tagged as unavailable no matter what its own results are during the same time interval.

During the 5-year operation of Spider I, OLCF only observed two data unavailability events. The lack of empirical data makes it difficult to validate the simulation tool on system-level data availability. However, the simulation results of each type of FRU can be validated using the field-gathered failure data. As listed in Table 3.4, the number of failures of each type of FRU observed in the empirical data are compared against the results from the simulation tool during a 5-year period (the simulation was run 10,000 times, and for each type of FRU the average number of failures were calculated). We can observe that the simulation results approximates to the empirical data, which demonstrates its accuracy.

Table 3.4: Validation on FRU failures estimation

FRU Type	# of Total Units	Empirical # of Failures	Estimated # of Failures	Estimation Error
Controller	96	78	79	1.04%
House Power Supply (Controller)	96	21	27	6.25%
Disk Enclosure	240	14	20	2.5%
House Power Supply (Disk Enclosure)	240	102	105	1.25%
I/O Module	480	22	24	0.42%
Disk Expansion Module (DEM)	1,920	28	42	0.73%
Disk Drive	13,440	264	338	0.55%

3.3 Initial Provisioning

Provisioning an HPC storage system for initial deployment involves understanding the tradeoffs between performance, cost, capacity and reliability. Often times, system architects are provided with a fixed budget for an initial acquisition and deployment, with an emphasis on optimizing for performance and capacity. Reliability characteristics at the SSU-level or at the system-level are also factored in during this phase, with vendor support and spare part pools as the primary vehicles for maintaining system reliability. In this section, we attempt to reconcile these factors for an initial deployment, and study their interplay.

3.3.1 Optimizing for performance

Each SSU can achieve a theoretical peak performance that is primarily determined by the type of the I/O controller and the number of disks in each SSU. An SSU does not necessarily have to be 100% populated (in terms of the number of disks it can accommodate) in order to achieve its peak I/O performance. Therefore, the overall performance of a storage system, consisting of multiple SSUs, can be expressed as:

$$\text{Performance} = N_{SSU} \times min(SSU_{Perf}, D_{SSU} \times BW_{disk}), \tag{3.2}$$

where N_{SSU} is the number of SSUs in the system, SSU_{Perf} is the peak performance of one SSU, D_{SSU} is the number of disks in one SSU and BW_{disk} is the bandwidth achievable from one disk. Equation 3.2 can be optimized independently for sequential or random I/O workloads. However, the selected workload should reflect the design parameters of the storage system and represent the expected production environment.

The cost of the storage system is the sum of the cost of all components (as listed in Table 3.2, with their respective price points per unit). The capacity of the whole system can be expressed as:

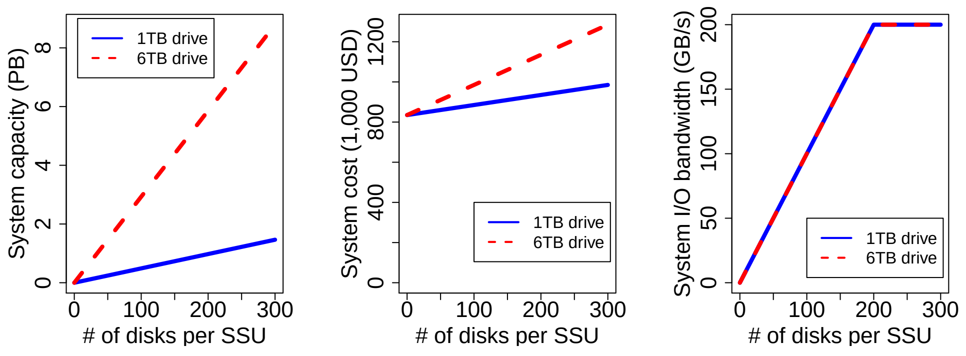
$$\text{Capacity} = C_{disk} \times D_{SSU} \times N_{SSU}, \tag{3.3}$$

where C_{disk} is the capacity of one disk.

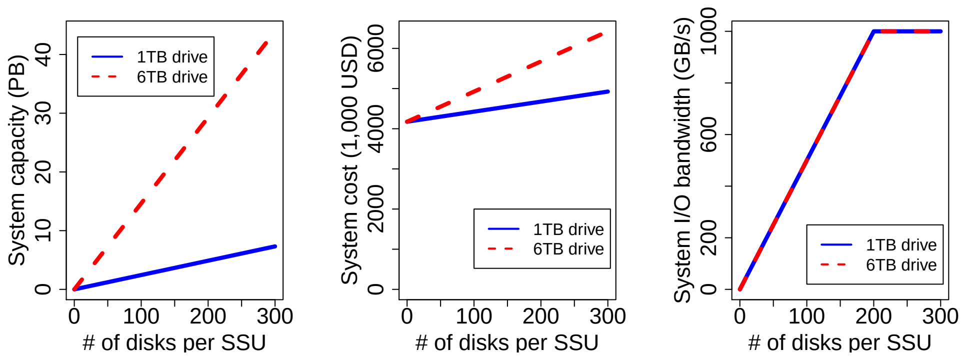
3.3.2 Impact of number of disks and disk capacity

Since disks constitute only 15-20% of the cost of one SSU, the prices of disks do not have the first order impact on provisioning a cost-effective or high-performance storage system. Therefore, when designing a storage system with performance as the primary objective, it is optimal to buy as many SSUs as possible before optimizing or negotiating for disk price or capacity. Once the number of SSUs is fixed (i.e., the peak achievable performance point is fixed), it remains unclear how the number of disks and the storage capacity per disk affect the cost and capacity of the overall system. To study that, next we present a case study where performance goals are set as 200 GB/s and 1 TB/s respectively, and build a storage system with the SSU as characterized by Table 3.2 and Figure 3.1. Note that, the results presented here assume specific parameters for disks and other components of the SSUs, but the same study can be carried out for other chosen parameters.

Let us assume each disk can provide 200 MB/s of bandwidth, therefore 200 such disks are enough to saturate one SSU (assuming a 40 GB/s peak I/O bandwidth per controller pair). Each SSU in our case can accommodate up to 300 disks, therefore



(a) For 200 GB/s system-wide I/O bandwidth performance target



(b) For 1 TB/s system-wide I/O bandwidth performance target

Figure 3.5: The cost and capacity trade-offs

buying any disks beyond 200 is equivalent to buying more capacity. Also, filling an SSU with less than 200 disks (the number of disks that saturate our SSU) always results in lower performance per unit price. The underlying reason is that other components of an SSU significantly dominate the cost of the whole system compared to the disks. Therefore, we should focus on how filling an SSU with 200 to 300 disks changes the capacity and cost of the system (Figure 3.5(a) and 3.5(b)). Let us consider two types of disks (1TB and 6TB, with same I/O performance bandwidth but different costs: 100 and 300 USD, respectively). As expected, the relationship is linear in terms of capacity and cost. It is worth noting that the relative increase in the cost of the system is very modest when going from 200 to 300 disks per SSU. However, if we are going to saturate the performance target (at least 200 drives per SSU), using 6TB drive costs much more money than using 1TB drive.

Also, if redundancy schemes such as RAID 6 is applied and multiple concurrent disk failures in the same RAID group occur simultaneously, a rebuild process is required. In such a scenario, 1TB disks are better than 6 TB disks as the rebuilding is faster if the amount of disk space that needs to be reconstructed is less. This is because the bandwidth usually does not change significantly across these disk types for a given family of disks. Of course, there are technologies that can improve the dynamics of disk redundancy or rebuild process. However, such new technologies are slow to penetrate the storage market. Parity declustering, as an example, substantially reduces the rebuild window by distributing data and redundancy stripes over a number of disks Holland and Gibson (1992). It was first proposed more than two decades ago, and today there are only two products in the HPC storage market that support the parity declustering feature.

3.3.3 Effect of increasing disks/SSU on system reliability

One may also note that there are availability and reliability issues involved in increasing the number of disks. Now let us study how the increase in extra capacity affects the data availability of the system. Using the simulation tool introduced in previous section, we can estimate the number of events when data becomes unavailable in a 1 TB/s system (25 SSUs) for a period of 5 years if no provisioning policy is applied. Based on the disk failure rate calculated from the failure data, the potential cost of disk replacement for a 1 TB/s system during a 5-year period can also be estimated. As can be seen in Figure 3.6, though very modest, the number of data unavailability events and disk replacement cost increase with the number of disks per SSU.

Therefore, fixed initial provisioning can be optimal from cost efficiency and capacity perspectives, but it alone is not sufficient for improving the reliability dynamics. A well-designed, continuous provisioning policy is needed to maintain the system's data availability requirement under a fixed provisioning budget. This is also true if we plan to increase the disks/SSU for extra capacity. In fact, if an optimal

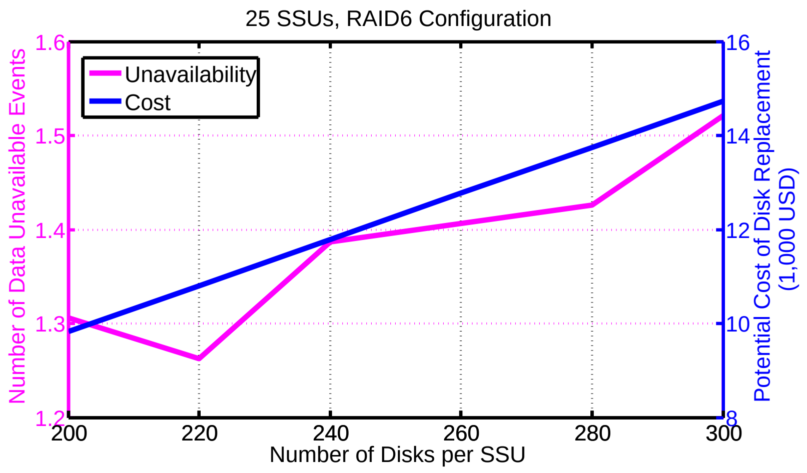


Figure 3.6: Number of data unavailable events and potential disk replacement cost for 1 TB/s systems (25 SSUs)

continuous provisioning policy is adopted, the unavailability caused by increasing the disks/SSU could be significantly mitigated.

3.4 Continuous Provisioning

Ideally, if we have an unlimited budget for spare provisioning, we can provide unlimited spares for each component in the system. However, in reality the budget is always limited, and we can only provision a limited number of spares. Therefore, the goal of continuous provisioning policy is to explore such dynamics under constraints.

3.4.1 Ad hoc provisioning

Most of the provisioning policies used in large-scale HPC storage systems are ad hoc, and are based on system administrators' intuition and experiences. Here we use Spider I to illustrate the ad hoc policies. As listed in Table 3.2, vendor-provided statistics for Spider I indicate that controllers have the highest failure rate among all FRUs, which can also be verified by the actual device replacement data. Thus, the first intuitive provisioning policy would be to provision as many controller spares as possible for a given provisioning budget. However, the controller-first provisioning policy does not improve the system data availability significantly when compared against not provisioning any budget for spares at all, as the two controllers in the same SSU are configured as a fail-over pair in the Spider I architecture. Only when both of them are down simultaneously does it lead to data unavailability, which is a rare event in practice.

The deficiency of the controller-first provisioning policy suggests that the built-in hardware redundancy might have more impact on data availability compared to the component failure rates. In other words, if the hardware redundancies are not well-designed, it could make the system more vulnerable to failures of some devices (also observed in Spider I). As shown in figures 3.1 and 3.4, the failure of a disk enclosure causes two disks in the same RAID group to become unavailable simultaneously. On the other hand, all the other FRUs combined will lead to at most one disk unavailability in each RAID group. This means that the storage system is more vulnerable to disk enclosure failures (Actually, The 5-disk enclosure architecture of Spider I was selected for minimizing the cost. However, this selection resulted in lower data availability. This was a lesson learned from the Spider I experience, and rectified in Spider II by switching to a 10-disk enclosure configuration.). Therefore, a more effective ad hoc provisioning policy for Spider I is to provide spares for disk enclosures first.

Based on the analysis of the system architecture and the redundancy characteristics of Spider I, we realized that most potential data unavailability scenarios could be caused by simultaneous failures of different types of FRUs (e.g., a disk enclosure failure coupled with a double power supply failure on another enclosure). If the budget is allocated for provisioning a specific type of FRU first, there might not be enough left to maintain spares for other types of FRU, which could negatively impact the data availability. Thus, neither controller-first nor enclosure-first provisioning policy is optimal and an optimized dynamic spare provisioning model is needed.

Table 3.5: Notations of symbols

N	Number of types of FRU in system
FRU_i	i-th type of FRU
$f_i(x)$	PDF of time between failures of FRU_i
$F_i(x)$	CDF of time between failures of FRU_i
$h_i(x)$	Hazard rate of FRU_i
$MTBF_i$	Mean time between failures of FRU_i
$MTTR_i$	Mean time to repair of FRU_i
τ_i	Delay caused by waiting for a new FRU_i to be delivered
t_i^{fail}	Time point when last failure of FRU_i occurred
t^{cur}	Current time when we need to update the spare pool
t^{next}	Next time when we need to update the spare pool
m_i	Impact FRU_i has on data unavailability
b_i	Unit price of FRU_i
B	Annual budget for spare provisioning

3.4.2 Dynamic spare provisioning model

This model aims to optimize the spare provisioning policy for large-scale storage systems in order to achieve high data availability, given a limited provisioning budget. The notations of all the symbols used in this section are listed in Table 3.5.

Intuition and assumption

The impact each FRU has on system availability is usually determined by two factors: its own reliability (e.g., some FRUs fail less often than others, or can be repaired more quickly) and the system architecture (e.g., in Spider I, as the lack of hardware redundancy makes the system more vulnerable to disk enclosure failures, disk enclosures have more impact on data availability). However, few existing ad hoc provisioning policies focus on these two factors simultaneously. Therefore, the basic idea behind this spare provisioning optimization model is to quantify both of these factors, and allocate more budget towards provisioning spare parts for FRUs that have more impact on the data availability of the storage system.

The assumption about the provisioning budget made here is simple but realistic. Specifically, at the beginning of each year, system administrators get a fixed budget that we call the annual budget, and use it to prepare spares for different FRUs according to a specific provisioning policy.

Reliability characteristics of different FRUs

Since we already have the field-gathered failure data and vendor-provided AFR, quantifying the reliability of each type of FRU is easy. We obtain the probability density function (PDF) of the time between failures of each type of FRU by fitting the failure data. Thereafter, we can estimate the number of failures that will occur during a future period for each type of FRU. For example, if we use $f_i(x)$ to denote the PDF of the time between failures of FRU_i, then the CDF can be calculated as $F_i(x) = \int_0^x f_i(t)dt$. Based on this definition, the hazard rate of FRU_i is given as:

$$h_i(x) = \frac{f_i(x)}{1 - F_i(x)} \tag{3.4}$$

Let us assume the last failure of FRU_i occurred at time t_i^{fail}. We need to estimate the number of failures, y_i, of FRU_i between the current time, t^{cur}, when the spare pool is being updated and the next time, t^{next}, the spare pool will need an update. This is given as follows:

$$y_i = \int_{t^{cur}-t_i^{fail}}^{t^{next}-t_i^{fail}} h_i(x)dx \tag{3.5}$$

The above formula can estimate the expected number of failures between t^{cur} and t^{next} accurately if the time between the failures fits an exponential distribution, which has a time-independent hazard rate. However, for a Weibull distribution, if the time between updating the spare pool is relatively longer compared to the mean time between failures, this formula cannot give an accurate estimation. This is because, once a failure occurs between t^{cur} and t^{next}, which is very possible because of the short mean time between failures, the hazard rate should increase. Therefore, for a

Weibull distribution, if

$$\frac{t^{next} - t^{cur}}{MTBF_i} > \int_{t^{cur}-t_i^{fail}}^{t^{next}-t_i^{fail}} h_i(x)dx, \tag{3.6}$$

we can use

$$y_i = \frac{t^{next} - t^{cur}}{MTBF_i}, \tag{3.7}$$

instead of formula 3.5, where $MTBF_i$ is the mean time between failure of FRU_i. Given the PDF of the time between failures of FRU_i, $f_i(x)$, $MTBF_i$ can be calculated by solving $\int_0^\infty x f_i(x)dx$.

In Spider I, if no spare part was available on-site, a device replacement could be delayed by at least 7 days. If we use $MTTR_i$ to denote the mean time to repair of FRU_i when spare parts are available on-site, τ_i to denote the delay caused by waiting for a new FRU_i to be delivered, then the mean time spent on replacing FRU_i is $MTTR_i + \tau_i$, if there is no spare on-site when the replacement is required.

Impact of system architecture on availability

To quantify the impact of the system architecture on data availability, we need to analyze the physical structure of the system, and derive the failure dependencies between different FRUs. Here we consider one SSU of Spider I as an example. All FRUs of this SSU are listed in Table 3.2.

The RBD illustrates the failure dependencies between different FRUs. By analyzing the structure of the RBD, we can derive the impact each FRU has on the data unavailability of the storage system. For instance, each RAID group in the SSU in Figure 3.4 contains 10 disk drives, which are organized as RAID level 6, and can tolerate 2 disk failures. Each leaf block represents a disk drive, and there are 16 different paths from one leaf block to the root. On each of these 16 paths, if one FRU fails, that path will be unavailable. If and only if all of these 16 paths are unavailable, the associated disk drive will become unavailable. If more than 2 disk drives in one

RAID group are unavailable, a data unavailability occurs. In fact, the more available paths each RAID group has, the more reliable each RAID group is. Therefore, we can quantify the impact of each FRU on data unavailability by counting the number of paths that will become unavailable in one RAID group, if such an FRU has been removed from the RBD.

Specifically, since triple-disk unavailability in one RAID 6 group leads to data unavailability, we only count unavailable paths of each triple-disk combination in one RAID group. For example, failure of one controller makes every disk in one RAID group lose 8 paths, while the failure of one disk enclosure only makes two disks in one RAID group completely unavailable (each loses 16 paths). Therefore, we use $8 \times 3 = 24$ as the impact of a controller, while $16 \times 2 = 32$ as that of a disk enclosure. Table 3.6 shows the impact of each FRU quantified in this way.

