ABSTRACT

BEHERA, SUBHENDU SEKHAR. Adaptive Multi-level Checkpointing On Modern High Performance Computing Systems. (Under the direction of Dr. Frank Mueller.)

Failures are imminent and diverse on today's High-Performance Computing (HPC) systems given the complexity of system design, scale, and heterogeneity of its components. These failures greatly reduce application efficiency. To mitigate these issues, Checkpoint/Restart (C/R) is widely used to provide fault tolerance on HPC systems. However, C/R adds significant overhead due to high latency to store a checkpoint on a Parallel File System (PFS).

This work develops an adaptive multi-level C/R model that incorporates multiple techniques: failure prediction and an analysis model, live migration, Burst Buffers (BBs), and safeguard checkpoints. The failure prediction and analysis model focuses on reducing the application overhead by predicting failure earlier and decides on when and where to write safeguard checkpoints, restart computation after failure, and migrate computation using live migration technique. The model also utilizes BBs to effectively reduce PFS contention during checkpoints and failure recovery. To optimize our model that incorporates asynchronous checkpoints from BBs to the PFS, we derive a first-order approximation of the optimal checkpoint interval for systems with BBs. We also derive a failure analysis model based on a rigorous log-based analysis of system logs collected from three real-world HPC systems. We incorporate this analysis in the derivation of the optimal checkpoint interval, which effectively reduces the frequency of checkpoints.

To evaluate our model, we simulate six real-world applications on an HPC system modeled after Summit, which is the fastest supercomputer as of November 2019. Results show a ≈53%-95% reduction in application overhead due to BBs, orchestrated failure prediction, migration, and checkpoint handling. We also observe a ≈29% decrease in the amount of checkpoint data written to BBs, which can reduce the wear-out and thus increase the longevity of the BB storage devices. Finally, we demonstrate the robustness of our C/R model against multiple failure distributions.

Adaptive Multi-level Checkpointing On Modern High Performance Computing Systems

by
Subhendu Sekhar Behera

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, North Carolina

2020

APPROVED BY:

Dr. Xipeng Shen	Dr. Xiaohui Gu

Dr. Frank Mueller
Chair of Advisory Committee

DEDICATION

To my family and friends.

BIOGRAPHY

The author was born in Berhampur, India. After completing his Bachelor in Technology in 2012, he worked for Broadcom communications, Bengaluru, India, and Juniper Networks, Bengaluru, India for more than five years. Since 2018 he has been pursuing his Master of Science degree in Computer Science at North Carolina State University, Raleigh, USA. He works as a Graduate Research Assistant under the guidance of Dr. Frank Mueller.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Frank Mueller, for his help and guidance. I would also like to thank Dr. Lipeng Wan, Computer Scientist, Oak Ridge National Laboratories, who I worked with during the summer of 2019.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1

INTRODUCTION

Failure rates in HPC are bound to increase in the era of exa-scale computing due to scale, heterogeneity, component count, and complexity of devices for compute, memory, networking, and storage [Gei03; Sat12; Sch07]. A DARPA report [Eln08] pointed out that large-scale systems waste 20% of its resources on failures and their recovery. Further, future exa-scale computing systems will have a mean time between failures (MTBF) in the order of hours or less. Table 1.1 [Gup17] represents some of the large-scale HPC systems and their MTBF.

Table 1.1 Failure Rate

HPC System	No. of Nodes	MTBF (hrs)
Jaguar XT4	7,832	36.91
Jaguar XT5	18,688	22.67
Jaguar XK6	18,688	8.93
Eos	736	189.04
Titan	18,688	14.51

C/R [Moo10] is the most common solution employed to provide fault tolerance where the application state is saved at regular intervals on permanent storage devices of a PFS on HPC systems. After failure, the application can continue execution from the point of failure with the help of a recovered application state.

However, with the rapid growth in computing power, the latency gap between computing and

the I/O subsystem has ever been on the rise. Further, the high contention for access to the PFS, which is amplified on exascale HPC systems, severely limits application performance [Luc14; Liu12]. A single checkpoint can take up to the order of minutes [Isk08].

In short, failures and high I/O contention add significant overhead to application execution and become the key challenge for C/R efficiency. This work aims to address these challenges in a novel manner exploiting recent technological advances.

1.1 Motivation

In past years, there have been significant contributions in failure prediction on large-scale systems [Das18a; Das18b]. Desh's model [Das18a] uses deep learning to identify sequences of system logs or phrases that may lead to failure. Based on the occurrence of such failure chains a prediction can be made in advance along with a lead time to avoid failure of a specific node.

The PFS has long been identified as a bottleneck for highly parallel applications [Khe19; Wan14; Liu12; Wan17]. Modern HPC systems address this problem by integrating BBs into the I/O subsystem as intermediate storage devices, which provide faster I/O performance via buffering. Also, there have been significant contributions in fault tolerance techniques such as live migration [Wan08; Wan09], and prediction-based checkpointing [Bou13; Tiw14].

We propose a multi-level C/R model that integrates Desh [Das18a] with its ability to predict failures. Based on the predicted lead time to failure, it chooses an appropriate action to avoid or at least reduce re-computation upon failure. It further incorporates BB devices into the model to reduce the latency for storing checkpoints, but also to consider additional latency before a checkpoint becomes globally available within the PFS, and to trade off C/R with live migration based on the predicted lead time to failure.

1.2 Hypothesis

We put forward the following hypothesis:

A multi-level C/R model for modern HPC systems can benefit in performance from failure prediction and an analysis model for effective use and orchestration of multiple fault resilience techniques such as safeguard checkpoints, live migration, and burst buffers.

We improve the efficiency of writing checkpoints by taking advantage of BBs. BBs can be used to store the checkpoint data faster than writes to PFS can. Once the data is in BBs, it is asynchronously bled off to PFS, i.e., computation within the application continues while a checkpoint previously committed to a BB is slowly written to the PFS in the background so that it can later be restored on a different node should the current node fail. These fast, node-local writes to the BB reduce the time required to store checkpoints on the critical path of application execution.