Table 3.6: Quantified impact of each type of FRU

FRU Type	Quantified Impact
Controller	24
House Power Supply (Controller)	12
UPS Power Supply (Controller)	12
Disk Enclosure	32
House Power Supply (Disk Enclosure)	16
UPS Power Supply (Disk Enclosure)	16
I/O Module	16
Disk Expansion Module (DEM)	8
Baseboard	16
Disk Drive	16

Optimization model and dynamic provisioning algorithm

In order to maximize data availability, our optimization model tries to minimize the total unavailable time of the end-to-end paths that belong to each triple-disk combination of a RAID group in the RBD. For example, as mentioned above, one disk enclosure failure makes a triple-disk combination in one RAID group lose 32 end-to-end paths. If the disk enclosure has no spare part on-site and cannot be replaced quickly, those 32 end-to-end paths will be unavailable for a longer duration, which increases the probability that all end-to-end paths of the triple-disk combination become unavailable within the same time interval.

We define a variable x_i to denote how many spare parts are provided for FRU_i. Then, the total unavailable time of the end-to-end paths, caused by failures of FRU_i can be calculated as

$$\Delta t_i^{down} = m_i x_i MTTR_i + m_i(y_i - x_i)(MTTR_i + \tau_i), \tag{3.8}$$

where m_i is the number of unavailable end-to-end paths caused by failures of FRU_i (see Table 3.6) and y_i is the estimated number of FRU_i failures that would occur before the next spare pool update.

Let us assume the unit price of FRU_i is b_i, the annual budget for spare provisioning is B. Then, we can establish the following linear programming optimization model to find out how many spares should be prepared for each type of FRU in the coming year.

$$\arg\min_{x_i} \sum_{i=1}^{N} m_i y_i (MTTR_i + \tau_i) - m_i x_i \tau_i; \tag{3.9}$$

$$\text{s.t.} \sum_{i=1}^{N} x_i b_i \leq B; \tag{3.10}$$

$$x_i \leq y_i, \forall i \in \{1 \ldots N\} \tag{3.11}$$

In this linear programming model, the objective function is to minimize the total unavailable time of the end-to-end paths that belong to each triple-disk combination of a RAID group in the RBD. The two constraints are that the total provisioning cost cannot exceed the annual budget and for each type of FRU, the number of provisioned spares should not exceed the expected number of failures.

The pseudo-code of the spare provisioning algorithm is shown in Algorithm 1. At the beginning of each of year, system administrators can first check the spare pool and find out how many spare parts each type of FRU has. Then they can obtain all

Algorithm 1 Spare Provisioning Algorithm

```
Input: Current spare pool $SP$, replacement log of each type of FRU,
  unit price of each type of FRU, annual budget for spare provisioning $B$.
Output: Spare provisioning results $\mathbf{X} = [x_1, x_2, \ldots, x_N]$.
  Obtain number of spares in $SP$, $\mathbf{n} = [n_1, n_2, \ldots, n_N]$;
  Calculate $[m_1, m_2, \ldots, m_N]$;
  Calculate $[MTTR_1, MTTR_2, \ldots, MTTR_N]$
  for $i = [1, 2, \ldots, N]$ do
    Calculate $y_i$, the expected number of failures of $FRU_i$;
    Add $y_i$ into $\mathbf{Y}$, and $MTTR_i$ into $\mathbf{MTTR}$;
    Add $m_i$ into $\mathbf{m}$, and $b_i$ into $\mathbf{b}$;
  end for
  $\mathbf{X} = \textsc{ResolveOptimizationModel}(\mathbf{Y}, \mathbf{MTTR}, \mathbf{m}, \mathbf{b}, B)$
  for $i = [1, 2, \ldots, N]$ do
    if $n_i < x_i$ then
      Add $(x_i - n_i)$ spares $FRU_i$ in to $SP$;
    end if
  end for
```

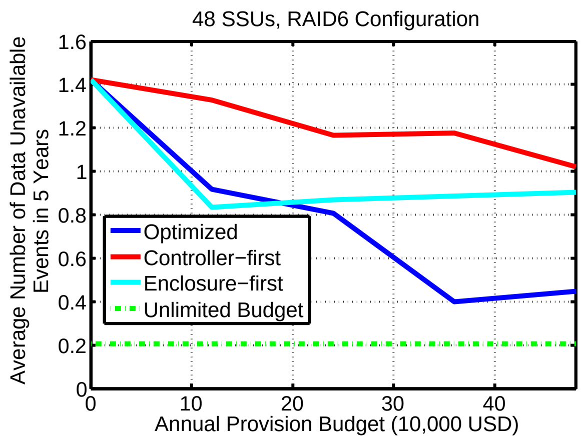
required parameters and resolve the optimization model to find out how many spare parts should be provisioned for each type of FRU. Finally, based on the optimization results, they add the needed spare parts into the spare pool.

3.4.3 Continuous provisioning evaluation

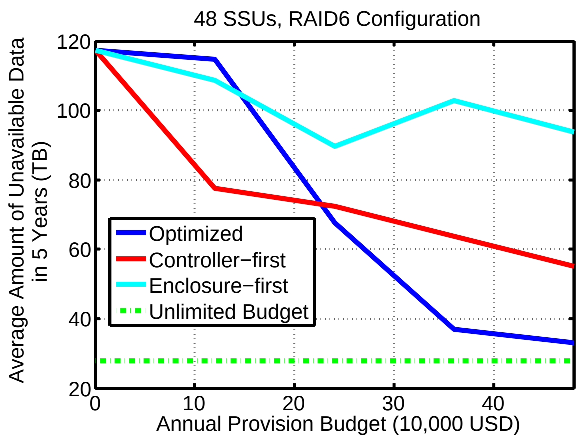
Now we compare the performance of the continuous provisioning model with the two ad hoc provisioning policies introduced at the beginning of this section in terms of data availability and provisioning cost by using the simulation tool. Spider I is still the simulated large-scale storage system.

Evaluation of data availability

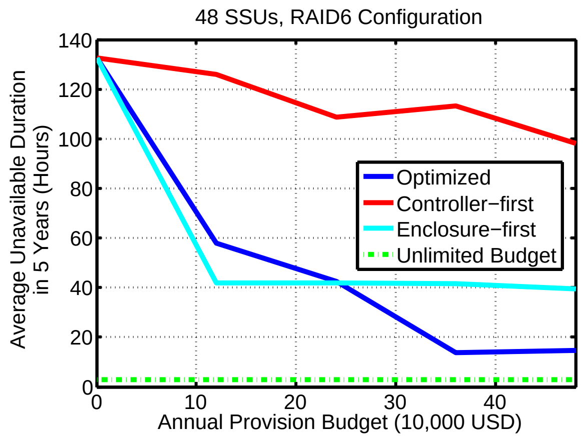
We first present the evaluation results of different provisioning policies to demonstrate the effectiveness of our optimized provisioning policy in reducing data unavailability. Note that besides the two ad hoc provisioning policies mentioned before, we also include the evaluation result of the scenario when unlimited provisioning budget is provided, which gives the lower bound for the data unavailability. Here, unlimited



(a) Number of data unavailability



(b) Amount of unavailable data



(c) Average unavailable duration

Figure 3.7: Performance comparison between different provisioning policies

provisioning budget means every individual component in the system can have a spare part on-site. For example, in Spider I there are 96 controllers, thus we can maintain 96 spare controllers in the spare pool if unlimited budget is provided.

First, we present the results of the average number of data unavailability events during the 5-year operation of 48 SSUs using different provisioning policies in Figure 3.7(a). The results illustrate that at least one data unavailability event will occur during 5 years if no provisioning policy is used (no provisioning budget is provided). Further, the optimized provisioning policy can reduce the data unavailability more significantly with increasing provisioning budget when compared to the ad hoc policies.

Since one data unavailability event might cause multiple RAID groups to become unavailable simultaneously, the volume of data that can become unavailable due to even a single unavailability event can range in the tens of terabytes. For the Spider I file system, each RAID level 6 group is composed of 10 1TB disks. Figure 3.7(b) shows the average amount of data that might become unavailable under different provisioning policies during 5 years of operations. We calculate this with the knowledge of how many RAID groups are affected by each data unavailability event. Similar to the results shown in Figure 3.7(a), the optimized provisioning policy can also reduce the amount of unavailable data significantly. For example, with an annual spare provisioning budget of just $480K, the optimized provisioning policy can protect as much as 90TB from becoming unavailable during the 5-year operation of the storage system.

Moreover, the optimized provisioning policy also decreases the duration of data unavailability as shown in Figure 3.7(c). For the same $480K annual provisioning budget, the optimized provisioning policy reduces the duration of data unavailability for the 48 SSUs in aggregate by as much as 52% (more than 20 hours) and 81% (more than 80 hours) compared to the enclosure-first and controller-first provisioning policies.

Evaluation of provisioning cost

First, we illustrate the total provisioning cost during 5 years using different provisioning policies, given different annual budgets, in Figure 3.8. Different from the two ad hoc policies, which try to squeeze every penny of the budget, the cost of our optimized provisioning policy does not increase with the budget linearly. The reason behind this is that the optimized provisioning policy allocates budget based on accurate failure estimation and failure dependency analysis, which are more economical and efficient compared to the ad hoc policies. Actually, by using the optimal provisioning policy, the savings can be more than 10% of the total storage system cost over the operational life of a large-scale storage system.

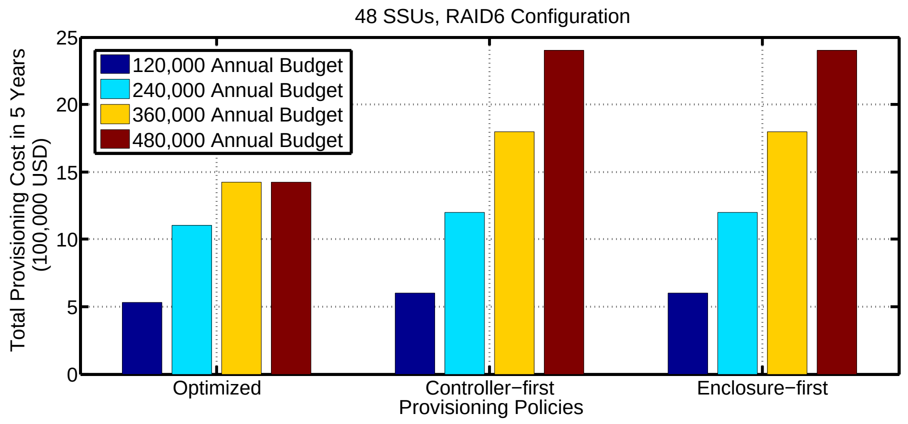


Figure 3.8: Total provisioning cost in 5 years using different provisioning policies.

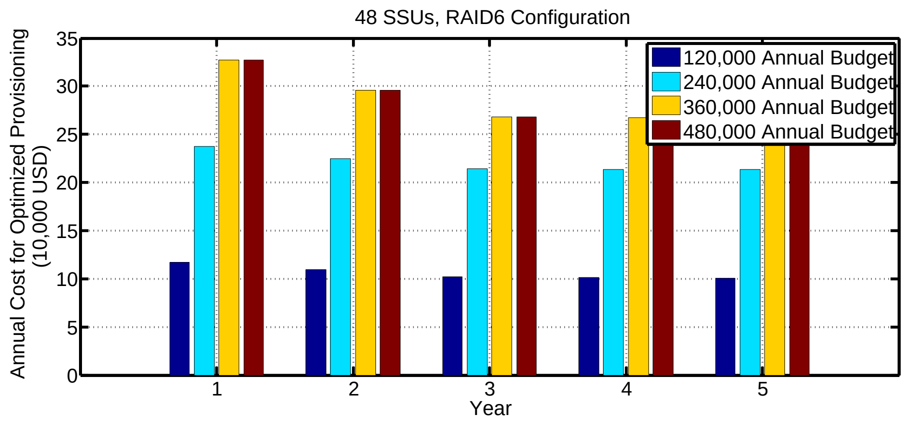


Figure 3.9: Annual cost for optimized provisioning policy.

Finally, we illustrate the cost for spare provisioning the 48 SSUs in each year using the optimized provisioning policy, given different annual budget limits (Figure 3.9). Two interesting observations can be made from Figure 3.9. First, the annual provisioning cost decreases year after year. This is because many FRUs in Spider I have decreasing failure rates (see Figure 3.2). Second, increasing the annual provisioning budget does not necessarily increase the annual provisioning cost. For example, when the annual budget is increased to $480K, the provisioning cost is almost the same as when the annual budget is \$360K. This is because the optimized provisioning policy attempts not to over-provision the spare parts, i.e., no more spare part will be added if the number of existing ones is equal to the number of FRUs that are expected to fail next year, resulting in cost savings.

Chapter 4

Optimal Workload-Adaptive Data Placement

4.1 Problem Formulation

With the development of storage technologies, SSDs have been eventually exploited by large-scale storage systems, as typically they can provide much higher I/O performance compared to conventional hard drives Park and Shen (2009b). However, SSDs are also limited in capacity and much more expensive than hard disk drives, meaning that it is not yet practical to use them to completely replace conventional hard disks. Therefore, how to cost-effectively integrate SSDs into large-scale storage systems and design efficient data placement mechanisms for such heterogeneous storage environments to maximize the I/O performance and improve the storage space efficiency becomes a critical yet challenging task.

The core problem can be formulated as follows. Given a heterogeneous storage system composed of both HDDs and SSDs, our task is to find a data placement solution that 1) satisfies user polices on data placement, 2) maximizes the data access throughput of a mixture of workloads produced by different user applications, and 3) improves the storage space efficiency without degrading the data availability. This problem is challenging due to 1) we do not have complete knowledge on future access patterns of data objects as they could change dynamically, 2) user policies can be highly heterogeneous and may change over time. Therefore, the feasible solution to this problem must be able to adapt to the varying I/O workloads and dynamically adjust the data placement to achieve the optimal performance.

4.2 System Model

The system model of our optimal workload-adaptive data placement is built upon several assumptions. First, the I/O workloads from user applications include both read and write operations, and the access pattern of these I/O operations could be either sequential or random. This assumption is often true in realistic I/O workloads collected from large-scale data centers, web server clusters and HPC environments. Second, the modern large-scale storage systems are usually object-based, in which the minimal data unit is called object. In practice, a large file can be divided into multiple data objects which will be stored on single or multiple object-based storage devices (OSDs). Different from the block-based storage, in object-based storage systems, the access history of each data object can be tracked more easily, meaning that it is possible to obtain the popularity of each data object. Third, the I/O workloads from user applications may change over time, therefore, the solution should adapt to the dynamic nature of the I/O workloads.

Figure 4.1 shows the overall system model, where the whole procedure works as follows: the first core component, the classification model, is trained based on the access history and access patterns of data objects. In our current work, we concentrate on the historical access frequency of data objects, while we leave exploiting the access pattern (sequential/random read or write) to improve data placement performance as our future work Wan et al. (2014a). After training, it provides parameters for the runtime prediction model which is used to predict the access popularity of data objects in the near future. Specifically, the prediction results decide if an object is going to

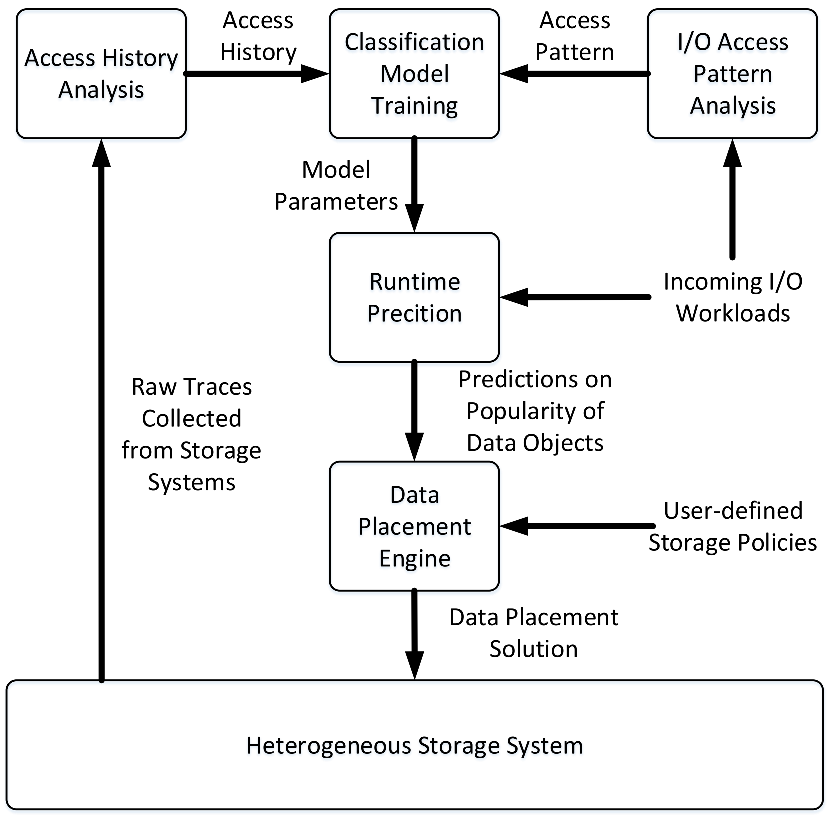

Access History Analysis
Access History
Classification Model Training
Access Pattern
I/O Access Pattern Analysis
Model Parameters
Runtime Precition
Incoming I/O Workloads
Raw Traces Collected from Storage Systems
Predictions on Popularity of Data Objects
Data Placement Engine
User-defined Storage Policies
Data Placement Solution
Heterogeneous Storage System

Figure 4.1: The system model

have "recurring" or "non-recurring" accesses, based on its history of accesses. Such prediction results are then used, together with user-defined storage policies, as the input of the data placement engine, whose goal is to generate an optimal placement of data objects among heterogenous storage devices so that the overall data access performance as well as the storage space efficiency can be improved.

4.3 Algorithm Design

Ideally, if we can record all access history of each data object, we can have the most accurate prediction on future data access. However, in reality it may not be practical to record even a relatively long access history for each data object since the trace collection overhead could be huge in that case. Therefore, in our design we need to achieve a reasonable tradeoff between the trace collection overhead and the prediction accuracy. To accomplish this goal, we intend to only maintain recent access history and utilize the temporal locality commonly existing in access history of data objects to dynamically predict the future popularity of data objects.

4.3.1 Temporal locality in data objects access

Temporal locality commonly exists in data accesses and has been widely studied and utilized in design of caching systems Megiddo and Modha (2003). Actually, many existing studies have revealed that one data object that is being frequently accessed is very likely to be accessed again in the near future. Here we use LASR traces Kuenning (2005) as a case study. The LASR traces include I/O workloads produced by different kinds of user applications running upon a network-based storage system. As demonstrated in Figure 4.2, we extract the access frequency (here the "access" could be either read or write operation) of a single data object during one month from the LASR traces. The X axis of Figure 4.2 is the range of one month time that has been divided into 720 time periods (each period is 1 hour). The Y axis represents the number of times the data object has been accessed during each time period. From the figure, we can observe that although the access frequency of such data object eventually decreased overtime, during some short time period, it was still heavily accessed. How to utilize such temporal locality will be introduced in next section.

Since maintaining the entire access history of each data object is not cost-effective, we only maintain recent access history for each data object. As shown in Figure 4.2, only the access history in the dotted window is used to train the prediction model. Moreover, such window will slide with time and the prediction model will be updated based on the recent-collected access traces.

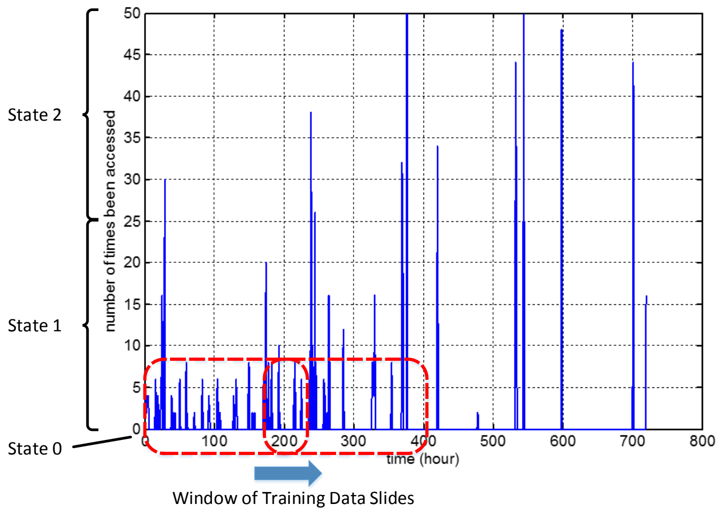


Figure 4.2: Access frequency of one data object in LASR traces

4.3.2 Markov chain prediction model

With the access history of each data object, we build and train a Markov chain model to predict the future access frequency of data objects. First, we need to determine how many states the Markov chain should have and the range of access frequency each state represents. Based on extensive analysis of the I/O traces, we find that three states are often enough for a Markov model to capture the temporal locality and produce fair prediction results. For example, as shown in Figure 4.2, if the maximum number of access times during an observation period is 50, then, for example, we can divide 50 evenly into two ranges, and build a Markov chain model that has three states: 0, $(0, 25]$, and $(25, 50]$, respectively. If during a time period, there is no access of the data object, then the Markov chain will stay in state 0. If the number of access times is larger than 0 but less than 25, then the Markov chain will stay in state 1, and so on. The transition diagram of the Markov chain is shown in Figure 4.3.

Second, we transform the access history to the state transition sequence of the Markov chain based on the specific range each state represents. For example, after transformation the state transition sequence of access history shown in Figure 4.2 is: $1, 1, 1, 1, 1, 0, 0 \ldots$ Based on this state transition sequence, we can estimate the

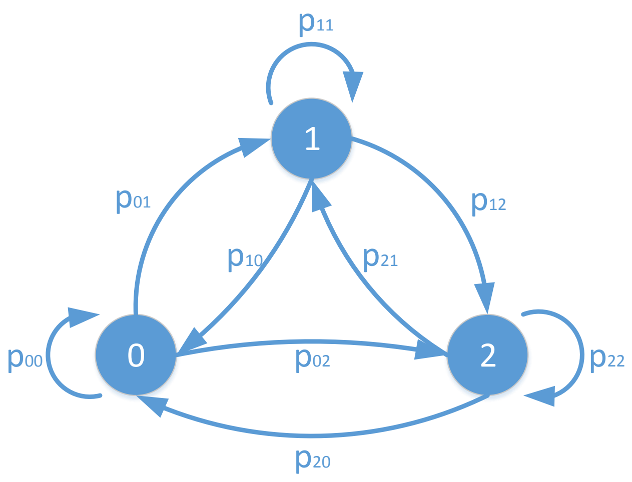


Figure 4.3: Transition diagram of Markov chain

transition probabilities between every two states and construct the transition matrix of the Markov chain shown below:

$$\mathbf{T} = \begin{pmatrix} p_{00} & p_{01} & p_{02} \\ p_{10} & p_{11} & p_{12} \\ p_{20} & p_{21} & p_{22} \end{pmatrix} \tag{4.1}$$

According to the properties of Markov chain, we have:

$$\lim_{n\to\infty} \mathbf{T}^n = \begin{pmatrix} \pi_0 & \pi_1 & \pi_2 \\ \pi_0 & \pi_1 & \pi_2 \\ \pi_0 & \pi_1 & \pi_2 \end{pmatrix} \tag{4.2}$$

in which $\boldsymbol{\pi} = [\pi_0, \pi_1, \pi_2]$ is called the stationary distribution of the Markov chain. We can simply calculate $\boldsymbol{\pi}$ through computing a normalized multiple of a left eigenvector $\mathbf{E}$ of the transition matrix $\mathbf{T}$ with an eigenvalue of 1:

$$\boldsymbol{\pi} = \frac{\mathbf{E}}{\sum_i e_i} \tag{4.3}$$

where e_i is the i-th element of eigenvector $\mathbf{E}$. Since the stationary distribution $\boldsymbol{\pi}$ reflects the probabilities that each state of Markov chain will be visited in the future, which can be used as a prediction of the future access frequency of each data object.

Based on the prediction results, we rank the data objects so that we can determine which data object should be placed or moved to SSD drives. Note that, however, even if the calculated stationary distribution tells us state 1 will be visited with higher probability than state 2, to rank the importance of the data object, we must consider that state 2 represents a higher access frequency. Therefore, we use a weighted sum of the stationary distribution to rank the importance of the data objects, where the weights are defined by values that are proportional to the access frequency ranges that the states represent. For example, if we obtain the stationary distribution of the data object as $\boldsymbol{\pi} = [0.31, 0.56, 0.13]$, and we assign weights $[0, 10, 20]$ to the three different states, we can calculate the rank of the data object by $rank_{obj_x} = 0.31 \times 0 + 0.56 \times 10 + 0.13 \times 20 = 8.2$. The calculation results will then be fed to the data placement engine to find an optimized data placement solution to improve the data access performance and storage space efficiency.

4.3.3 Optimal data placement for maximizing data access throughput

Let us introduce how to maximize the data access throughput of the heterogenous storage system through optimal data placement first. In this aspect, we assume that users' requests will be parametric, meaning that all requests will be embedded into equations or constraints. For example, by using the notations in Table 4.1, a requirement on the number of replicas maintained for data object i stating that at least three must be made can be expressed as $n_i \geq 3$. Now we can formulate the data

Table 4.1: Notations of symbols

M	Total number of storage drives
N	Total number of data objects
C_j	Capacity of storage drive j
s_i	Size of data object i
$\hat{f}_i$	Future access frequency of data object i
thr_i	Average throughput for storage drive j
e_{ij}	Whether data object i is stored on storage drive j (0 or 1)
n_i	Number of replicas data object i has

placement problem as an optimization model like follows:

$$\arg\max_{e_{ij}} \sum_{i=1}^{N} \hat{f}_i \times max[thr_j \times e_{ij}, \forall j \in M]; \tag{4.4}$$

$$\text{s.t.} \sum_{\forall i \ \ s.t. \ \ e_{ij}=1} s_i \leq C_j, \forall j \in M; \tag{4.5}$$

$$\sum_{j=1}^{M} e_{ij} = n_i, \forall i \in N \tag{4.6}$$

In this optimization model, the objective function specifies that we aim to find a way that assigns data objects to heterogeneous storage devices (HDDs and SSDs) so that the average throughput of accessing each data object is maximized. Note that we use the equation $thr_j \times e_{ij}$ to filter those storage devices where the particular data object i is stored on: if i is stored on device j, we know $e_{ij} = 1$, otherwise $e_{ij} = 0$. Meanwhile, the constraints of our model specifies storage system requirements and user policies. For instance, in our model here, the first constraint states that the data objects stored on a storage device should not exceed the capacity of that device, while the second constraint denotes how many replicas each data object should have. In fact, more constraints can be integrated into this model. For example, in order to mitigate resource contention and avoid the appearance of *hotspot*, we might need to add a constraint to limit the I/O workloads each storage device handles in short time period.

4.3.4 Adaptive data replication for increasing the storage space efficiency

In order to reduce the data unavailability caused by device failures or system crashes, thereby improve the user experiences, many modern large-scale storage systems have introduced the data replication schemes into their functionalities. For example, Google file system (GFS) Ghemawat et al. (2003b) intends to distribute replicas of same data block to different physical locations (usually different racks), while Ceph file system Weil et al. (2006a), which is based on CRUSH algorithm Weil et al. (2006b), divides OSDs of a storage system into different failure domains based on the storage system's physical structure and places replicas of same data object on OSDs belong to different failure domains, since the correlation between failures of OSDs in different failure domains is small.

In both GFS and Ceph file system, the number of replicas created for each data object is set to be a same constant value, which means all data objects are treated with same priority through time. However, in reality, the pre-configured, constant number of replicas adopted by most existing data replication and placement approaches is not optimal, since not all data objects have the same popularity. Intuitively, if the storage space is limited, more replicas should be created for those frequently-accessed data objects, because requests for those popular data objects are more likely to encounter simultaneous failures of large number of storage devices than those unpopular ones. Moreover, as the popularity of data objects varies with time, the number of replicas of each data object should also be changed dynamically. Therefore, we propose an adaptive data placement algorithm which can dynamically adjust the number of replicas for each data object based on their popularity. By using our algorithm, we can guarantee the data availability while increase the storage space efficiency significantly.

As shown in Algorithm 2, every time our data replication algorithm is triggered, it utilizes the history I/O traces to train the Markov model and predict the future

Algorithm 2 Adaptive Data Replication Algorithm

```
1: Obtain I/O traces in previous 6 days for each data object $d_i$
2: Preprocess the traces and feed them to the Markov model to predict the future access
   frequency $\hat{f}_i$ for each data object $d_i$
3: Calculate how possible data object $d_i$ could be unavailable in the future $p_i = \frac{\hat{f}_i}{\sum \hat{f}_i}$
4: Resolve the optimization model
       $\arg\max_{n_i} \sum_{i=1}^{N} (1 - p_i^{n_i})$
       $\text{s.t.} \sum_{i=1}^{N} n_i s_i \leq \sum_{j=1}^{M} C_j$
5: Adjust the data replication based on the optimization result
6: Wait until next optimization is launched, go to step 1
```

popularity (access frequency) of each data object. The prediction results can be used to indicate how possible the requests for each data object could fail in the near future. Here our assumption is if the requests for a data object are issued more often, these requests are more likely to encounter data unavailability caused by simultaneous failures. Then our algorithm resolves the optimization model whose objective is to maximize the likelihood that the requests for all data objects are successful. Finally, the optimization results are applied to the adjustment of the number of replicas for each data object.

4.4 Evaluation

In this section, we present the evaluation results of the proposed data placement and replication algorithms.

4.4.1 I/O trace analysis and preprocess

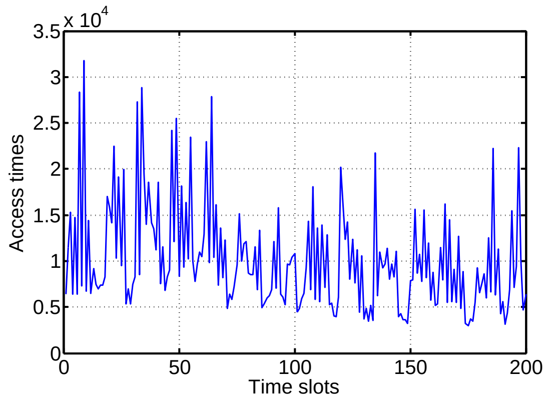
We first present a study on the traces of data object accesses, based on which we evaluate the performance of our data placement algorithms by replaying these traces.

We use a long-term I/O traces, LASR traces Kuenning (2005), which were collected at system-call level. We track the access frequency of different files during their lifetime. Specifically, we divide the time span into hundreds of time slots and each of which has same length. We then count how many times each file was accessed during each time slot. In the LASR traces, we eliminate those files which were accessed less than 10 times during their lifetime (the accesses of these files almost have no impact on the performance of the storage system) and focus on the remaining ones (1,703 files) which were more frequently accessed.

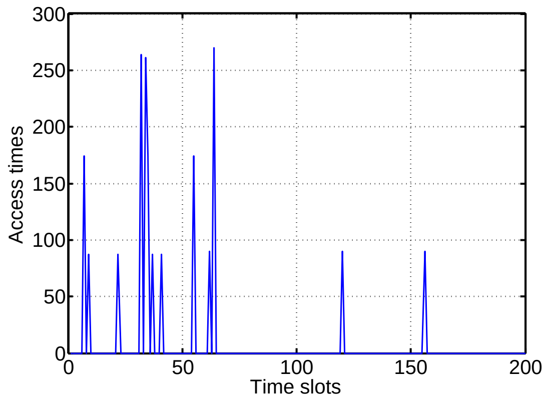
By analyzing the access history of these frequently accessed files, we find out that these files can be roughly put into two categories according to their access patterns. The first category contains files that have constant access patterns. Files in this category were frequently accessed during their whole lifetime, without too much difference between the maximum and minimum access periods. Figure 4.4(a) shows a typical file falls into this category. The data popularity prediction model, especially our Markov chain based approach can achieve a higher level of accuracy for this kind of files. The second category contains files with a bursty access pattern. Files in this category were only accessed during a few time slots, but within those time slots the access counts could be very large. Figure 4.4(b) shows a typical file falling into this category. For files belong to the second category, it is relatively difficult for any prediction algorithm, including our Markov chain based approach, to accurately predict their future access frequency.

4.4.2 Evaluation on optimal data placement model

Before every time the data placement optimization is launched, we use the traces of previous 6 days as the training data to train our Markov prediction model while use the traces of next 6 days as the testing data. Due to the space limits, here we only illustrate the prediction results for 40 data objects (files) in the LASR dataset. As illustrated in Figure 4.5, the bars represent the future access frequency of the



(a) Data access type I



(b) Data access type II

Figure 4.4: Illustration of different data access types

40 different files which are extracted from the testing dataset. Since we only have limited SSD storage space, our goal is to store the files that will be most frequently accessed in the future on SSD devices to improve the average data access throughput. For example, if we can only put 10 of 40 files on SSDs, as shown in Figure 4.5, the light-colored bars illustrate the files predicted by our Markov model that should be placed on SSD devices. From the results we can observe that, 7 of 10 files that have the highest future access frequency have been chosen by our prediction algorithm.

We next choose random selection approach as baseline and compare the average read throughput achieved by our Markov-based model with that achieved by random selection algorithm. Here the random object selection means that we randomly choose several data objects and put them on SSD devices. The number of data objects that can be placed on SSD devices is also limited. For example, in the simulation, we vary the number of data objects that can be put on SSD devices from 2.5% to 50%. Besides, we set the read throughput of SSD devices as 550 MB/s and that of HDD devices as 120 MB/s, consistent with typical datasheets provided by manufacturers of storage devices Chen et al. (2009). As shown in Figure 4.6, our object selection approach can achieve higher average read throughput than random selection, demonstrating the effectiveness of our proposed approaches.

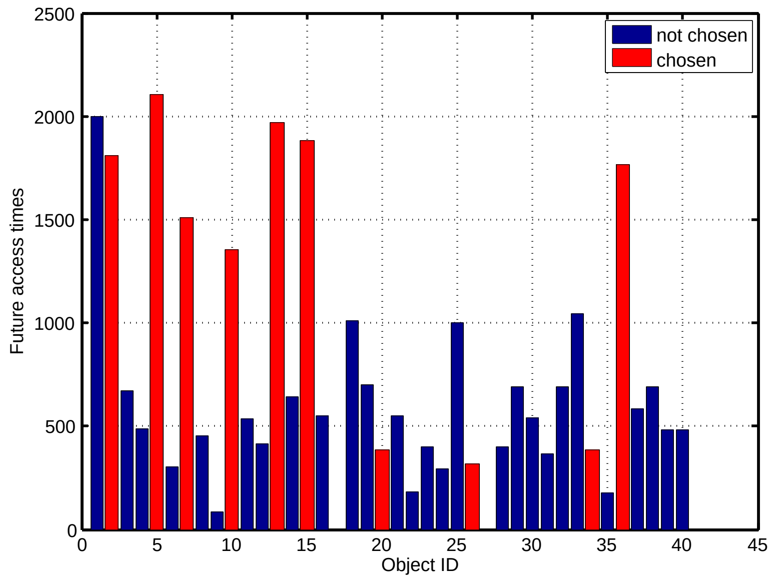


Figure 4.5: Future access times of selected data objects

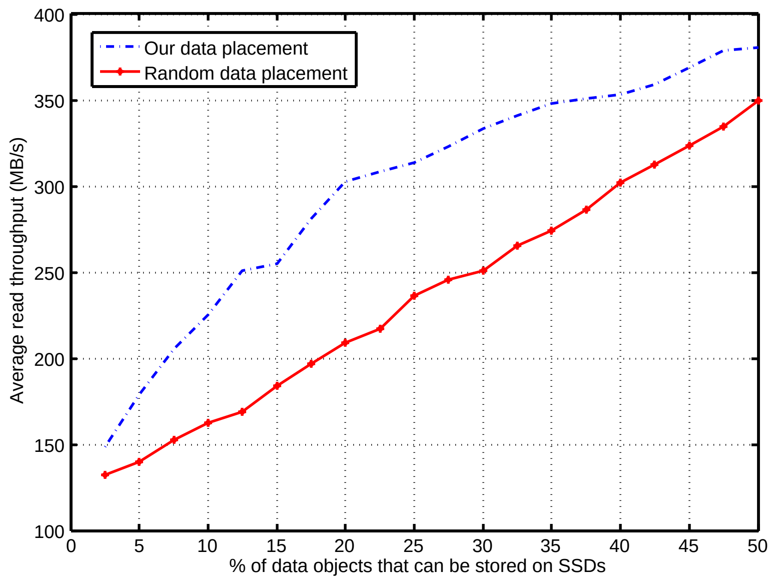


Figure 4.6: Our data placement v.s. random data placement

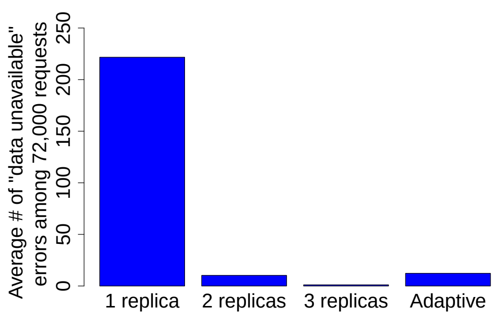
4.4.3 Evaluation on adaptive data replication

We evaluate the impact of our adaptive data placement on storage space efficiency and data availability through trace-driven event-based simulation. Both the failure logs and I/O traces used in our simulation were collected from deployed large-scale storage systems. Particularly, since we only have the hardware replacement logs for the Spider I file system, we do not use the Spider I as the simulated storage system in our evaluation since those software failures could also make data objects unaccessible. Instead, we choose NERSC file system NERSC (2016) as the simulated storage system, since it maintained both hardware and software failure logs for nearly 12 months during January to December in 2006 Petascale Data Storage Institute (2016).