We leverage failure prediction models to reduce the overhead caused by re-computation due to failures. Our checkpoint model integrates Desh's model [Das18a] to predict failures with known lead times. We utilize the Desh's log-based failure chain characterization technique to perform rigorous failure analysis and prediction on system logs collected from real-world HPC systems to identify instances of different failures and distributions of lead times of such failures. We also derive a first-order approximation of the optimal checkpoint interval for the multi-level C/R model and further optimize it by incorporating a rigorous failure analysis model, which effectively reduces the frequency of checkpoints. The novelty is that it considers choices in actions such as adaptive safeguard checkpoints to different I/O layers and live migration to reduce the cost of re-computation upon immanent failures, and it studies the impact of such a trade-off.

Our study shows that the utilization of BBs results in a significant reduction of checkpoint overhead for both small and large-sized application checkpoints. The comparative study also shows that our failure analysis with its prediction model allows for orchestration of C/R with live migration resulting in a more significant reduction in application overhead than adaptive safeguard checkpoints without migration. An evaluation shows that our C/R model is robust and maintains its efficiency for different failure distributions.

The remainder of this document is organized as follows: Chapter 2 provides an overview of the background followed by related work in Chapter 3. Chapter 4 introduces the system design, develops our checkpoint model and explains how BBs and proactive live migration are utilized. Chapter 5 provides the failure analysis and derivation of the optimal checkpoint interval. Chapter 6 discusses the I/O performance model and its instantiation for ORNL's Summit system. Chapter 7 discusses results from experiments followed by limitations and future work in Chapter 8 and a summary of the contributions in Chapter 9.

CHAPTER

2

BACKGROUND

As failures are more and more common in large-scale systems, resiliency is considered a key feature of such systems. Fault tolerance solutions can recover the system from failures and reduce computational waste. There are three prominent categories of fault tolerance solutions: redundancy, algorithm-based fault tolerance, and checkpoint restart. There are many variants of theses solutions that have been implemented and their impact on system efficiency is significant. In this chapter, we discuss in brief existing solutions and the technologies that are being adopted in modern HPC systems.

2.1 Solutions

In this section, we discuss the three categories of resilience methods.

2.1.1 Redundancy

In this method, multiple redundant jobs are executed to perform the same task [Fer09; Ell12]. At the end of the computation, the final results are evaluated to check if the results match. This resiliency method mainly focuses on soft and silent errors, e.g., silent data corruption [Fia11]. However, redundancy requires more resources.

2.1.2 Algorithm Based Fault Tolerance

This class of resilience methods exploits the mathematical properties of applications and relies on additional data to provide fault tolerance [Gei03]. This requires applications to be modified and redesigned in such a manner that they can recover automatically after failures. There are two classes of ABFT techniques. The first one is checksum-based ABFT [Kua84] and relies on extra data that is checked at run time to detect soft errors. The second one is checksum-less ABFT, which exploits the mathematical properties of applications to recover from failures [Che13].

2.1.3 Checkpoint/Restart

C/R is a fault tolerance mechanism in which the application state is stored periodically on the permanent storage disk. Upon failure, the application can continue to run from the point of failure by recovering the stored data. Figure 2.1 depicts an execution with C/R. Since this work focuses on a C/R-based solution, we discuss it in further detail.

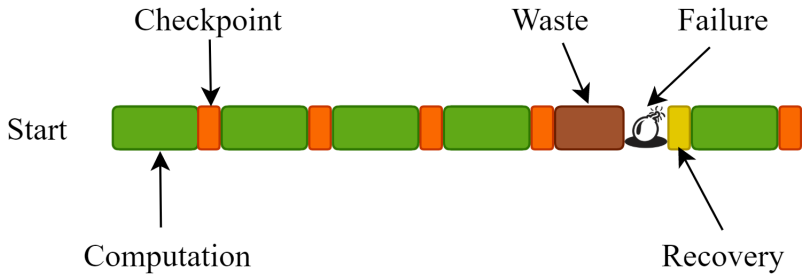

Figure 2.1 Checkpoint/Restart

C/R can be broadly classified into two categories. System-level: It allows the application state to be saved without changing any application code. Application-level: It requires application code to include the checkpoint APIs to be called. In general, a system-level C/R solution has a larger memory footprint than an application-level C/R solution.

C/R solutions can be further classified based on the below characteristics.

- ***Coordinated vs. Uncoordinated:*** Coordinated checkpointing mandates all the processes of a job to be synchronized during the checkpoint process [Har06]. In contrast, uncoordinated checkpointing relies on message logging to let each process store checkpoint data at their convenience with minimal or no synchronization [Cot06].
- ***Incremental Checkpointing:*** This version of checkpointing allows systems to reduce the checkpoint data size by allowing dirty or modified pages to be saved as checkpoint data after one initial full checkpoint [Wan09]. Many variants also allow incremental checkpointing along with less frequent full checkpoints.
- ***Diskless Checkpointing:*** This method requires applications to procure a certain amount of RAM to store checkpoint data. This allows for faster checkpointing [Pla93]. However, this is a disadvantage for applications with larger checkpoint data.
- ***Multilevel Checkpointing:*** Modern HPC systems have heterogeneous storage devices with different capacities and latencies. Multi-level checkpointing models [Moo10] take advantage of this hierarchical storage setting to store checkpoint data faster. Such systems determine checkpoint locations on different storage devices based on the failure rate, which results in a checkpoint interval for each storage level.

2.2 Modern Technology

Significant progress has been made in HPC ecosystems and resilience methods in recent time. We discuss them briefly below.

2.2.1 Burst Buffers

Access to the PFS has always been subject to high contention [Wan14; Liu12]. BBs act as an intermediate device to serve I/O requests faster, to reduce I/O blocking, and to improve I/O latency. Burst Buffers are typically NVMe SSD storage devices with low read and write latency. Some of the large-scale HPC systems equipped with BBs are Cori [Bhi16] and Summit [Vaz18].

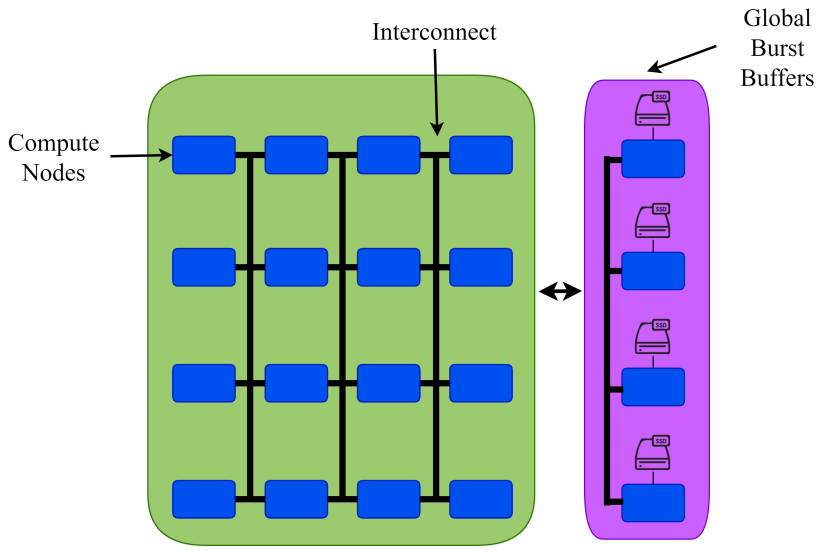

Figure 2.2 Global Burst Buffers

There are three different architectures based on which Burst Buffers are implemented. They are described briefly below.

- ***Global Burst Buffers:*** Global Burst Buffers are formed as a cluster of NVMe SSD devices. These devices are shared by all the compute nodes. In other words, they act together as a pseudo-PFS that has lower latency than the global PFS. NERSC's Cori [Bhi16] employs this Burst Buffer architecture. Figure 2.2 depicts the global Burst Buffers architecture.
- ***Grouped Burst Buffers:*** This is another architecture of Burst Buffers where compute nodes are grouped and allocated a specific portion of the Burst Buffers. This model takes advantage of local links to reduce network congestion and enhance latency. It also makes it easier to provide data sharing capabilities. Figure 2.3 depicts grouped Burst Buffers.

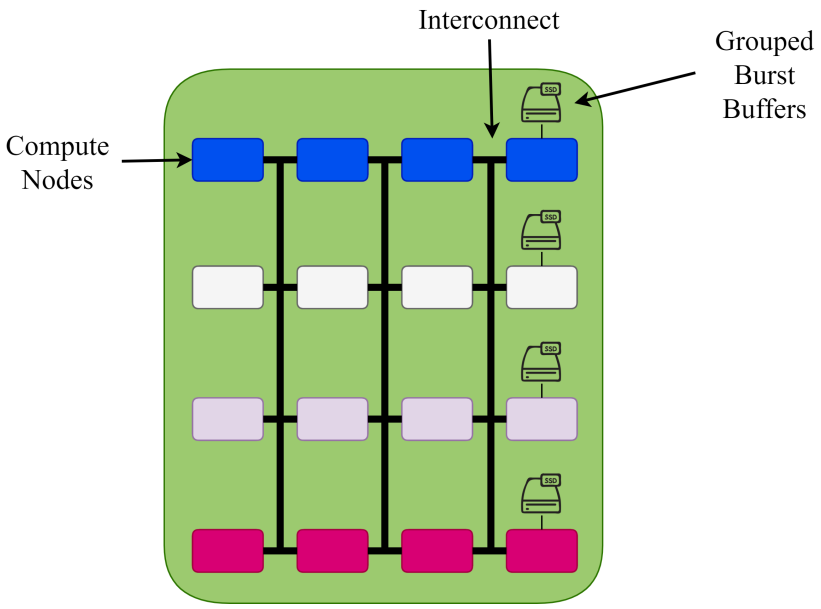

Figure 2.3 Grouped Burst Buffers

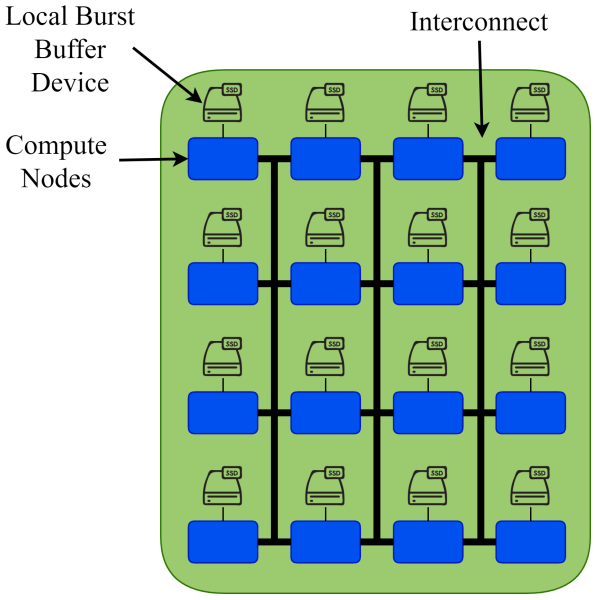

Figure 2.4 Local Burst Buffers

- ***Local Burst Buffers:*** In this category, the NVMe SSD devices are attached to individual compute nodes just like local hard disks. This provides zero contention among compute nodes. However, data sharing becomes a challenge in this scenario. Summit [Vaz18] has employed this Burst Buffer architecture. Figure 2.4 depicts the architecture of local Burst Buffers.