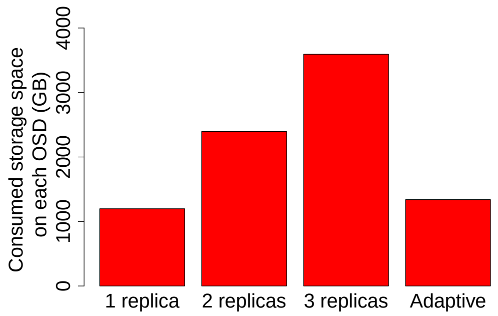
In 2006, NERSC file system had 24 I/O nodes which can be treated as the OSDs in our simulation. The storage capacity attached to each I/O node is 5TB and the annual failure rate of the entire system is around 350 failures per year. The average time spent on recovering the system from failures is around 100 minutes. In our simulation, we replay the LASR traces (in order to reduce time spent on running the simulation, we randomly select 72,000 read/write requests that occurred within 1 month time frame from the LASR traces) to generate the I/O workloads and use the failure logs collected from NERSC file system to trigger the failure events. Then we compare our adaptive data replication algorithm with the static data replication schemes in terms of storage space efficiency and data availability.

As shown in Figure 4.7(a), if constant number of replicas are created for each data object, with the increase of the number of replicas each data object has, the number of "data unavailable" errors reduces significantly. However, the storage space consumed by these data objects also increases linearly with the number of each data object's replicas. On the other hand, as shown in Figure 4.7(b), if our adaptive data replication algorithm is used, the storage system can achieve similar number of "data unavailable" errors to the scenario when two replicas are created for each data object, while the storage space consumed is similar to the scenario when each data object has only one replica. In other words, our algorithm can almost double the storage space efficiency without sacrificing the data availability.

Since our adaptive data replication algorithm needs to dynamically adjust the number of replicas for each data object, such adjustment could incur data movement overhead. For instance, if the optimization result indicates one of the data objects



(a) Data availability



(b) Storage space usage

Figure 4.7: Data availability and storage space efficiency achieved by different data replication schemes

should have three replicas while now it only has two, one more replica must be created and distributed in the storage system, which consumes the bandwidth of both the storage devices and the interconnect network.

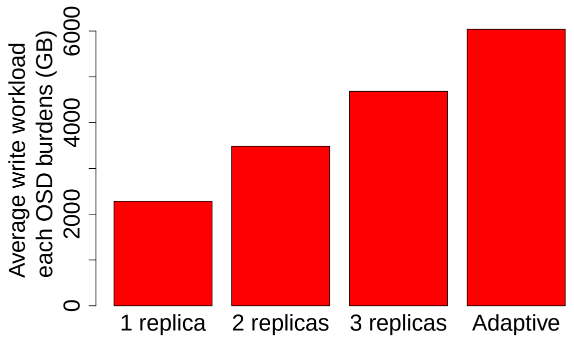


Figure 4.8: Average write workloads each OSD burdens with

In the simulation, our algorithm is triggered to re-optimize the data replication once an hour. In Figure 4.8, we compare the average write workloads each OSD burdens with when static and adaptive data replication are used. From this figure we can observe that using our adaptive data replication algorithm increases the write workloads each OSD handles, but we can control such overhead by reducing the frequency of triggering the optimization algorithm to save the bandwidth if necessary.

Chapter 5

Optimal Checkpoint Placement with Guaranteed Burst Buffer Endurance

5.1 The Overview

Large-scale high performance computing (HPC) systems usually support running tens of scientific simulations on hundreds of thousands of compute nodes simultaneously. Due to the scale of both hardware and software components involved, failures are common and a fact of life in large-scale HPC systems' daily operation. Most scalable scientific applications cope with potential failures using some form of defensive programming technique – by periodically exporting their execution state and intermediary results as a "checkpoint" to a persistent storage. In the event of failures, they will be able to continue the execution (restart) without repeating previous computation.

Checkpoints generated by scientific applications are written to the parallel file systems (PFS) which are usually built using traditional storage servers and spinning disk drivers for balanced cost, performance, and capacity. Parallel file systems provide an efficient data access mechanism between various computation resources over high-performance storage area networks. However, given the frequency of the checkpoints and the amount of data written at each checkpoint step, the total checkpoint size written in an application's runtime can be daunting. Trying to absorb such large-scale checkpoint I/O with traditional parallel file systems can be cost-prohibitive. On the other hand, studies have shown that PFS has been underutilized in the sense that it operates in much lower bandwidth spectrum most of the time which is nowhere near the peak Liu et al. (2012). In order to resolve the dichotomy, the concept of "burst buffer" was recently proposed and has been designed and prototyped in some large-scale HPC systems Liu et al. (2012); Sato et al. (2012); Bent et al. (2012); Xu et al. (2014); Sato et al. (2014). The basic idea behind the "burst buffer" is that we can build an intermediate hardware and I/O middleware layer between compute nodes and parallel file systems to better handle I/O workloads from scientific applications by utilizing flash-based storage devices, such as solid-state drive (SSD). The checkpoint data from scientific applications will be temporarily written into the burst buffer layer first and then drained to the underlying parallel file systems asynchronously. Since SSDs can provide much higher read and write bandwidth than regular hard disk drives, with the help of the burst buffer layer, the I/O performance of scientific applications will be improved significantly, which also means the checkpoints can be written and read faster and more CPU time can be saved for computation.

Ideally, the burst buffer was designed to absorb all I/O workloads generated by large-scale applications running on supercomputers. However, in reality we may have to limit the amount of data written to the burst buffer if the endurance requirements on SSD devices are to be considered. Specifically, each block in a SSD must be erased before being rewritten and only a finite number of erasures are possible before the bit error of SSD becomes unacceptably high. As an example, let us assume designing a burst buffer layer for a hypothetical large-scale HPC platform with tens of thousands of compute clients. If the building blocks are typical 256GB SSDs and if we are targeting a relatively moderate sized burst buffer layer (e.g. 5PB aggregate capacity), then the total number of SSDs required is about 20,000. According to the datasheet Samsung (2015), the newest Samsung 850 Pro SSD (256GB) has a warranty for maximum 150TB write. If the burst buffer is designed to serve 5 years, the maximum amount of data that can be written to the entire burst buffer per day is 1,600TB. We further assume that the write amplification factor is around 1.3 Hu et al. (2009), then the actual allowed write is about 1,200TB per day. On the other hand, some common large-scale scientific applications often produce huge checkpoint data. For example, the size of each checkpoint from CHIMERA application UCSF (2015) running on ORNL's Titan supercomputer OLCF (2012) is almost 160TB Tiwari et al. (2014). If several such scientific applications run simultaneously, the total size of the write workloads per day will be much larger than the SSD endurance requirements. Therefore, without constraints, the intensive write workloads produced by large-scale long-running scientific applications through checkpointing could degrade the endurance of SSD devices and the reliability of the burst buffer significantly.

Many techniques and approaches Yang and Ren (2011); Lee et al. (2012); Kaiser et al. (2013) have been proposed to optimize the endurance of SSD devices under different I/O workloads, particularly the kinds of workloads produced by personal computers, web servers, database systems, etc. Few of them tackles SSD endurance issues in HPC environment, because the HPC I/O workloads usually consist of extremely intensive write operations which can quickly wear out the SSD devices even when cutting-edge endurance optimization techniques are used. In fact, the HPC community does not have a full understanding in how to effectively maintain sustainable cost-to-performance and cost-to-capacity ratios for SSD devices under such write-heavy I/O workloads. One possible solution might be replacing the worn-out SSDs often to maintain a given capacity level, however, this solution is not feasible or cost-effective. The system-exclusive burst buffer can be built either by using node-local SSDs (i.e, an SSD device on every compute node) or can be shared (i.e, a set of pool of SSDs serving all compute nodes in a given HPC system). In the node-local case, the number of SSDs required grows linearly with the number of compute nodes. For the shared case, the required number of SSDs will grow linearly with the total memory size to absorb and flush the output data burst. In either case, we end up with thousand or tens of thousands of SSDs for a large-scale HPC system. To maintain the wear-out levels of this number of SSDs in a large-scale HPC facility will require extensive resources (i.e., man power to monitor and physically replace the worn-out devices on regular basis). Also, this approach will incur additional costs of the replaced devices. As an example, a modest size SSD can easily cost a few hundred U.S. dollars today and the replacing just the half of a 5,000 SSD population will amount to a few million U.S. dollars. Moreover, this approach requires compute node downtimes and interruptions to replace the worn-out devices from otherwise a "healthy" node (in terms of remaining components, such as CPU and memory), which is also an additional but hidden cost for the total cost of ownership (TCO) of a large-scale HPC system. For all these reasons combined, solely relaying on physically replacing worn-out SSDs to maintain a set of required capacity and endurance targets is not cost-effective and practical.

Besides frequently replacing worn-out SSD devices, another possible solution would be reducing the amount of data written to the burst buffer. In Fang and Chien (2015), the authors proposed a checkpoint interval optimization model for large-scale scientific applications which takes the constraint of burst buffer capacity into consideration. In such model, SSD-based burst buffers of supercomputers are used to absorb all checkpoint data of the scientific applications. Therefore, in order to satisfy the capacity constraint, the model intends to reduce the checkpointing frequency of some write-heavy jobs so that the amount of data written to the burst buffer can be reduced. However, a direct effect caused by such reduction in checkpointing frequency is that the potential wasted computation time due to system failures also increases significantly, especially for large computation jobs.

In order to solve the problems mentioned above, we propose a new checkpoint placement optimization model which collaboratively utilizes both the burst buffer and the parallel file system in a large-scale storage system to store the checkpoint data generated by scientific applications. Specifically, our model guarantees the endurance requirements of the SSD-based burst buffer layers without sacrificing too much of the computational efficiency. Moreover, in order to make the model feasible to real HPC systems, we also design an adaptive algorithm which can dynamically adjust the checkpoint placement based on the changing runtime characteristics of the HPC system and continuously optimize the usage of the burst buffer. The results from intensive evaluation demonstrate the effectiveness of our checkpoint placement model and our adaptive checkpoint placement algorithm. Particularly, using our adaptive checkpoint placement algorithm can guarantee the endurance of the burst buffer without degrading the performance of each job by more than 5%. Even better is that the degradation of the system computation efficiency is less than 3% if this adaptive algorithm is used.

5.2 Background

Before we discuss our checkpoint placement optimization model with guaranteed burst buffer endurance, we first briefly introduce existing studies on determining the optimal checkpoint interval and how to adjust the checkpoint intervals when different constraints are taken into consideration. The notations of all the symbols used in the following sections are listed in TABLE 5.1.

5.2.1 Optimal checkpoint interval

Most existing studies on finding optimal checkpoint interval model the execution of each large-scale scientific application as a sequence of activities alternating between the computation and checkpoint phases. Here we use $\Delta t_{cmpt,i}$ to denote the computation time period between two consecutive checkpoint activities of the i-th job, which is also referred to as the *checkpoint interval* of the i-th job. We use $\Delta t^{wr}_{ckpt,i}$

Table 5.1: Notations of symbols

M	Total number of scientific application jobs running on the HPC system
N	Total number of compute nodes in the HPC system
λ	Failure rate per compute node in the HPC system
l_{max}	Daily write limits of the burst buffer
n_i	Number of compute nodes assigned to execute the i-th job
$T_{cmpt,i}$	Total computation time required to finish the i-th job
$\Delta t_{cmpt,i}$	Checkpoint interval of the i-th job
S_i	Data size of one checkpoint from the i-th job
α_i	Percentage of checkpoints from the i-th job that should be written to the burst buffer
$thrpt_i^{wr,bb}$	Throughput when the i-th job writes checkpoints to the burst buffer
$thrpt_i^{wr,pfs}$	Throughput when the i-th job writes checkpoints to the parallel file systems
$\Delta t_{ckpt,i}^{wr,bb}$	Time required to write one checkpoint of the i-th job to the burst buffer
$\Delta t_{ckpt,i}^{wr,pfs}$	Time required to write one checkpoint of the i-th job to the parallel file systems

to denote the time spent on writing one checkpoint of the i-th job to the storage system.

Both hardware and software failures of compute nodes could interrupt the computation or checkpoint activities and trigger a restart phase (reading the latest checkpoint from the storage system to re-launch the job from the last correct state). As shown in Figure 5.1, If we assume ϵ percent of computation and checkpoint activities are wasted on average due to failures, we can then calculate the overhead caused by one failure of the i-th job as $\Delta t_{overhead,i} = \epsilon(\Delta t_{cmpt,i} + \Delta t_{ckpt,i}^{wr}) + \Delta t_{ckpt,i}^{rd}$, where $\Delta t_{ckpt,i}^{rd}$ is the time required to read one checkpoint of the i-th job from the storage system (usually $\Delta t_{ckpt,i}^{rd}$ can be ignored since $\Delta t_{ckpt,i}^{rd} \ll \Delta t_{cmpt,i} + \Delta t_{ckpt,i}^{wr}$).

Most existing studies assume the arrival of system failures follows a Poisson distribution and on average half of the work after last checkpoint is wasted if a failure occurs ($\epsilon = 0.5$) Young (1974); Vaidya (1997); Daly (2006); Fang and Chien

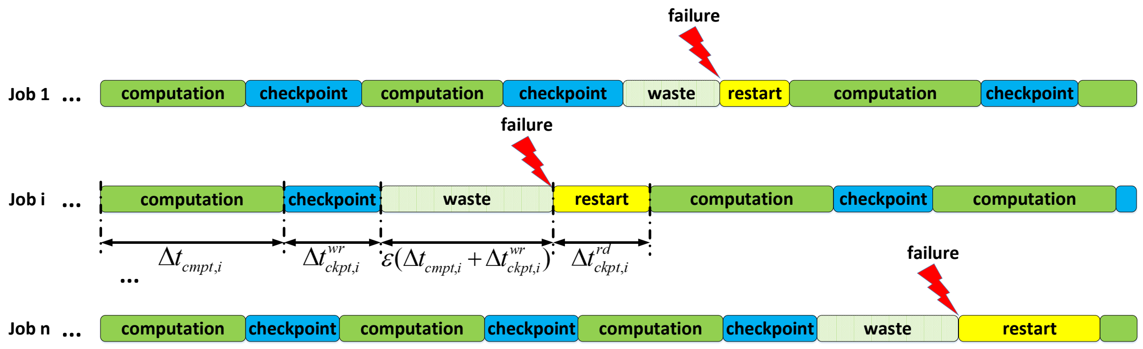


Figure 5.1: Checkpoint/restart for scientific applications running on HPC systems

(2015). If the failure rate per compute node in the HPC system is λ and the number of compute nodes occupied by the i-th job is n_i, the total execution time of the i-th job can be denoted as:

$$T_{total,i} = T_{cmpt,i}(1 + \frac{\Delta t^{wr}_{ckpt,i}}{\Delta t_{cmpt,i}} + \frac{1}{2}\lambda n_i \Delta t_{cmpt,i}), \tag{5.1}$$

where $T_{cmpt,i}$ is the total computation time required to finish the i-th job, $\frac{\Delta t^{wr}_{ckpt,i}}{\Delta t_{cmpt,i}} T_{cmpt,i}$ is the total time spent on checkpointing, and $\frac{1}{2}\lambda n_i \Delta t_{cmpt,i} T_{cmpt,i}$ is the total overhead caused by the failures. According to Young (1974); Vaidya (1997); Daly (2006), $T_{total,i}$ can be minimized when the following value is used as the checkpoint interval:

$$\Delta t_{cmpt,i} = \sqrt{\frac{2\Delta t^{wr}_{ckpt,i}}{\lambda n_i}} \tag{5.2}$$

5.2.2 Identifying checkpoint intervals by exploiting the temporal locality of failures

In Tiwari et al. (2014), the failure datasets collected from several launched HPC systems are analyzed. The analysis results show that the time between failures in HPC systems follows a Weibull distribution and the failure rate decreases over time since the last failure (and until the next failure). Therefore, Tiwari et al. (2014) proposes an adaptive checkpoint model called “lazy checkpointing”, which increases the checkpoint interval until the next failure. The incrementally increasing checkpoint interval is given by the following formula:

$$\Delta t_{cmpt,i}^{lazy} = \Delta t_{cmpt,i} (\frac{t}{\Delta t_{cmpt,i}})^{1-\beta}, \tag{5.3}$$

where β is the shape parameter of the Weibull distribution, t is the present time and $\Delta t_{cmpt,i}$ is the optimal checkpoint interval derived using formula 5.2. Once a failure happens, the checkpoint interval is reset to $\Delta t_{cmpt,i}$.