2.2.2 Live Migration

Live Migration [Wan08; Wan09] is the process in which a failing node can migrate its running processes to a new and healthy node. When a failure alert is received by the scheduler on the failing node, the process migration process starts. In general, there are three stages of the migration process. First, all the pages owned by the process are copied to the new node. Then the process execution is paused for a few seconds and all the dirty pages are copied. In the end, the process is terminated and the new process continues its execution.

2.2.3 Safeguard Checkpoint

Safeguard checkpointing [Bou13] is the just-in-time checkpoint that is performed in the presence of failure prediction. Figure 2.5 depicts the concept of safeguard checkpointing.

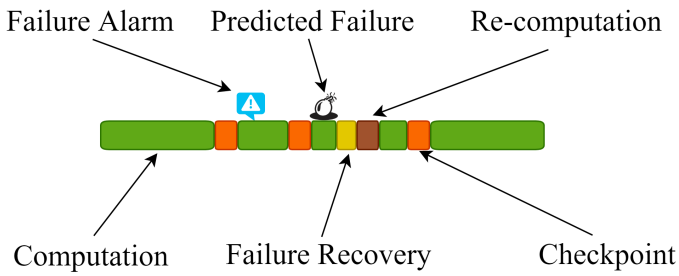

Figure 2.5 Safeguard Checkpoint

2.2.4 Failure Prediction

With significant progress made in the field of ML, there has been tremendous improvement in Failure Prediction(FP). Apart from the fault tolerance techniques mentioned previously, FP can provide more efficient fault tolerance. DeepLog [Du17], Cloudseer [Yu16], and ELSA [Gai12] are some of the contribution in this direction. Further, Desh [Das18c] contributed to predicting node failures with lead times. We use the Desh model in our C/R solution to orchestrate multiple fault tolerance techniques in combination with advanced storage technology.

CHAPTER

3

RELATED WORK

Table 3.1 C/R Models Comparison

C/R Model	Failure Awareness	Safeguard Checkpoint	Live Migration	BB to PFS bleed off	Failure Prediction
Our Work	Lead Time	✓	✓	✓	✓
Wang et al.	Health Monitoring	✗	✓	✗	✗
Bouguerra et al.	Lead Time	✓	✗	✗	✓
Tiwari et al.	Failure Locality	✗	✓	✗	✗
George et al.	Failure Locality	✗	✗	✗	✗
Garg et al.	Failure Locality	✗	✗	✗	✗

3.1 Multilevel Checkpointing and Optimal Checkpoint Interval

A number of extensive and significant contributions have been made to address fault resilience with C/R techniques over the past years. C/R solutions [Moo10; Di17; Di14; Ben17; Sat12] focus on reducing checkpoint overhead by using multiple storage devices with varying latency to store checkpoints. Moody et al. [Moo10] distribute checkpoints to RAM, flash devices, hard disks, and the PFS depending on the commonality of failures. Di et al. [Di14; Di17] optimize the selection of the location of checkpoints based on a failure distribution and compute the optimal checkpoint interval for each level. Benoit et al. [Ben17] assess the cost of a multilevel checkpoints, characterize the pattern of checkpoints, and further, provide a dynamic programming solution to find the optimal subset of levels for checkpointing. Sato et al. [Sat12] implement a non-blocking checkpoint mechanism in which checkpoints are stored in node-local storage devices and later bled off to the PFS. These works provide a foundation for our work but lack the integration of failure prediction into C/R models.

Further, our derivation of the first-order optimal checkpoint interval is novel for a C/R solution where checkpoints are cached and then asynchronously stored on to a secondary storage. Previous work by Di et al. [Di14; Di17] and Benoit et al. [Ben17] focused on the optimal checkpoint interval for multiple types of checkpoints, each of them stored on a separate storage medium.

3.2 Using Burst Buffers

Other studies [Liu12; Wan14; Sat14; Fan15; Wan17] have investigated the use of BBs to reduce the checkpoint and the I/O overhead in large-scale HPC systems. Liu et al. [Liu12] and Wang et al. [Wan14] use the BBs to absorb I/O bursts temporarily and analyze their impact on application efficiency. Sato et al. [Sat14] developed a checkpoint strategy to use BBs for large-scale systems. Fang et al. [Fan15] study BBs capacity requirement, and Wan et al. [Wan17] utilize both the PFS and BBs with guaranteed BB endurance. Our solution, a two-level C/R model that stores its checkpoints efficiently, goes beyond these earlier approaches in its efficient recovery strategy upon failures to mandate only the replacement node to recover checkpoint from the PFS in orchestration with failure prediction, which reduces restart time considerably.

3.3 Failure-Aware C/R Solutions

Several failure-aware C/R mechanisms [Bou13; Tiw14; Geo15; Gar18; Wan08; Liu08] have been devised. Wang et al. [Wan08] monitor healthy nodes and migrate processes if the node's health deteriorates. Bouguerra et al. [Bou13] use proactive checkpoints upon failure prediction along with preventive checkpoints to reduce computational waste. However, as per our evaluation, proactive checkpoints fail when checkpoint data size grows. Tiwari et al. [Tiw14] increase the checkpoint interval until failure and skip selected checkpoints post-failure using a temporal distribution of failures. Our C/R model also takes a similar approach to update the optimal checkpoint rate at a fixed point during C/R handling. However, our C/R model differs significantly from the lazy checkpointing solution based on the prediction techniques underlying our model, which are unique.

George et al. [Geo15] deploy partial replication of a set of processes and use failure prediction to change the replicated process group. Garg et al. [Gar18] exploit failure locality to schedule applications with higher checkpoint overhead during lower failure rates and applications with lower checkpoint overhead during higher failure rates. In contrast, our model relies on dynamically predicted failures. Table 3.1 compares our C/R model with other C/R models and illustrates the uniqueness and comprehensiveness of our approach in contrast to prior work.

CHAPTER

4

SYSTEM DESIGN

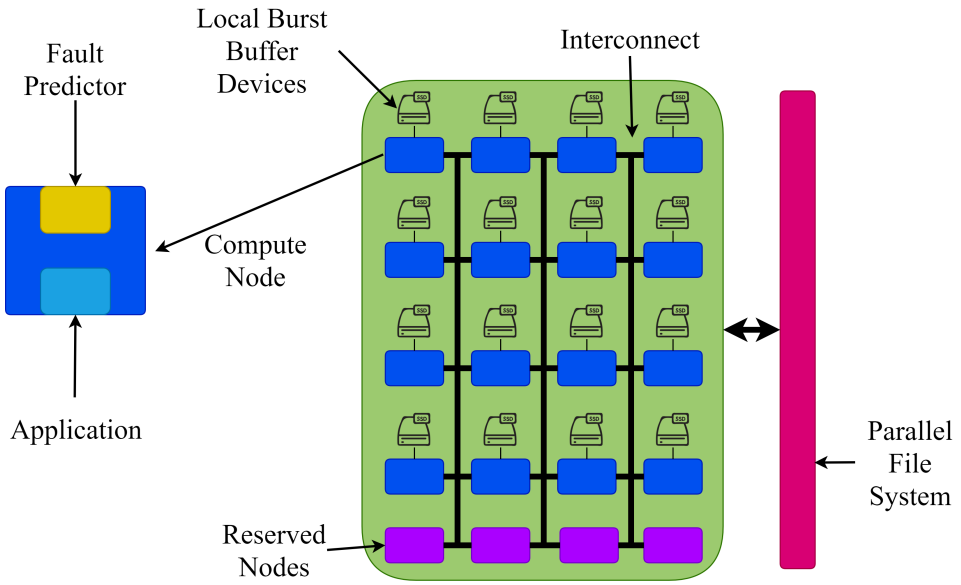

Figure 4.1 HPC System Architecture

Our C/R model is devised for a Summit-like HPC system shown in Figure 4.1. On this HPC system, the BB devices are locally attached to compute nodes. As our model uses proactive live migration, a small set of nodes is reserved but yet not allocated to any application. Workload managers such as Slurm [Yoo03] and Flux [Ahn18] support the allocation of spare nodes. We assume that spare nodes

are always available, which is given as long as rate of failure is slower than the rate at which failed nodes can be recovered. Upon a failure, a node can be taken from this reserved set to replace the failed one. Our failure predictor analyzes system logs on a per-node basis and predicts failures with their estimated lead times, i.e., the predictor daemon is placed on each compute node (e.g., in a spare core, which could be shared with other services). Since only system logs are used for analysis, predictors are independent of the application running on the compute nodes. Our model also features a multi-level adaptive checkpoint capability that comprises multiple techniques, including BB bleed-off, failure analysis and prediction, proactive live migration, and adaptive safeguard checkpoints, which are discussed in the following.

4.1 Checkpoint Model

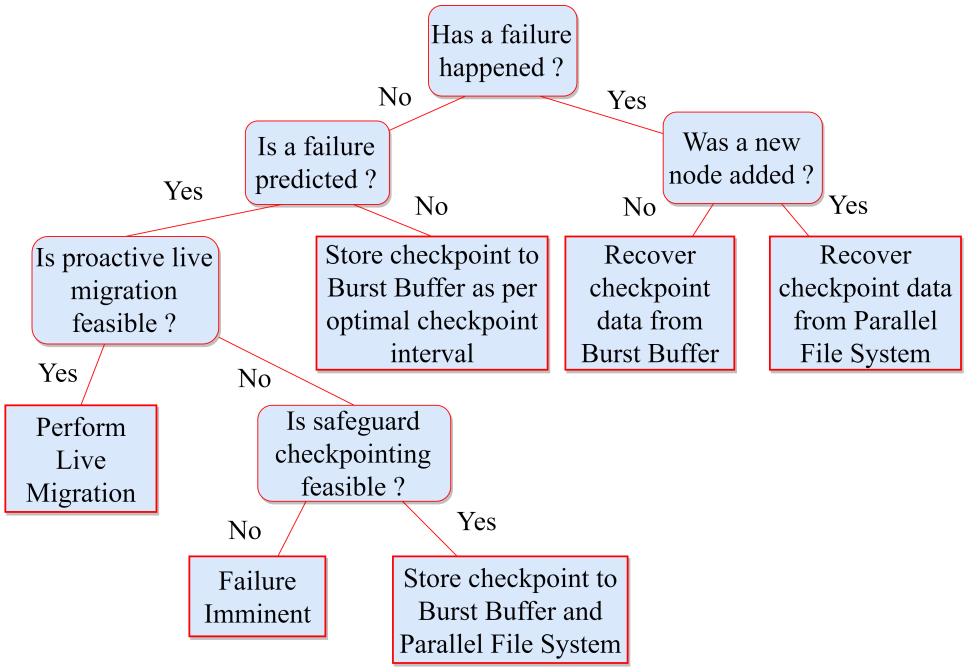

Figure 4.2 Decision Tree of the Checkpoint Model

Our base checkpoint model derives the optimal checkpoint interval from a failure analysis model without considering failure prediction. With the addition of failure prediction, the model is extended to select effective and informed actions based on the decision tree of Figure 4.2. The criteria based on which decisions are made are as follows:

- *Has a failure happened?* At the top of the decision tree, it is assessed if any failure has occurred. Upon a failure, the application state needs to be recovered from the last checkpoint in a BB or the PFS upon availability of an existing node or addition of a new node from the reserved set, respectively. The recovery strategy is to restart from the last checkpoint. In the absence of a failure, we move to the node where the failure prediction status is checked.

- *Was a new node added?* Upon a failure, all the ranks of an application traditionally recover the checkpoint data from the PFS. This introduces significant contention that increases with checkpoint data size and the number of processes in the application. Since Summit's BBs are local, our checkpoint model mandates only a newly added node, drawn from the reserved set to replace the failed one, to recover the checkpoint data from the PFS. The remaining nodes recover the checkpoint data from local BBs. This considerably reduces failure recovery overhead.