The basic idea behind the above formula is to increase the checkpoint interval over time so that it has the same slope as the curve of the failure rate. Since this "lazy checkpointing" model intends to skip some checkpoints, it can mitigate the I/O overhead caused by checkpointing and improve the efficiency of the HPC system compared to the static optimal checkpoint interval represented by 5.2.

5.2.3 Identifying checkpoint intervals for a fixed burst buffer capacity

For absorbing the output of large-scale scientific applications with high checkpoint frequencies and large data sizes in each step, a burst buffer may require tens of thousands of SSD devices. Since SSDs are more expensive than traditional spinning disk drives of the same capacity, system designers often try to limit the number of SSDs. Fang and Chien (2015) follows a similar logic and tries to determine a new optimal checkpoint interval under a fixed burst buffer capacity. Their model is:

$$\underset{\Delta t_{cmpt,i}}{\arg\min} \sum_{i=1}^{M} T_{total,i} \tag{5.4}$$

$$\text{s.t.} \sum_{i=1}^{M} \frac{T_{cmpt,i}}{\Delta t_{cmpt,i}} S_i \leq C_{max}, \tag{5.5}$$

where C_{max} is the total capacity of SSDs provisioned for the burst buffer to store the checkpoint data.

There are two disadvantages in this model. First, the objective function just simply sums up the total execution time of each job, which cannot accurately represent the total consumed computation resources. Instead, the execution time of each job should be weighted by the number of compute nodes each job occupies. Second, the above model intends to extend the checkpoint interval of each job, especially those with larger checkpoint data sizes or longer total computation times, to limit the amount of data written to the burst buffer, which significantly increases the potential wasted computation time caused by system failures.

In this work, we argue that the endurance is as critical as capacity in designing SSD-based burst buffers.

5.3 Checkpoint Placement Optimization with Guaranteed Burst Buffer Endurance

All existing studies assume that the burst buffer is used to absorb all checkpoint data from the HPC systems and only some of the checkpoints (e.g. every n-th checkpoint) are flushed from the burst buffer to the underlying parallel file systems for backup. In that case, if multiple jobs that all produce large amounts of checkpoint data execute concurrently, then it becomes very challenging to maintain the burst buffer endurance requirements without negatively affecting the computational efficiency. Therefore, in order to optimize the computational efficiency of large-scale scientific applications while guaranteeing the lifetime of the SSD-based burst buffer, we propose a new optimization model which collaboratively leverages both the burst buffer and parallel file systems to store checkpoint data. A major difference from existing approaches is that our proposed model can keep the original optimal checkpoint interval (to the best it can) and also reduce the potential wasted computation time caused by system failures without exceeding the write limit of the burst buffer.

We denote the time consumed by writing one checkpoint of the i-th job to the burst buffer as $\Delta t_{ckpt,i}^{wr,bb}$ and that to the parallel file systems as $\Delta t_{ckpt,i}^{wr,pfs}$. If the write throughput of the i-th job to the burst buffer and parallel file systems are given, we can calculate $\Delta t_{ckpt,i}^{wr,bb}$ and $\Delta t_{ckpt,i}^{wr,pfs}$ as:

$$\Delta t_{ckpt,i}^{wr,bb} = \frac{S_i}{thrpt_i^{wr,bb}}, \tag{5.6}$$

$$\Delta t_{ckpt,i}^{wr,pfs} = \frac{S_i}{thrpt_i^{wr,pfs}} \tag{5.7}$$

The challenge is to determine the optimal percentage of the checkpoints that should be written to the burst buffer per application. If we use α_i to denote the percentage of checkpoints from the i-th job which should be written to the burst buffer, then the average time spent on writing one checkpoint of the i-th job, $\Delta t_{ckpt,i}^{wr}$, can be calculated as:

$$\Delta t_{ckpt,i}^{wr} = \alpha_i \Delta t_{ckpt,i}^{wr,bb} + (1-\alpha_i)\Delta t_{ckpt,i}^{wr,pfs} \tag{5.8}$$

If we replace $\Delta t_{cmpt,i}$ with $\sqrt{\frac{2\Delta t_{ckpt,i}^{wr}}{\lambda n_i}}$ in (5.1), we can obtain the total execution time of the i-th job as:

$$\begin{aligned}
T_{total,i} =& T_{cmpt,i}(1 + \frac{\Delta t_{ckpt,i}^{wr}}{\Delta t_{cmpt,i}} + \frac{1}{2}\lambda n_i \Delta t_{cmpt,i}) \\
=& T_{cmpt,i}(1 + \frac{\Delta t_{ckpt,i}^{wr}}{\sqrt{\frac{2\Delta t_{ckpt,i}^{wr}}{\lambda n_i}}} + \frac{1}{2}\lambda n_i \sqrt{\frac{2\Delta t_{ckpt,i}^{wr}}{\lambda n_i}}) \\
=& T_{cmpt,i}(1 + \sqrt{2\lambda n_i \Delta t_{ckpt,i}^{wr}}) \\
=& T_{cmpt,i}\{1 + \sqrt{2\lambda n_i[\alpha_i \Delta t_{ckpt,i}^{wr,bb} + (1-\alpha_i)\Delta t_{ckpt,i}^{wr,pfs}]}\}
\end{aligned} \tag{5.9}$$

Now we can formulate the objective function of our optimization model as follows

$$\arg\min_{\alpha_i} \sum_{i=1}^{M} n_i T_{cmpt,i}\{1 + \sqrt{2\lambda n_i[\alpha_i \Delta t_{ckpt,i}^{wr,bb} + (1-\alpha_i)\Delta t_{ckpt,i}^{wr,pfs}]}\}, \tag{5.10}$$

in which each job's total execution time is weighted by the number of compute nodes each job occupies.

The total size of the checkpoint data written to the burst buffer can be calculated as

$$\sum_{i=1}^{M} \frac{T_{cmpt,i}}{\sqrt{\frac{2[\alpha_i \Delta t_{ckpt,i}^{wr,bb} + (1-\alpha_i)\Delta t_{ckpt,i}^{wr,pfs}]}{\lambda n_i}}} \alpha_i S_i \tag{5.11}$$

Since vendors provide the daily write limits of the SSD devices, we denote the daily write limit of the entire SSD-based burst buffer as l_{max}. After we divide (5.11) by $T_{cmpt,i}$, we can obtain the average checkpoint data written per hour, which should not exceed the per-hour write limit of the burst buffer, as the constraint of our model:

$$\sum_{i=1}^{M} \frac{\alpha_i S_i}{\sqrt{\frac{2[\alpha_i \Delta t_{ckpt,i}^{wr,bb} + (1-\alpha_i)\Delta t_{ckpt,i}^{wr,pfs}]}{\lambda n_i}}} \leq l_{max}/24 \tag{5.12}$$

Putting them all together, we establish the following optimization model to determine the optimal percentage of checkpoints that should be written to the burst buffer for each scientific application.

$$\begin{aligned} \arg\min_{\alpha_i} & \sum_{i=1}^{M} n_i T_{cmpt,i} \{1 + \sqrt{2\lambda n_i [\alpha_i \Delta t_{ckpt,i}^{wr,bb} + (1-\alpha_i)\Delta t_{ckpt,i}^{wr,pfs}]}\} \\ \text{s.t.} & \sum_{i=1}^{M} \frac{\alpha_i S_i}{\sqrt{\frac{2[\alpha_i \Delta t_{ckpt,i}^{wr,bb} + (1-\alpha_i)\Delta t_{ckpt,i}^{wr,pfs}]}{\lambda n_i}}} \leq l_{max}/24 \end{aligned} \tag{5.13}$$

Apparently, our optimization model is a nonlinear programming model. To make it more easier to resolve, we substitute $\sqrt{2\lambda n_i [\alpha_i \Delta t_{ckpt,i}^{wr,bb} + (1-\alpha_i)\Delta t_{ckpt,i}^{wr,pfs}]}$ with x_i in both the objective function and constraint to transform the model into the following form:

$$
\begin{aligned}
&\arg\min_{\alpha_i} \sum_{i=1}^{M} n_i T_{cmpt,i}(1+x_i) \\
&\text{s.t.} \sum_{i=1}^{M} \frac{\frac{1}{2}x_i - \frac{\lambda n_i t_{ckpt,i}^{wr,pfs}}{x_i}}{t_{ckpt,i}^{wr,bb} - t_{ckpt,i}^{wr,pfs}} S_i \leq l_{max}/24
\end{aligned}
\tag{5.14}
$$

We can solve the this constrained nonlinear programming problem by using the interior-point algorithm Wächter and Biegler (2006).

After we obtain the optimal value for x_i, thereby α_i, according to formula (5.2), we can also calculate the new optimal checkpoint interval of the i-th job as

$$
\Delta t_{cmpt,i} = \sqrt{\frac{2(\alpha_i \Delta t_{ckpt,i}^{wr,bb} + (1-\alpha_i)\Delta t_{ckpt,i}^{wr,pfs})}{\lambda n_i}}
\tag{5.15}
$$

By using both the optimal α_i and $\Delta t_{cmpt,i}$ for all scientific applications, we can maximize the computation efficiency of an HPC system while guaranteeing the SSD endurance requirements for the burst buffer.

5.4 Adaptive Checkpoint Placement for Optimal HPC System and Burst Buffer Usage

Assuming runtime characteristics, such as failure rates, job size, checkpoint size, of scientific applications are given, we can solve the above optimization model and determine the checkpoint interval and the percentage of checkpoint data that should be stored on the burst buffer on a per job basis. However, in practice, some of these runtime characteristics cannot be obtained before execution and others vary with time. Therefore, we'd like to design an adaptive algorithm which can dynamically adjust the checkpoint placement based on those time-dependent characteristics and continuously optimize the usage of the burst buffer.

5.4.1 Runtime characteristics of HPC systems and scientific applications

First, let us categorize the runtime characteristics of HPC systems and scientific applications that are required for determining the optimal checkpoint placement and identify the ones varying during the application execution.

- **Checkpoint size:** Although different scientific applications write checkpoints of different sizes, for a specific scientific application the size of each checkpoint is usually constant.
- **Job size:** Job size means the number of compute nodes each computation job occupies. Job size not only determines the aggregate I/O write and read bandwidths, but also effects the failure rate of each running scientific application. The job size of each application is usually determined before the execution and does not change if the computation continues normally. However, when the job is restarting from a failure, the job size might be changed as it depends on how the job scheduler allocates the compute nodes to restart the job.
- **Aggregate I/O bandwidth:** The aggregate I/O bandwidth of each job can achieve when writing checkpoints to either the burst buffer or the parallel file system are determined by the job size. Jobs with larger sizes often have higher aggregate I/O bandwidth. If the job size remains the same, the aggregate I/O bandwidth for each job will not change.
- **Total computation time:** Total computation time of each scientific application is the time required to finish all computation tasks, not including that spent on writing checkpoints or recovering from failures. It depends on the complexity of the job and the job size. If the job size does not change, this value for each scientific application will be constant.

- **Per-node failure rate:** Per-node failure rates of HPC systems are often estimated using historical failure logs. Though constant failure rates are often assumed in most of existing models, several studies have shown that the failure rates of compute nodes in large-scale HPC systems might vary with time Schroeder and Gibson (2006); Tiwari et al. (2014); El-Sayed and Schroeder (2014). For instance, Tiwari et al. (2014) presents the analysis of failure data collected from multiple supercomputing facilities including Oak Ridge Leadership Computing Facility (OLCF) and the Los Alamos National Laboratory (LANL). The results indicate that there is a strong temporal locality between compute node failures in HPC systems.

5.4.2 Effect of dynamic runtime characteristics on checkpoint placement optimization

From the above analysis, we realize that except the checkpoint size, all other runtime characteristics might change during the execution. Next, we need to study how the variation in these runtime characteristics effects the optimization results of the checkpoint placement.

When job size is changed

The job size could be changed when the job is restarting from a failure. For example, on Titan supercomputer, when a job is terminated by a failure, the system might try to restart it by resubmitting it to the job scheduler. Then the job scheduler will re-launch the job on currently available compute nodes, which usually are not the nodes used by the job before the failure. If the available compute nodes are less than those used before the failure, the size of the re-launched job has to be downgraded.

In the optimization model given by formula 5.13, the size of the i-th job is denoted by n_i. If n_i decreases, the aggregate failure rate of the i-th job, λn_i, will decrease. Meanwhile, the decrease of n_i also reduces the aggregate I/O bandwidth the i-th job can achieve, and thereby increases the average time spent on writing checkpoints to the storage, $[\alpha_i \Delta t_{ckpt,i}^{wr,bb} + (1 - \alpha_i)\Delta t_{ckpt,i}^{wr,pfs}]$. Therefore, if n_i decreases, according to the constraint of the optimization model, the checkpoints written to the burst buffer will decrease. In that case, if we do not write more percentage of checkpoints to the burst buffer, the burst buffer will be underutilized.

When an existing job finishes or a new job joins

If an existing job finishes, all the computing resources it occupies will be released. It does not need the burst buffer to store its checkpoints anymore. In that case, the checkpoint placement should be re-optimized and the new percentage of checkpoint data that should be stored on the burst buffer as well as the new checkpoint interval should be calculated for those remaining jobs. Basically, the remaining jobs will be allocated more burst buffer write permit since the total write workload to the burst buffer has decreased.

Similarly, when a new job joins, the optimization results on checkpoint placement should be calculated again and some burst buffer write permit should be allocated to the new job accordingly.

When failure rate is time-dependent

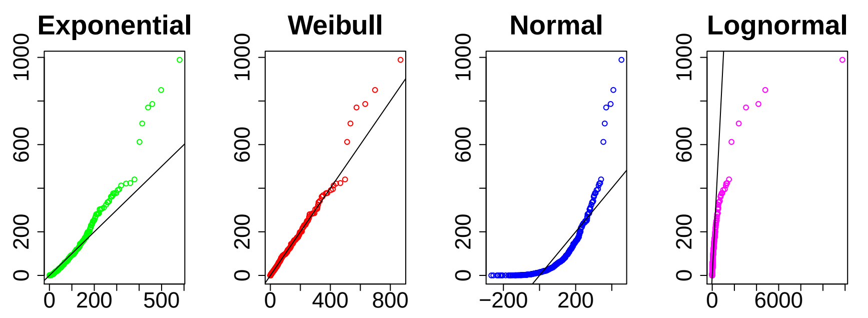
Time-dependent failure rates have been observed among compute nodes in HPC systems of different sizes. For example, we analyze the failure logs collected from three HPC computing clusters constructed with different number of compute nodes and use four different distributions to fit the time between failures (in hours) extracted from the failure logs. The fitting results are shown in TABLE 5.2, which lists the parameters of each distribution estimated by using Maximum Likelihood Estimation. In order to illustrate the fitness of these four different distributions, we present the Q-Q plot for each distribution in Figure 5.2. Specifically, each Q-Q plot compares the quantiles drawn from the failure data (y-axis) to theoretical quantiles calculated from a particular distribution using parameters listed in TABLE 5.2 (x-axis). If the quantiles of the failure data came from the same distribution, the points in the Q-Q plot will approximately lie on the line $y = x$. As shown in Figure 5.2, for each of these three HPC systems, the Weibull distribution fits the failure data best.

Table 5.2: Distributions fitted to the failure data

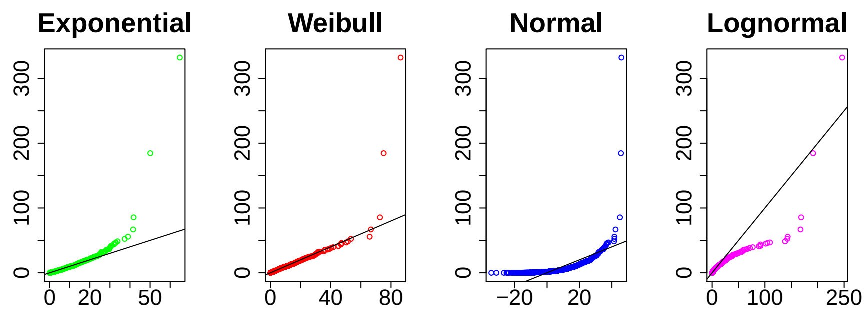
System	Distribution			
	Exponential	Weibull	Normal	Lognormal
LANL System 8 (164 nodes)	rate=0.0119	shape=0.7111, scale=67.375	mean= 84.079, sd=122.00	meanlog=3.3992, sdlog=1.7931
LANL System 18 (1024 nodes)	rate=0.1336	shape=0.8170, scale=6.6293	mean=7.4829, sd=12.806	meanlog=1.2194, sdlog=1.4505
OLCF Titan (18,688 nodes)	rate=0.1378	shape=0.6885 scale=5.4527	mean=7.2565 sd=12.731	meanlog=0.9197 sdlog=1.5817

For each of these three HPC systems given in TABLE 5.2, the shape parameter of the Weibull distribution is less than 1, indicating a decreasing failure rate. This means the expected remaining time until the next failure increases with the time since the last failure has occurred, or in other words, the next failure is more likely to happen within a relatively short time period after the last failure. For example, as shown in Figure 5.3, if failure 2 is the next failure after the failure 1, then it is more likely to occur very soon after failure 1 because of the decreasing failure rate.