- *Is a failure predicted?* In the absence of a predicted failure, the checkpoint model falls back to the optimal checkpoint interval to save the application state periodically. This provides tolerance against the failures that are neither predictable nor can be handled with proactive actions such as live migration and adaptive safeguard checkpoints. If a failure can be predicted, then the next action is indicated further down in the decision tree.

- *Is proactive live migration feasible?* If a failure is predicted, then we select an action based on the lead time to failure. With enough lead time to proactively migrate the application, the failure can be avoided by live migration. The application can continue with its computation while being migrated to a new and healthy node selected from the set of reserved nodes. The failed node is recovered and added to the set of reserved nodes for future use. If lead time is insufficient for migration, then the next action is indicated further down in the decision tree.

- *Is safeguard checkpointing feasible?* Without enough lead time for live migration, the application state is saved with a safeguard checkpoint to the BB, which is a just-in-time checkpoint on top of the periodic checkpoints of the base model. Upon failure, application execution restarts from the safeguard checkpoint if it completed. The total time needed to commit a safeguard checkpoint is the combination of time required to store the checkpoint to BB and then PFS, one after another. Applications with longer checkpoint intervals can benefit from these just-in-time safeguard checkpoints in that less recompute is required after restarting from a checkpoint. One may argue that the checkpoint can be directly written to the PFS bypassing the BB, but this may increase PFS contention during application restart as all processes need to recover the checkpoint from PFS. Hence, the safeguard checkpoint is stored in the BBs of all nodes and then bled off to PFS. The replacement node recovers the checkpoint from PFS while other nodes recover theirs from BBs. This allows applications to recover faster. If the predicted lead time is insufficient for both checkpointing and live migration, no action is taken as failure is unavoidable. Upon failure, application execution restarts from the previous regular checkpoint.

One point of discussion is the accuracy of prediction. It is assumed here that the accuracy is high (see [Das18a]) and lead times are reliable except for a few outliers [Das18b]. Should lead times vary significantly, then safeguard checkpoints should be committed even if their write times exceed the lead time. Clearly, if prediction accuracy itself was low, only periodic checkpoints of the base model should be taken, which is not the focus of this work. In our model, live migration takes precedence over safeguard checkpoint. The reason for doing this is the ability to perform computation during live migration, which is explained in Section 4.4.

The following assumptions underlie the model:

1. Any checkpoint made to save the application state is performed by all the processes of an application.

2. Failures happen on a single compute node and do not propagate to other nodes.

3. No distinction is made between soft failures and hard/node failures, i.e., both are handled uniformly.

4. The data is transferred from BBs to the PFS asynchronously, e.g. via the Spectral library on Summit [ORN].

4.2 BB Utilization

Concurrent access to PFS by all the compute nodes would introduce high overhead during checkpoints. Instead, BBs are utilized to provide faster access and lower contention depending on the BB architecture. On Summit, a BB device is attached to each compute node locally. In contrast, the BB system on NERSC's Cori [Bhi16] hosts a set of dedicated nodes, which form a cluster of BB hosting devices but require communication over the interconnect (Cray Dragonfly) to commit data to BBs. On Summit, each BB device has 1.6 TB capacity with up to 2.1 GB/sec write and 5.5 GB/sec read I/O bandwidth on a compute node [Vaz18].

There are two scenarios in which concurrent access to PFS leads to performance degradation. First, for every checkpoint, all the processes need to write checkpoint data concurrently to PFS. We resolve this issue by storing the checkpoint data in BBs and later bleeding it off to the PFS asynchronously, i.e., by limiting the number of nodes with BB to PFS transfer at any time. Second, for post-failure recovery of the application state, all processes need to access PFS concurrently. However, each checkpoint is stored to the local BB first. Hence, only the new node replacing the failed one needs to recover the checkpoint data from PFS. All other nodes recover from their local BB device.

Such a model leads to four possible flows of checkpoint data indicated in Figure 4.3: Regular checkpoints are written to local BB devices (1) and further bled off to the PFS (2); application state is recovered for non-failed nodes from BBs (3) and for the new node (replacing the failed one) from PFS (4).

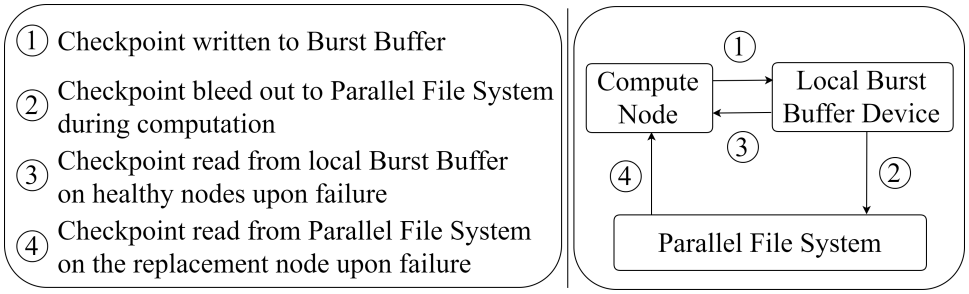

Figure 4.3 Checkpoint Dataflow

4.3 Adaptability

The schedule of checkpoints is variable in our model. It depends on factors such as the time of failure prediction, the location of the predicted failure, and the proactive action chosen. In Figure 4.4(A), the schedule of a subsequent checkpoint does not change when the predicted failure occurs after the checkpoint data is bled off to PFS. However, in Figure 4.4(B), the scheduled checkpoint is shifted as it would be incomplete due to a predicted failure that forced live migration. Similarly, in Figure 4.4(C), the scheduled checkpoint changes due to a safeguard checkpoint triggered by and completed before a predicted failure. Upon a failure that was not predicted, the schedule of checkpoints always changes, which is shown in Figure 4.4(D). Table 4.1 summarizes all scenarios handled by our checkpoint model.