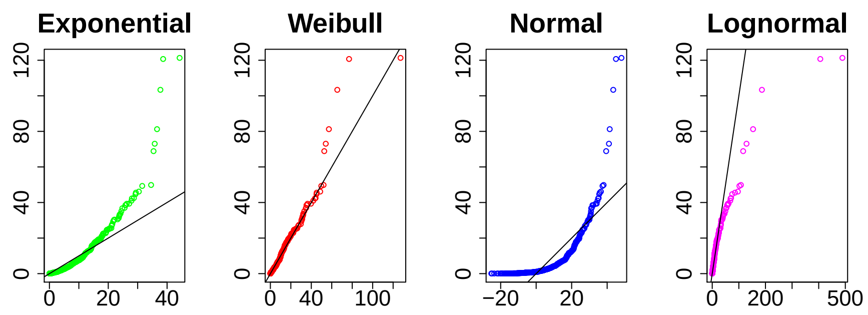
Ideally, the most efficient way to utilize the burst buffer is limiting the write workloads to extend the SSDs' lifetime while maximizing the read workloads to reduce the job restarting time by leveraging the high read throughput of the burst buffer. However, if the static checkpoint placement is used, as shown in Figure 5.3, the checkpoints are written with constant frequency and a fixed percentage of them are stored on the burst buffer. In this case, the probability that the job uses a checkpoint stored on the burst buffer to recover from the failure is low since the failures are not uniformly distributed due to the decreasing failure rate. Therefore, in order to utilize the HPC system and the burst buffer more efficiently, the checkpoint placement algorithm must also be able to adapt to the time-dependent failure rate. For instance, as shown in Figure 5.3, the adaptive checkpoint placement should be able to adjust



(a) LANL System 8 (164 nodes, small scale)



(b) LANL System 18 (1024 nodes, medium scale)



(c) OLCF Titan (18,688 nodes, large scale)

Figure 5.2: Q-Q plot for visualizing fitness of different distributions

the percentage of checkpoints written to the burst buffer after failure 1, so that the likelihood of restarting the job using checkpoints from the burst buffer when failure 2 happens can be increased.

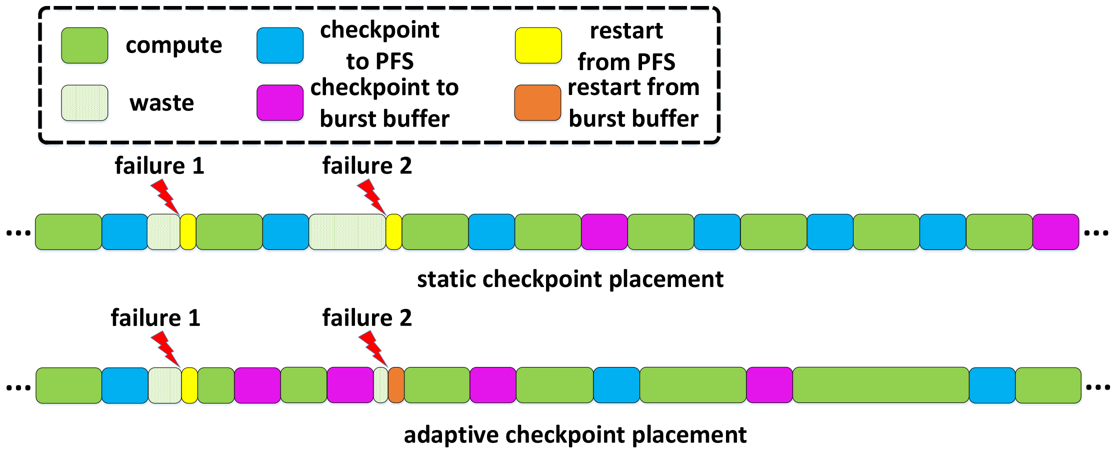

compute
checkpoint to PFS
restart from PFS
waste
checkpoint to burst buffer
restart from burst buffer
failure 1
failure 2
static checkpoint placement
failure 1
failure 2
adaptive checkpoint placement

Figure 5.3: Static/adaptive checkpoint placement

5.4.3 Adaptive checkpoint placement optimization algorithm

The design of our adaptive checkpoint placement algorithm is based on two assumptions. First, in HPC systems, it is possible to obtain all running jobs' runtime characteristics (such as the job size, the remaining computation time, failure rates, etc.), which will be fed to our optimization algorithm as the input parameters. Second, there is a way to apply the optimization results given by our algorithm to adjusting the checkpoint interval of each scientific computation job. In practice, the first assumption is often valid as most of the commonly used workload manager softwares are able to collect jobs' runtime information during operation. The second assumption is also possible as some workload managers, such as SLURM SchedMD (2015), can utilize interfaces provided by checkpoint/restart libraries, such as BLCR Laboratory (2015), to configure the checkpoint interval of each job. Therefore, it is possible to integrate our algorithm into the workload managers to achieve adaptive checkpoint placement optimization.

In order to make the checkpoint placement adaptive to those changing runtime characteristics, the previous model designed for the static checkpoint placement optimization needs to be modified. First, given a fixed optimization period, Δ_{opt}, the objective function of the model, as shown bellow, is to minimize the overhead, including the time spent on writing, reading checkpoints and the wasted computation time due to failures, during such time period.

$$\begin{aligned}\arg\min_{\alpha_i}\sum_{i=1}^{M} n_i\{\frac{\Delta_{opt}}{\Delta t_{cmpt,i}}[\alpha_i\Delta t_{ckpt,i}^{wr,bb}+(1-\alpha_i)\Delta t_{ckpt,i}^{wr,pfs}]+\\ \frac{n_i}{N}N_f[\alpha_i\Delta t_{ckpt,i}^{rd,bb}+(1-\alpha_i)\Delta t_{ckpt,i}^{rd,pfs}]+\frac{n_i}{N}N_f\frac{\Delta t_{cmpt,i}}{2}\},\end{aligned} \tag{5.16}$$

where N_f is the expected number of node failures during Δ_{opt}. If we assume the current time is t_{now} and the last failure happened at t_{fail}, N_f can be calculated as $N_f = \int_{t_{now}-t_{fail}}^{t_{now}+\Delta t_{opt}-t_{fail}} h(t)dt$, where $h(t)$ is the hazard rate function estimated by using the failure logs collected from the Titan supercomputer ($h(t)=\frac{\beta}{\eta}(\frac{t}{\eta})^{\beta-1}$, where β is the shape parameter and η is the scale parameter of the Weibull distribution). Apparently, the above objective function achieves the minimum when $\Delta t_{cmpt,i} = \sqrt{\frac{\Delta_{opt}[\alpha_i\Delta t_{ckpt,i}^{wr,bb}+(1-\alpha_i)\Delta t_{ckpt,i}^{wr,pfs}]}{n_iN_f/(2N)}}$, and after we replace the $\Delta t_{cmpt,i}$ with this value, the objective function can be simplified as:

$$\begin{aligned}\arg\min_{\alpha_i}\sum_{i=1}^{M} n_i\{\sqrt{2\frac{n_i}{N}N_f\Delta_{opt}[\alpha_i\Delta t_{ckpt,i}^{wr,bb}+(1-\alpha_i)\Delta t_{ckpt,i}^{wr,pfs}]}\\ +\frac{n_i}{N}N_f[\alpha_i\Delta t_{ckpt,i}^{rd,bb}+(1-\alpha_i)\Delta t_{ckpt,i}^{rd,pfs}]\}\end{aligned} \tag{5.17}$$

Furthermore, the constraint on burst buffer write limit can be denoted as follows:

$$\sum_{i=1}^{M}\frac{\alpha_iS_i}{\sqrt{\frac{\Delta_{opt}[\alpha_i\Delta t_{ckpt,i}^{wr,bb}+(1-\alpha_i)\Delta t_{ckpt,i}^{wr,pfs}]}{n_iN_f/(2N)}}} \le l_{max}/24 \tag{5.18}$$

As illustrated in Algorithm 3, our algorithm will be called by the workload manager to optimize the checkpoint placement at the beginning of every Δt_{opt}. Specifically, our algorithm first obtains the runtime characteristics of each running job from the workload manager, including the checkpoint size and job size. Then based on the job size, it estimates the I/O throughput each job can achieve when writing the checkpoints to or reading the checkpoints from the storage. For example,

Algorithm 3 Adaptive Checkpoint Placement Optimization

```
1: Obtain runtime characteristics from the workload manager:
   checkpoint sizes $[S_1, S_2 \dots, S_M]$,
   job sizes $[n_1, n_2 \dots, n_M]$
2: Estimate I/O throughput of each job:
   burst buffer: $[thrpt_1^{wr,bb}, \dots, thrpt_M^{wr,bb}]$,$[thrpt_1^{rd,bb}, \dots, thrpt_M^{rd,bb}]$
   PFS: $[thrpt_1^{wr,pfs}, \dots, thrpt_M^{wr,pfs}]$,$[thrpt_1^{rd,pfs}, \dots, thrpt_M^{rd,pfs}]$
3: Calculate time spent on writing/reading one checkpoint for each job:
   burst buffer: $[\Delta t_{ckpt,1}^{wr,bb}, \dots, \Delta t_{ckpt,M}^{wr,bb}]$,$[\Delta t_{ckpt,1}^{rd,bb}, \dots, \Delta t_{ckpt,M}^{rd,bb}]$
   PFS: $[\Delta t_{ckpt,1}^{wr,pfs}, \dots, \Delta t_{ckpt,M}^{wr,pfs}]$,$[\Delta t_{ckpt,1}^{rd,pfs}, \dots, \Delta t_{ckpt,M}^{rd,pfs}]$
4: Estimate the number of node failures during $\Delta t_{opt}$:
   $N_f = \int_{t_{now}-t_{fail}}^{t_{now}+\Delta t_{opt}-t_{fail}} h(t)dt$
5: Resolve the optimization model:
   $$\arg\min_{\alpha_i} \sum_{i=1}^{M} n_i \{ \sqrt{2\frac{n_i}{N} N_f \Delta_{opt}[\alpha_i \Delta t_{ckpt,i}^{wr,bb} + (1-\alpha_i)\Delta t_{ckpt,i}^{wr,pfs}]} + \frac{n_i}{N} N_f [\alpha_i \Delta t_{ckpt,i}^{rd,bb} + (1-\alpha_i)\Delta t_{ckpt,i}^{rd,pfs}] \}$$
   $$\text{s.t.} \sum_{i=1}^{M} \frac{\alpha_i S_i}{\sqrt{\frac{\Delta_{opt}[\alpha_i \Delta t_{ckpt,i}^{wr,bb} + (1-\alpha_i)\Delta t_{ckpt,i}^{wr,pfs}]}{n_i N_f/(2N)}}} \leq l_{max}/24$$
6: Calculate new checkpoint intervals for each job:
   $\Delta t_{cmpt,i} = \sqrt{\frac{\Delta_{opt}[\alpha_i \Delta t_{ckpt,i}^{wr,bb} + (1-\alpha_i)\Delta t_{ckpt,i}^{wr,pfs}]}{n_i N_f/(2N)}}$
7: Apply new checkpoint intervals and $[\alpha_1, \alpha_2, \dots, \alpha_M]$ to the workload manager to adjust
   runtime characteristics of each job
8: Wait until $t_{now} + \Delta t_{opt}$, go to step 1
```

the burst buffer layers in HPC systems are usually built upon SSD devices locally attached to each compute node. Therefore, the aggregate I/O throughput the job can achieve is roughly estimated as the product of the job size and each SSD's I/O bandwidth. For the parallel file system, such as Spider II Oral et al. (2013), when the job size is below some threshold, the aggregate I/O throughput is also proportional to the job size, but once the job size is larger than such threshold, the I/O bandwidth of the parallel file system will be saturated Oral et al. (2014). After estimating the I/O throughput, our algorithm calculates the time spent on writing and reading one checkpoint for each job.

The most challenging part in our algorithm is estimating the number of node failures in the HPC system. Most large-scale HPC systems maintain failure logs during their operation Tiwari et al. (2014). By analyzing the failure logs, we can obtain the failure characteristics of compute nodes in the system. For example, as we mentioned in previous paragraphs, the analysis of failure logs demonstrates a Weibull distribution with decreasing failure rate fits the time between failures of Titan supercomputer the best. By estimating the parameters of such Weibull distribution, we can also obtain the hazard rate function, $h(t)$.

Finally, with all the runtime characteristics that have been obtained, estimated or calculated, our algorithm resolves the optimization model to get new percentage of checkpoints that should be written to the burst buffer as well as the new checkpoint interval for each job. These optimization results will be applied to all jobs through the workload manager to guide the checkpoint placement in the coming period.

5.5 Evaluation

In this section, we evaluate our adaptive checkpoint placement algorithm through event-based simulation and the failure traces collected from Titan supercomputer are used to drive the simulation.

5.5.1 Evaluation setup

Our simulation parameter settings are based on system configuration settings of deployed leadership computing facilities and runtime characteristics obtained from real scientific application jobs.

System features of leadership computing facilities

- **Compute platform:** Our models and algorithms are primarily evaluated based on the architecture and characteristics of Titan, the second fastest supercomputer in the world, which is deployed and managed by OLCF. Titan is composed of 18,688 compute nodes and has more than 700TB memory capacity. According to the failure data collected by OLCF, the *mean time between failure* (MTBF) of the entire Titan supercomputer is about 11 hours and the time between node failures satisfies a Weibull distribution (See TABLE 5.2). In our evaluation, we use the Titan failure data to drive the compute node failures in our simulation.

- **Burst buffer:** Since Titan does not have a burst buffer layer *per se*, we apply system characteristics of Gordon supercomputer SDSC (2015), one of the first supercomputers utilizing SSDs, to our evaluation.
  - *SSD model:* The solid state drives used to equip Gordon supercomputer are Intel 710 (300GB) SSDs, which are relatively expensive compare to other manufactures' products. For example, each Samsung 850 Pro (256GB) SSD Samsung (2015) costs \$120, only one forth of the list price of the Intel 710 (300GB) SSD. Since Titan has much more compute nodes than Gordon, we choose Samsung 850 Pro (256GB) SSDs to build a more cost-effective burst buffer for Titan in our evaluation, even though some expensive SSDs might provide better performance and longer endurance.
  - *Capacity:* The burst buffer layer of Gordon consists of 64 I/O nodes. Each of these I/O nodes contains 16 SSDs, and serves 16 compute nodes. In our evaluation, we also assume that averagely each compute node of Titan can be served by one SSD, no matter such SSD is integrated into the I/O node or locally attached to the compute node. Therefore, the total capacity of the burst buffer in our evaluation is about $256 \times 18688 = 4.6$PB.
  - *Bandwidth:* As the 64 I/O nodes (the burst buffer layer) of Gordon system can provide 320 GB/s aggregate write bandwidth, each SSD can roughly provide 320 MB/s write bandwidth. Let us assume each compute node of Titan supercomputer can also enjoy a similar write bandwidth. Since

Titan has 18,688 compute nodes in total, the aggregate write bandwidth of the burst buffer will be around $320 \times 18688/1024 = 5,840$ GB/s.

  – *Write limit:* If each compute node is served by a dedicated SSD, then the total number of SSDs in Titan's burst buffer is 18,688. Since each Samsung 850 Pro SSD has a warranty for maximum 150TB write, if the burst buffer is designed to be in operation for at least 5 years, the maximum amount of data can be written to the burst buffer per day is around $(150 \times 18688)/(5 \times 365) = 1,536$TB. If the write amplification factor is about 1.3, then the actual write limit should be $1536/1.3 = 1,100$TB per day. In all evaluation cases, we set the daily write limit of the burst buffer as $1,000$TB.

- **Parallel file system:** Spider II file system Oral et al. (2013) is a Lustre-based parallel file system used by Titan supercomputer. It consists of 20,106 hard disk drives and provides 32PB capacity (after RAID). The aggregate I/O bandwidth of Spider II file system depends on the number of clients issuing the I/O operations concurrently. According to measurement results provided by OLCF Oral et al. (2014), the aggregate write bandwidth of Spider II can linearly increase up to 300GB/s when the number of clients is less than 6,000. Once more than 6,000 clients are writing data to Spider II concurrently, the I/O bandwidth gets saturated.

Runtime characteristics of scientific applications

- **Checkpoint size:** Different scientific applications usually write checkpoints of different sizes and the difference could be huge. For example, CHIMERA and VULCAN are two scientific applications running on Titan supercomputer. Each checkpoint of CHIMERA is 160TB while that of VULCAN is only 0.83GB.

- **Job size:** Job size means the number of compute nodes each computation job occupies. Job size not only determines the aggregate I/O bandwidth of writing and reading checkpoints, but also effects the failure rate of each running

Table 5.3: Runtime characteristics of common scientific applications running on Titan supercomputer

Domain	Application	Checkpoint size	Job size	Computation time
Astrophysics	CHIMERA	160TB	9216	360Hours
Astrophysics	VULCAN	0.83GB	256	720Hours
Climate	POP	26GB	512	480Hours
Combustion	S3D	5TB	2048	240Hours
Fusion	GTC	20TB	6144	120Hours
Fusion	GYRO	50GB	512	120Hours

scientific application. In our simulation, we vary the job size from 256 to 9216 to cover the wide range of job characteristics in real HPC environment.

- **Computation time:** Computation time of each scientific application is the time required to finish all computation task, not including that spent on writing checkpoints or recovering from failures. It depends on the complexity of the job and number of compute nodes used by the job. In our simulation, the computation time of simulated scientific applications varies from 120 to 720 hours.

TABLE 5.3 lists all runtime characteristics of several common scientific applications running on Titan supercomputer. We apply these characteristics to our evaluation and simulate the computation/checkpointing activities of these scientific applications when they are running concurrently on Titan supercomputer.

Evaluated models and algorithms

In order to demonstrate the effectiveness of our adaptive checkpoint placement algorithm, we also evaluate several other models and algorithms for comparison. The abbreviation and descriptions of all models and algorithms we evaluate are listed in TABLE 5.4.

In order to have a fair comparison, the system features of the leadership computing facility and the runtime characteristics of those scientific computation jobs are kept the same when evaluating each of these models and algorithms.