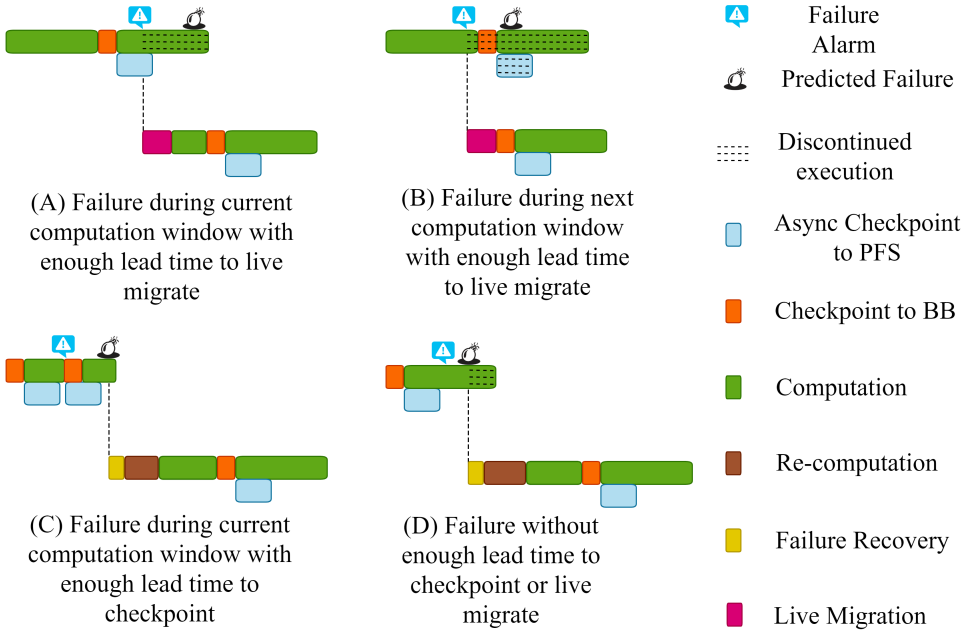

Figure 4.4 Adaptability under Different Scenarios

Table 4.1 Handling of Failure Prediction

Failure Location			**Lead Time Driven Action**	**Change in checkpoint schedule**
During checkpoint to BB	***During checkpoint bleed off to Parallel File System***	***During Computation***		
✓	✗	✗	Live Migration	✓
✗	✓	✗	Live Migration	✓
✗	✗	✓	Live Migration	✗
✓	✗	✗	Checkpoint	✗
✗	✓	✗	Checkpoint	✗
✗	✗	✓	Checkpoint	✓
✓	✗	✗	Imminent Failure	✓
✗	✓	✗	Imminent Failure	✓
✗	✗	✓	Imminent Failure	✓

4.4 Proactive Live Migration

Proactive live migration allows us to relocate the processes on a compute node that may fail based on our failure prediction to a new and healthy node. We make several assumptions regarding our proactive live migration technique. First, during most live migration interval, the process continues to execute (until a final frozen state is transferred) [Wan08; Wan12]. Second, we assume that the costs associated with the process of live migration, e.g., loading the application on one of the reserved nodes and making the changes required in the MPI runtime environment, are minimal and can thus be ignored. Finally, the total amount of data that needs to be transferred to complete live migration is upper bounded by the DRAM size on a compute node. On summit, DRAM memory is 512 GB per node [Vaz18]. We also experimentally determined that the inter-node bandwidth on Summit is ≈12.5 GB/sec. Hence, the maximum amount of time needed to migrate an application is 41 seconds.

CHAPTER

5

FAILURE ANALYSIS AND OPTIMAL CHECKPOINT INTERVAL

Table 5.1 Symbol Notations

Symbol	Description
T_w	Wall clock run time for job i
T_{cmpt}	Total computation time for job i
$T_{failure}$	Total loss of time due to failure of job i
t	Single iteration of computation and checkpoint to the BBs for job i
t_{cmpt}	Checkpoint interval for job i
t_{ckpt}^{bb}	Time required to write one checkpoint to the BBs by job i
t_{ckpt}^{pfs}	Time required to write one checkpoint to the PFS by job i
λ	Failure rate per node
θ	Time required to perform proactive live migration of an application
σ	Percentage of failures that can be predicted with lead time larger than θ
c	Number of compute nodes job i is running on

In this section, we derive a first order approximation of the optimal checkpoint interval for our

multi-level checkpoint model. Further, we integrate the rigorous analysis of failure logs into the optimal checkpoint interval formula. Table 5.1 provides a legend of the symbols used for deriving the optimal checkpoint interval with short descriptions.

The wall clock run time of a job supported by our C/R model is the aggregate of time spent in computation, checkpointing, node losses, and recovery due to the imminent and unpredicted failures as shown in Eq. (5.1).

$$T_w = T_{cmpt} + \left(\frac{T_{cmpt}}{t_{cmpt}} - 1 \right) t_{ckpt}^{bb} + T_{failure}. \tag{5.1}$$

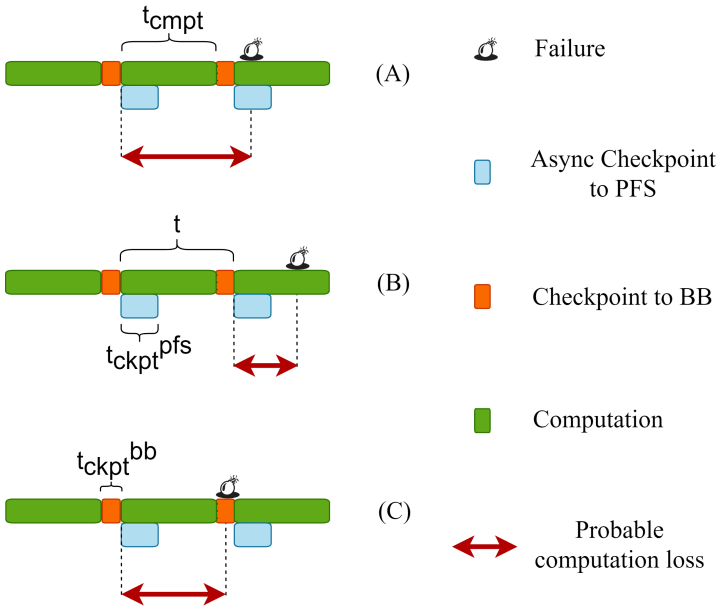

Figure 5.1 Computation loss scenarios when a failure occurs during (A) asynchronous checkpointing to PFS (B) computation post checkpointing to PFS (C) synchronous checkpointing to BB