Table 5.4: Evaluated models and algorithms

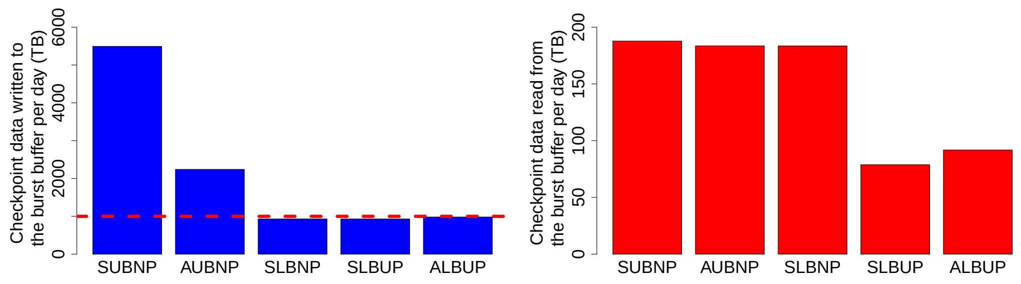
Abbreviation	Description of the model or algorithm
Static-**U**nlim-**B**B-**N**o-**P**FS (**SUBNP**)	Static optimal checkpoint interval given by formula 5.2 is used. All checkpoints are written to the burst buffer without limit. No checkpoint is written to the PFS directly.
Adapt-**U**nlim-**B**B-**N**o-**P**FS (**AUBNP**)	Adaptive checkpoint model proposed by Tiwari et al. (2014) is used. All checkpoints are written to the burst buffer without limit. No checkpoint is written to the PFS directly.
Static-**L**im-**B**B-**N**o-**P**FS (**SLBNP**)	Static checkpoint interval from the model proposed by Fang and Chien (2015) is used. Limited checkpoints are written to the burst buffer. No checkpoint is written to the PFS directly.
Static-**L**im-**B**B-**U**nlim-**P**FS (**SLBUP**)	Our static checkpoint placement model with guaranteed burst buffer endurance (see formula 5.13) is used. Limited checkpoints are written to the burst buffer. Unlimited checkpoints can be written to the PFS directly.
Adapt-**L**im-**B**B-**U**nlim-**P**FS (**ALBUP**)	Our adaptive checkpoint placement algorithm is used. Limited checkpoints are written to the burst buffer. Unlimited checkpoints can be written to the PFS directly.

5.5.2 Evaluation results

I/O workloads the burst buffer burdens with

We first evaluate the average I/O workloads the burst buffer burdens with when different checkpoint placement models or algorithms are used by the scientific computation jobs. Specifically, in the evaluation, we simulate the checkpoint/restart activities of all jobs listed in TABLE 5.3 during 100-hour operation of Titan and calculate the average amount of checkpoint data written to and read from the burst buffer. During this 100-hour operation, except the per-node failure rate, all other runtime characteristics, such as the number of jobs, the size of each job, etc., do not vary with time. The evaluation result is shown in Figure 5.4.

Model SUBNP means all checkpoints from all jobs running on Titan are written to the burst buffer directly without limit and formula 5.2 is adopted to determine the optimal checkpoint interval for each job, as shown in Figure 5.4(a), the amount of checkpoint data written to the burst buffer per day could be more than 5,500TB. If the burst buffer is built with Samsung 850 Pro SSDs, under such intensive write workload, the lifetime of the burst buffer might be only $(150 \times 18688)/(5500 \times 365) = 1.4$ years. Even if model AUBNP is used, which can dynamically adjust the checkpoint



(a) Write workloads

(b) Read workloads

Figure 5.4: Average amount of checkpoint data written to and read from the burst buffer per day when different checkpoint placement models or algorithms are used

interval according to the changing failure rate to reduce the write workloads, as long as all checkpoints are directly absorbed by the burst buffer, the data written to the burst buffer per day is still about 2,200TB, much more than that is allowed if the burst buffer is expected to serve more than 5 years. Therefore, in order to prolong the lifetime of the burst buffer, write workloads issued to the burst buffer must be limited. For example, since the write limit of the burst buffer has been taken into consideration, when model SLBNP, SLBUP and algorithm ALBUP are used, the daily write workloads issued to the burst buffer are all less than 1,000TB (the dash line in Figure 5.4(a)) and the lifetime of the burst buffer can be extended to more than 5 years.

In Figure 5.4(b), we also illustrate the average amount of checkpoint data read from the burst buffer per day when different checkpoint placement models are used. Apparently, since model SUBNP, AUBNP and SLBNP allow all checkpoints to be written to the burst buffer directly without limit, the jobs always restart from failures using checkpoints stored on the burst buffer. That is why the average amount of data read from the burst buffer per day when using these three models are similar. For model SLBUP and algorithm ALBUP, the read workloads issued to the burst buffer are much less, as some of the checkpoints have been written to the parallel file system instead of the burst buffer. Moreover, using algorithm ALBUP generates 16.5% more read workloads than model SLBUP. This is because algorithm ALBUP dynamically adjusts the percentage of checkpoints written to the burst buffer based on the changing failure rate and increases the likelihood of restarting the job using checkpoints stored on the burst buffer.

Computation efficiency of the scientific applications and entire HPC system

Next we evaluate the computation efficiency of each scientific application job as well as that of the entire HPC system when different checkpoint placement models are applied to.

The computation efficiency of the i-th scientific application is defined as follows:

$$i\text{-th job's computation efficiency} = \frac{T_{cmpt,i}}{T_{total,i}} \times 100\%, \tag{5.19}$$

where $T_{total,i}$ is the total execution time and $T_{cmpt,i}$ is the total time spent on computation. As shown in Figure 5.5, if model SUBNP is used, each scientific application job can achieve the best computation efficiency. Compare to model SUBNP, using model AUBNP results in a slightly decrease in the computation efficiency as the dynamic adjustment of checkpoint interval might potentially increase the wasted computation time. However, both model SUBNP and AUBNP cannot satisfy the burst buffer endurance requirement. If the write limits of the burst buffer is set as 1,000TB per day, using model SLBNP reduces the computation efficiency of jobs that generate large checkpoints (such as CHIMERA and GTC) significantly. This is because such model intends to increase the checkpoint intervals of those jobs to reduce the amount of data written to the burst buffer, which inevitably increases the wasted time due to potential failures of compute nodes. On the other hand, with the same write limits, if our SLBUP model or ALBUP algorithm is used, computation efficiency of each job at most decreases by 5% compare to model SUBNP. This is because these two models utilize both burst buffer and the underlying parallel file

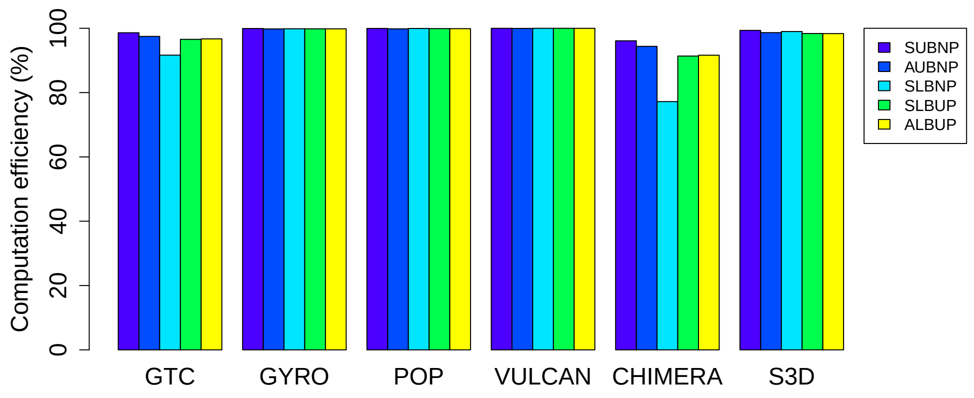


Figure 5.5: Average computation efficiency of each scientific application job when different checkpoint placement models or algorithms are used

system collaboratively to store checkpoint data and can keep the original optimal checkpoint interval to some extent without exceeding the write limit of the burst buffer.

The computation efficiency of the entire HPC system is defined as follows:

$$\text{system computation efficiency} = \frac{\sum_{i=1}^{M} n_i T_{cmpt,i}}{\sum_{i=1}^{M} n_i T_{total,i}} \times 100\%, \tag{5.20}$$

where M is the number of jobs running on the HPC system and n_i is the size of the i-th job. Therefore, after obtaining the average total execution time and computation time of each scientific application job from the simulation, we can calculate the system computation efficiency when different checkpoint placement models or algorithms are used. The calculation results are listed in TABLE 5.5.

Table 5.5: System computation efficiency when different checkpoint placement models or algorithms are used

Model/algorithm	System efficiency (%)
SUBNP	97.55
AUBNP	96.22
SLBNP	84.57
SLBUP	94.40
ALBUP	94.58

The calculation results demonstrate that, compare to model SUBNP, the system efficiency decreases by 13% when model SLBNP is used, while using our SLBUP model or ALBUP algorithm, the system efficiency at most decrease by 3%.

Comparison of static vs. adaptive checkpoint placement

Since some runtime characteristics, such as the failure rate and job sizes, might vary with time during the execution of the scientific applications, the checkpoint placement optimization needs to adapt to the changing runtime characteristics to achieve the most efficient utilization of the burst buffer. Next, we present the comparison between the static and the adaptive checkpoint placement when the runtime characteristics are not constant during the operation of the HPC system.

First, let us study how static and adaptive checkpoint placement perform when the per-node failure rate is time-dependent. In the previous section, through failure trace analysis, we have revealed that the time between computer node failures of Titan fits a Weibull distribution with decreasing failure rate. By using such failure trace to drive the simulation, we obtain the evaluation results shown in TABLE 5.5. From TABLE 5.5 we can observe that, though not much, our adaptive checkpoint placement algorithm ALBUP does achieve higher system computation efficiency than the static model SLBUP. The reason that algorithm ALBUP performs better than model SLBUP when the failure rate is time-dependent can be clearly illustrated using Figure 5.6.

In Figure 5.6, we adopt box plot to visualize the distribution of different runtime overhead observed in 1,000 simulation runs of each model, including system time spent on writing checkpoints, wasted computation due to failures and job restarting. From this figure, we can observe that: 1) Algorithm ALBUP can significantly reduce the system time spent on checkpointing. This is because ALBUP can dynamically adjust the checkpoint intervals for each running job based on the time-dependent failure rate and reduce the number of checkpoints written to the storage. 2) Though the number of checkpoints reduces when ALBUP algorithm is used, the distribution of wasted

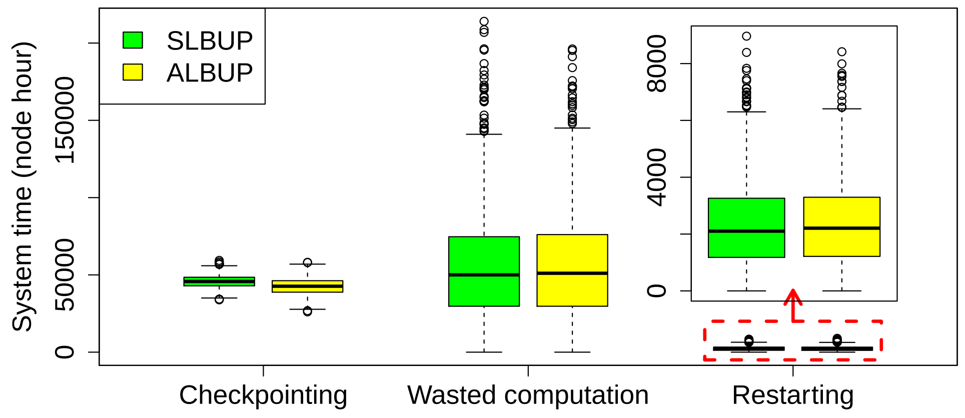
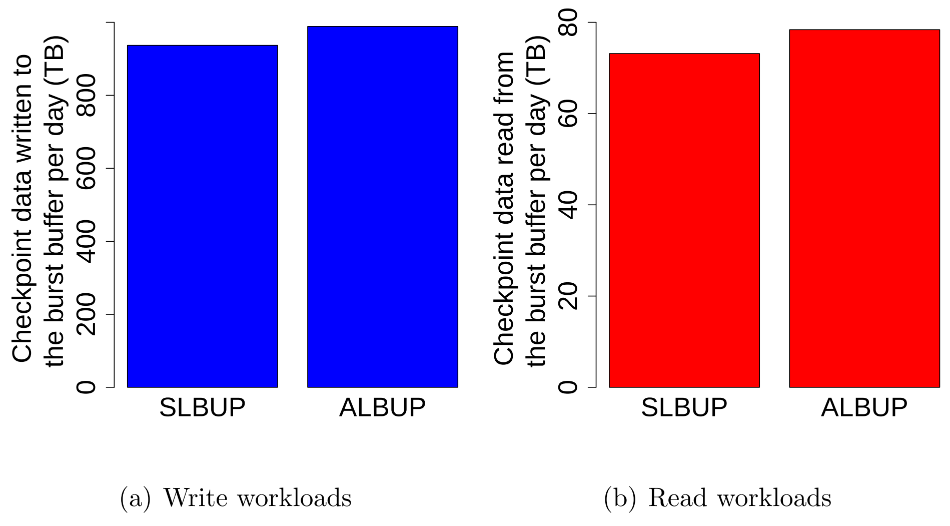


Figure 5.6: Box plot of system time spent on writing checkpoints, wasted computation due to failures and job restarting, when the node failure rate is time-dependent (1,000 simulation runs for each model)

computation time and the time spent on restarting the failed jobs have no obvious change compare to model SLBUP, which means the checkpointing overhead is reduced without increasing other overhead. 3) Actually, when we calculate the expectations of these two distributions, it shows that the average wasted computation time is reduced by 175 node hour while the average time spent on restarting is decreased by 72 node hour if ALBUP is used, which indicates that our ALBUP algorithm can perform checkpoint placement more efficiently compare to our SLBUP model.

Second, we evaluate how static and adaptive checkpoint placement perform when the job sizes are varying over time. In our evaluation, we assume that once a job fails, the chance it will lose some nodes after restarting is 70% and the percentage of nodes it will lose is a random variable whose value is selected between 0 and 10%. Then we run the simulation for model SLBUP and algorithm ALBUP respectively and the results are shown in Figure 5.7.

As shown in Figure 5.7(a), when the job sizes are varying over time, the total amount of checkpoint data written to the burst buffer is only 930TB per day if the static checkpoint placement is used, 7% less than the 1,000TB per day write limit, which means the burst buffer is not fully utilized. However, if the adaptive checkpoint placement is used, the daily write workloads to the burst buffer is still around 1,000TB



(a) Write workloads

(b) Read workloads

Figure 5.7: Average amount of checkpoint data written to and read from the burst buffer per day when job sizes are varying over time

per day. From Figure 5.7(b), we can also observe similar phenomenon on the total amount of checkpoint data read from the burst buffer. This is because the ALBUP algorithm can dynamically adjust the percentage of checkpoints written to the burst buffer based on the changing job sizes so that the burst buffer can always be fully utilized. Besides, we also calculate the average system computation efficiency when the job sizes are varying over time. The calculation results indicate that using model SLBUP the system efficiency is 94.67%, while using the ALBUP algorithm the system can achieve 94.75% efficiency.

Effect of optimization period on adaptive checkpoint placement

Since our ALBUP algorithm is designed to be triggered periodically to optimize the checkpoint placement, the length of the time period between two consecutive calls of ALBUP might have non-negligible impact on the model's performance. Therefore, we also evaluate our ALBUP algorithm by varying the period of triggering it during system operation.

As shown in Figure 5.8, when we increase the period of calling the ALBUP algorithm from 1 hour to 9 hours, we obtain the following results: 1) The average

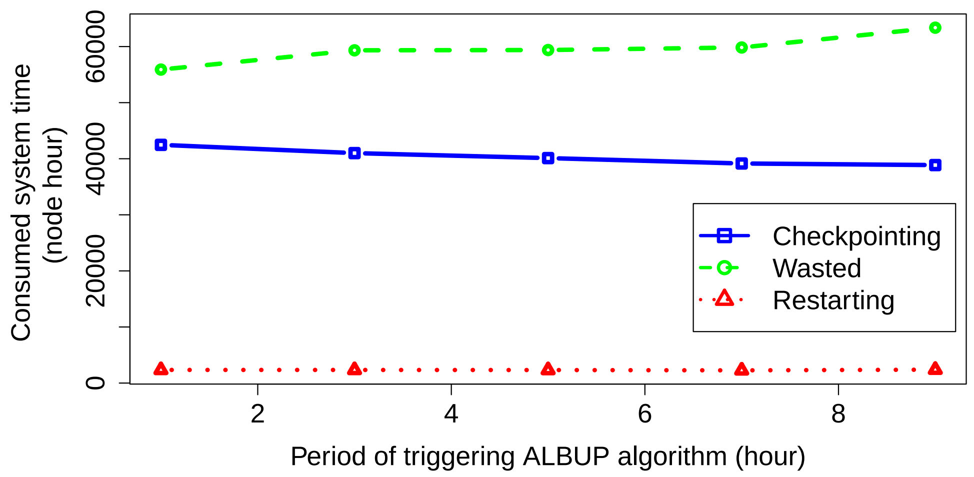


Figure 5.8: Performance of the ALBUP algorithm when varying the period between optimizations

system time spent on checkpointing decreases. Specifically, the increase of the optimization period results in underestimate on the number of node failures during each period, which makes the algorithm incorrectly extend the checkpoint intervals. Since less checkpoints are written, the time consumed by the checkpointing operation decreases. 2) The average wasted computation time increases significantly. This is also caused by the increase of the checkpoint interval, as more failed jobs have to restart using checkpoints that were not recently written. 3) The system time spent on restarting the failed jobs has no obvious variation. 4) The system computation efficiency does not decrease significantly until the optimization period is longer than 9 hours, demonstrating the robustness of our ALBUP algorithm.

Chapter 6

Conclusion

In this dissertation, I focus on the following three critical issues that commonly exist in maintaining and managing large-scale storage systems: 1) How to minimize the impact brought by component failures and ensure a highly operational experience in maintaining large-scale storage systems? 2) How to cost-effectively integrate solid-state drives (SSD) into large-scale storage system to improve system performance and efficiency? 3) How to maximize computation efficiency of large-scale scientific applications while guarantee the endurance requirements of the SSD-based burst buffer in high performance hierarchical storage systems? In order to solve these issues, I propose multiple novel models and algorithms.

One of the major challenges I encountered was how to evaluate these models and algorithms, as those deployed large-scale storage systems are seldom open to public, let alone allowed to make some changes to them. Therefore, all the models and algorithms proposed in my dissertation have been evaluated through simulation. In order to guarantee the fidelity of the simulation results, I setup the simulation based on the real parameters of those deployed large-scale storage systems and use the data collected from the real storage systems to validate the simulation results. The evaluation results demonstrate that: 1) the simulation results are comparable with data gathered from real system measurement, 2) these proposed models and algorithms can significantly improve the reliability and efficiency of large-scale storage systems.

Bibliography

Agrawal, N., Prabhakaran, V., Wobber, T., Davis, J. D., Manasse, M., and Panigrahy, R. (2008). Design Tradeoffs for SSD Performance. In *USENIX 2008 Annual Technical Conference*, ATC'08, pages 57–70. 9

Alam, M. and Mani, V. (1988). Queueing Model of a Bi-Level Markov Service-System and Its Solution using Recursion. *Transactions on Reliability*, 37:427–433. 6

Beaver, D., Kumar, S., Li, H. C., Sobel, J., and Vajgel, P. (2010). Finding a Needle in Haystack: Facebook's Photo Storage. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–8. 1

Behlendorf, B. (2012). Sequoia's 55PB Lustre+ZFS Filesystem. In *Lustre User Group (LUG) Meeting*. OpenSFS. 1

Bent, J., Faibish, S., Ahrens, J., Grider, G., Patchett, J., Tzelnic, P., and Woodring, J. (2012). Jitter-Free Co-Processing on a Prototype Exascale Storage Stack. In *28th Symposium on Mass Storage Systems and Technologies*, MSST '12, pages 1–5. 8, 56

Cai, M., Chervenak, A., and Frank, M. (2004). A Peer-to-Peer Replica Location Service Based on a Distributed Hash Table. In *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, SC '04, pages 56–67. 6

Chen, F., Koufaty, D. A., and Zhang, X. (2009). Understanding Intrinsic Characteristics and System Implications of Flash Memory based Solid State Drives. In *Proceedings of the Eleventh International Joint Conference on Measurement and*

Modeling of Computer Systems, volume 37 of *SIGMETRICS '09*, pages 181–192. ACM. 51

Chen, F., Koufaty, D. A., and Zhang, X. (2011a). Hystor: Making the Best Use of Solid State Drives in High Performance Storage Systems. In *Proceedings of the International Conference on Supercomputing*, ICS '11, pages 22–32, New York, NY, USA. ACM. 7, 9

Chen, F., Luo, T., and Zhang, X. (2011b). CAFTL: A Content-aware Flash Translation Layer Enhancing the Lifespan of Flash Memory Based Solid State Drives. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, FAST '11, pages 6–6. 9

Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., and Patterson, D. A. (1994). RAID: High-performance, Reliable Secondary Storage. *ACM Computing Surveys*, 26(2):145–185. 5, 16

Daly, J. T. (2006). A Higher Order Estimate of the Optimum Checkpoint Interval for Restart Dumps. *Future Generation Computer Systems*, 22(3):303–312. 9, 60, 61

DataDirect Networks, Inc. (2011). S2A9900 Datasheet, `http://www.ddn.com/support/downloads-documentation/`. 14

DataDirect Networks, Inc. (2014). DDN SFA12K Family, `http://www.ddn.com/products/storage-platform-sfa12kx/`. 14

Devroye, L. (1986). Sample-based Non-uniform Random Variate Generation. In *Proceedings of the 18th Conference on Winter Simulation*, WSC '86, pages 260–265. 23

El-Sayed, N. and Schroeder, B. (2014). To Checkpoint or not To Checkpoint: Understanding Energy-Performance-I/O Tradeoffs in HPC Checkpointing. In *Proceedings of the International Conference on Cluster Computing*, CLUSTER '14, pages 93–102. 68

Elerath, J. G. and Pecht, M. (2007). Enhanced reliability modeling of raid storage systems. In *In Proceedings of the International Conference on Dependable Systems and Networks (DSN*, pages 175–184. 5, 18

Elerath, J. G. and Schindler, J. (2014). Beyond MTTDL: A Closed-Form RAID 6 Reliability Equation. *ACM Transactions on Storage*, 10(2):7:1–7:21. 5, 18

Fang, A. and Chien, A. A. (2015). How Much SSD Is Useful for Resilience in Supercomputers. In *Proceedings of the 5th Workshop on Fault Tolerance for HPC at eXtreme Scale*, FTXS '15, pages 47–54. 9, 58, 60, 62, 79

Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003a). The Google File System. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 29–43. 1

Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003b). The Google File System. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 29–43, New York, NY, USA. ACM. 48

Ghodrati, B., Benjevic, D., and Jardine, A. (2012). Product support improvement by considering system operating environment: A case study on spare parts procurement. *International Journal of Quality and Reliability Management*, 29(4):436–450. 6

Gibson, G. A. and Patterson, D. A. (1993). Designing Disk Arrays for High Data Reliability. *Journal of Parallel and Distributed Computing*, 17(1-2):4–27. 5, 16

Greenan, K. (2009). Reliability and Power-Efficiency in Erasure-Coded Storage Systems. Technical Report UCSC-SSRC-09-08, University of California, Santa Cruz. 5, 16, 18

Greenwood, P. E. and Nikulin, M. S. (1996). *A Guide to Chi-Squared Testing*. Wiley, New York. 22

Hill, J. (June 13, 2014). Spider I system administrators on component replacement time. 18

Holland, M. and Gibson, G. A. (1992). *Parity Declustering for Continuous Operation in Redundant Disk Arrays*, volume 27. ACM. 27

Honicky, R. J. and Miller, E. L. (2004). Replication Under Scalable Hashing: A Family of Algorithms for Scalable Decentralized Data Distribution. In *Proceedings of the 18th International Parallel & Distributed Processing Symposium*, IPDPS '04. 6

Hu, X., Eleftheriou, E., Haas, R., Iliadis, I., and Pletka, R. (2009). Write Amplification Analysis in Flash-based Solid State Drives. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, SYSTOR '09, pages 10:1–10:9. 57

IBM DS8000 Series (2014). `http://www-03.ibm.com/systems/storage/disk/ds8000/overview.html`. 14

Jardine, A. and Tsang, A. (2005). *Maintenance, Replacement, and Reliability: Theory and Applications*. Dekker Mechanical Engineering. Taylor & Francis. 6

Kaiser, J., Margaglia, F., and Brinkmann, A. (2013). Extending SSD Lifetime in Database Applications with Page Overwrites. In *Proceedings of the 6th International Systems and Storage Conference*, SYSTOR '13, pages 11:1–11:12. 57

Kuenning, G. (2005). LASR Traces, `http://www.lasr.cs.ucla.edu/seer/seer_traces.html`. 43, 50

Laboratory, L. B. N. (2015). Berkeley Lab Checkpoint/Restart (BLCR) for LINUX, `http://crd.lbl.gov/departments/computer-science/CLaSS/research/BLCR/`. 72

Lee, E. K. and Thekkath, C. A. (1996). Petal: Distributed Virtual Disks. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS VII, pages 84–92. 6

Lee, S., Kim, T., Kim, K., and Kim, J. (2012). Lifetime Management of Flash-based SSDs Using Recovery-aware Dynamic Throttling. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST '12, pages 26–26. 57

Lewis, T. P. and Cochran, J. K. (1995). Applying Queueing Theory to Improve the Modeling of Spares Provisioning of Small Combat Aircraft Units. In *Proceedings of the 17th International Conference on Computers and Industrial Engineering*, ICC&IE '94, pages 297–301, Tarrytown, NY, USA. Pergamon Press, Inc. 6

Liu, N., Cope, J., Carns, P. H., Carothers, C. D., Ross, R. B., Grider, G., Crume, A., and Maltzahn, C. (2012). On the Role of Burst Buffers in Leadership-Class Storage Systems. In *28th Symposium on Mass Storage Systems and Technologies*, MSST '12, pages 1–11. 8, 56

Lu, Y., Shu, J., and Zheng, W. (2013). Extending the Lifetime of Flash-based Storage Through Reducing Write Amplification from File Systems. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies*, FAST '13, pages 257–270. 9

MacCormick, J., Murphy, N., Najork, M., Thekkath, C. A., and Zhou, L. (2004). Boxwood: Abstractions As the Foundation for Storage Infrastructure. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI '04, pages 105–120. 6

Mani, V. and Sarma, V. (1984). Queuing Network Models for Aircraft Availability and Spares Management. *Transction on Reliability*, R-33(3):257–262. 6

Megiddo, N. and Modha, D. S. (2003). ARC: A Self-Tuning, Low Overhead Replacement Cache. In *Proceedings of the 2Nd USENIX Conference on File and Storage Technologies*, FAST '03, pages 115–130. 43

Moody, A., Bronevetsky, G., Mohror, K., and Supinski, B. R. d. (2010). Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11. 8

NERSC (2016). NERSC File Systems, `http://www.nersc.gov/users/storage-and-file-systems/file-systems/`. 53

NetApp, Inc. (2014). FAS8080 EX, `http://www.netapp.com/us/products/storage-systems/fas8000/`. 14

OLCF (2012). ORNL Debuts Titan Supercomputer, `https://www.olcf.ornl.gov/wp-content/themes/olcf/titan/Titan_Debuts.pdf`. 57

Oral, S., Dillow, D. A., Fuller, D., Hill, J., Leverman, D., Vazhkudai, S. S., Wang, F., Kim, Y., Rogers, J., Simmons, J., et al. (2013). OLCF's 1 TB/s, Next-Generation Lustre File System. *Proceedings of the Cray User Group Conference (CUG)*. 1, 74, 77

Oral, S., Simmons, J., Hill, J., Leverman, D., Wang, F., Ezell, M., Miller, R., Fuller, D., Gunasekaran, R., Kim, Y., Gupta, S., Tiwari, D., Vazhkudai, S. S., Rogers, J. H., Dillow, D., Shipman, G. M., and Bland, A. S. (2014). 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*, SC '14, pages 217–228. 74, 77

Panasas, Inc. (2014). ActiveStor 16 & 18, `http://www.panasas.com/products/activestor`. 14

Park, S. and Shen, K. (2009a). A Performance Evaluation of Scientific I/O Workloads on Flash-Based SSDs. In *Proceedings of the 2009 International Conference on Cluster Computing*, pages 1–5. 8

Park, S. and Shen, K. (2009b). A Performance Evaluation of Scientific I/O Workloads on Flash-based SSDs. In *IEEE International Conference on Cluster Computing and Workshops*, CLUSTER '09, pages 1–5. IEEE. 40

Patterson, D. A., Gibson, G., and Katz, R. H. (1988). A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, SIGMOD '88, pages 109–116. 5, 16

Payer, H., Sanvido, M. A., Bandic, Z. Z., and Kirsch, C. M. (2009). Combo Drive: Optimizing Cost and Performance in a Heterogeneous Storage Device. In *Proceedings of the 1st Workshop on integrating solid-state memory into the storage hierarchy*, WISH '09. 8

Petascale Data Storage Institute (2016). Failure Data, http://pdsi.nersc.gov/. 53

Pinheiro, E., Weber, W.-D., and Barroso, L. A. (2007). Failure Trends in a Large Disk Drive Population. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, FAST '07, pages 2–2. 16

Pritchett, T. and Thottethodi, M. (2010). SieveStore: A Highly-selective, Ensemble-level Disk Cache for Cost-performance. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 163–174, New York, NY, USA. ACM. 7

Rao, K. K., Hafner, J. L., and Golding, R. A. (2006). Reliability for Networked Storage Nodes. In *International Conference on Dependable Systems and Networks (DSN)*, pages 237–248. IEEE Computer Society. 5

Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. (2001). A Scalable Content-addressable Network. In *Proceedings of the 2001 Conference* *on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 161–172. 6

Rausand, M. and Hoyland, A. (2003). *System Reliability Theory: Models, Statistical Methods and Applications*. Wiley-IEEE, 3 edition. 20

Rowstron, A. and Druschel, P. (2001). Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-peer Storage Utility. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, pages 188–201. 6

Sakai, K., Sumimoto, S., and Kurokawa, M. (2012). High-Performance and Highly Reliable File System for the K Computer. *FUJITSU Science Technology*, 48(3):302–209. 1

Samsung (2015). Samsung SSD 850 PRO Series Datasheet, `http://www.samsung.com/global/business/semiconductor/minisite/SSD/downloads/document/Samsung_SSD_850_PRO_Data_Sheet_rev_2_0.pdf`. 57, 76

Sato, K., Maruyama, N., Mohror, K., Moody, A., Gamblin, T., de Supinski, B. R., and Matsuoka, S. (2012). Design and Modeling of a Non-blocking Checkpointing System. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 19:1–19:10. 8, 56

Sato, K., Mohror, K., Moody, A., Gamblin, T., Supinski, B. R. d., Maruyama, N., and Matsuoka, S. (2014). A User-Level InfiniBand-Based File System and Checkpoint Strategy for Burst Buffers. In *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGrid '14, pages 21–30. 8, 56

SchedMD (2015). Slurm Workload Manager, `http://slurm.schedmd.com/`. 72

Schroeder, B. and Gibson, G. A. (2006). A Large-scale Study of Failures in High-performance Computing Systems. In *Proceedings of the International Conference on Dependable Systems and Networks*, DSN '06, pages 249–258. 68

Schroeder, B. and Gibson, G. A. (2007). Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You? In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, FAST '07. 17

Schulze, M., Gibson, G., Katz, R., and Patterson, D. (1989). How Reliable Is A RAID. In *Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage, Digest of Papers*, pages 118–123. IEEE. 5, 16

Schwarz, T. J. E., Xin, Q., Miller, E. L., Long, D. D. E., Hospodor, A., and Ng, S. W. (2004). Disk Scrubbing in Large Archival Storage Systems. In *12th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, MASCOTS '04, pages 409–418. 18

SDSC (2015). Gordon Supercomputer, `http://www.sdsc.edu/support/user_guides/gordon.html`. 2, 76

Seagate Technology (2014). ClusterStor 9000, `www.xyratex.com/products/clusterstor-9000`. 14

Shipman, G., Dillow, D., Oral, S., and Wang, F. (2009). The Spider Center Wide File System: From Concept to Reality. In *Cray User Group (CUG) Conference*, Atlanta. 1, 14

Soundararajan, G., Prabhakaran, V., Balakrishnan, M., and Wobber, T. (2010). Extending SSD Lifetimes with Disk-based Write Caches. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, FAST '10, pages 8–8. 9

Srinivasan, M., Saab, P., and Tkachenko, V. (2015). Flashcache, `https://github.com/facebook/flashcache/`. 7

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. (2001). Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 149–160. 6

Tiwari, D., Gupta, S., and Vazhkudai, S. S. (2014). Lazy Checkpointing: Exploiting Temporal Locality in Failures to Mitigate Checkpointing Overheads on Extreme-Scale Systems. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '14, pages 25–36. 57, 61, 68, 75, 79

UCSF (2015). UCSF CHIMERA: an Extensible Molecular Modeling System, `http://www.rbvi.ucsf.edu/chimera/`. 57

Vaidya, N. H. (1997). Impact of Checkpoint Latency on Overhead Ratio of a Checkpointing Scheme. *IEEE Transactions on Computers*, 46(8):942–947. 9, 60, 61

Vaughan, T. S. (2005). Failure Replacement and Preventive Maintenance Spare Parts Ordering Policy. *European Journal of Operational Research*, 161(1):183–190. 6

Wächter, A. and Biegler, L. T. (2006). On the Implementation of an Interior-point Filter Line-search Algorithm for Large-scale Nonlinear Programming. *Mathematical Programming*, 106(1):25–57. 66

Wan, L., Lu, Z., Cao, Q., Wang, F., Oral, S., and Settlemyer, B. W. (2014a). SSD-Optimized Workload Placement with Adaptive Learning and Classification in HPC environments. In *IEEE 30th Symposium on Mass Storage Systems and Technologies*, MSST '14, pages 1–6. 41

Wan, L., Wang, F., Oral, S., Tiwari, D., Vazhkudai, S. S., and Cao, Q. (2015). A Practical Approach to Reconciling Availability, Performance, and Capacity in Provisioning Extreme-Scale Storage Systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 75:1–75:12. 14

Wan, L., Wang, F., Oral, S., Vazhkudai, S. S., and Cao, Q. (2014b). A Report on Simulation-Driven Reliability and Failure Analysis of Large-Scale Storage Systems. Technical Report ORNL/TM-2014/421, Oak Ridge National Laboratory. 14

Wang, T., Oral, S., Wang, Y., Settlemyer, B. W., Atchley, S., and Yu, W. (2014). BurstMem: a High-Performance Burst Buffer System for Scientific Applications. In *International Conference on Big Data*, pages 71–79. 8

Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D. E., and Maltzahn, C. (2006a). Ceph: A Scalable, High-performance Distributed File System. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 307–320. 48

Weil, S. A., Brandt, S. A., Miller, E. L., and Maltzahn, C. (2006b). CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06. 7, 48

Wu, G. and He, X. (2012). Delta-FTL: Improving SSD Lifetime via Exploiting Content Locality. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 253–266. 9

Wu, P.-L., Chang, Y.-H., and Kuo, T.-W. (2009). A File-system-aware FTL Design for Flash-memory Storage Systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pages 393–398. 9

Xin, Q., Miller, E. L., Schwarz, T. J. E., Long, D. D. E., Brandt, S. A., and Litwin, W. (2003). Reliability Mechanisms for Very Large Storage Systems. In *IEEE Symposium on Mass Storage Systems*, pages 146–156. 5

Xu, W., Lu, Y., Li, Q., Zhou, E., Song, Z., Dong, Y., Zhang, W., Wei, D., Zhang, X., Chen, H., Xing, J., and Yuan, Y. (2014). Hybrid Hierarchy Storage System in MilkyWay-2 Supercomputer. *Frontiers of Computer Science*, 8(3):367–377. 8, 56

Yang, Q. and Ren, J. (2011). I-CASH: Intelligently Coupled Array of SSD and HDD. In *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, HPCA '11, pages 278–289, Washington, DC, USA. IEEE Computer Society. 7, 9, 57

Young, J. W. (1974). A First Order Approximation to the Optimum Checkpoint Interval. *Communications of the ACM*, 17(9):530–531. 9, 60, 61

Zhang, X., Davis, K., and Jiang, S. (2012). iTransformer: Using SSD to Improve Disk Scheduling for High-performance I/O. In *IEEE 26th International Parallel & Distributed Processing Symposium*, IPDPS '12, pages 715–726. IEEE Computer Society. 7

Vita

Mr. Lipeng Wan is a Ph.D. candidate in computer science major at University of Tennessee, Knoxville. He received his M.S. degree (2011) in information & communication engineering from Southeast University, China, and his B.S. degree (June 2008) in communication engineering from Nanjing University of Science and Technology, China. His research interests include parallel and distributed storage systems, high performance computing, embedded systems, and performance optimization for solid-state drives.