In our multi-level model, the losses due to the failures can occur in three plausible phases during an iteration of computation and checkpointing as shown in Figure 5.1. A failure during the asynchronous checkpointing to the PFS (Figure 5.1(A)) requires recovery with the checkpoint data two iterations back. The failures during computation post completion of asynchronous checkpointing to the PFS (Figure 5.1(B)) and during checkpointing to the local BB (Figure 5.1(C)) need the checkpoint data from the last iteration. We also make the following assumptions while deriving the optimal checkpoint interval.

- Recovery time post failure is minimal and thus ignored.
- Failures happen only on one compute node.

- Failures arrive independent of each other following a Poisson distribution and $t \ll \frac{1}{\lambda}$. So the average number of failures is given by $\left(\frac{T_{cmpt}\, t\, \lambda\, c}{t_{cmpt}}\right)$.

The total amount of loss due to failure is expressed in Eq. (5.2).

$$\begin{aligned} T_{failure} = & \left(\left(\frac{t_{cmpt} - t_{ckpt}^{pfs}}{t}\right)\left(\frac{t_{cmpt} + t_{ckpt}^{pfs}}{2}\right) + \right. \\ & \left(\frac{t_{ckpt}^{pfs}}{t}\right)\left(t + \frac{t_{ckpt}^{pfs}}{2}\right) + \\ & \left.\left(\frac{t_{ckpt}^{bb}}{t}\right)\left(t - t_{ckpt}^{bb} + \frac{t_{ckpt}^{bb}}{2}\right)\right)\left(\frac{T_{cmpt}\, t\, \lambda\, c}{t_{cmpt}}\right). \end{aligned} \tag{5.2}$$

The term $\left(\frac{t_{cmpt} - t_{ckpt}^{pfs}}{t}\right)$ is the fraction of one iteration post asynchronous checkpointing to the PFS during computation and $\left(\frac{t_{cmpt} + t_{ckpt}^{pfs}}{2}\right)$ is the average loss of computation during this window. $\left(\frac{t_{ckpt}^{pfs}}{t}\right)$ is the fraction of one iteration when checkpointing to the PFS is in progress and $\left(t + \frac{t_{ckpt}^{pfs}}{2}\right)$ is the average loss during this period. Eq. (5.2) similarly captures the proportional loss during checkpointing to the local BBs.

Simplifying Eq. (5.2), we obtain

$$\begin{aligned} T_{failure} = & \left(t_{cmpt}^2 - {t_{ckpt}^{pfs}}^2 + \right. \\ & 2\left(t_{cmpt} + t_{ckpt}^{bb}\right) t_{ckpt}^{pfs} + {t_{ckpt}^{pfs}}^2 + \\ & \left. 2\left(t_{cmpt} + t_{ckpt}^{bb}\right) t_{ckpt}^{bb} - {t_{ckpt}^{bb}}^2\right)\left(\frac{T_{cmpt}\lambda c}{2 t_{cmpt}}\right), \end{aligned} \tag{5.3}$$

rewritten as

$$\begin{aligned} T_{failure} = & \left(t_{cmpt} + 2t_{ckpt}^{pfs} + \frac{2 t_{ckpt}^{bb} t_{ckpt}^{pfs}}{t_{cmpt}} \right. \\ & \left. + 2t_{ckpt}^{bb} + \frac{{t_{ckpt}^{bb}}^2}{t_{cmpt}}\right)\left(\frac{T_{cmpt}\lambda c}{2}\right). \end{aligned} \tag{5.4}$$

Replacing $T_{failure}$ in Eq. (5.1), the wall clock run time of a job is given by Eq. (5.5).

$$T_w = T_{cmpt} + \left(\frac{T_{cmpt}}{t_{cmpt}} - 1\right) t_{ckpt}^{bb} + \left(t_{cmpt} + 2t_{ckpt}^{pfs} + \frac{2t_{ckpt}^{bb} t_{ckpt}^{pfs}}{t_{cmpt}} + 2t_{ckpt}^{bb} + \frac{{t_{ckpt}^{bb}}^2}{t_{cmpt}} \right) \left(\frac{T_{cmpt} \lambda c}{2} \right). \tag{5.5}$$

Taking first derivative with respect to t_{cmpt} to minimize the wall clock time, we get

$$-\frac{t_{ckpt}^{bb} T_{cmpt}}{{t_{cmpt}^{opt}}^2} + \frac{T_{cmpt} \lambda c}{2} \left(1 - \frac{2t_{ckpt}^{pfs} t_{ckpt}^{bb}}{{t_{cmpt}^{opt}}^2} - \frac{{t_{ckpt}^{bb}}^2}{{t_{cmpt}^{opt}}^2} \right) = 0 \tag{5.6}$$

further simplified to

$${t_{ckpt}^{opt}}^2 = \frac{2t_{ckpt}^{bb}}{\lambda c} + {t_{ckpt}^{bb}}^2 + 2t_{ckpt}^{pfs} t_{ckpt}^{bb} \quad . \tag{5.7}$$

Ignoring the term ${t_{ckpt}^{bb}}^2$ as it is very small, we get

$${t_{ckpt}^{opt}}^2 = \frac{2t_{ckpt}^{bb}}{\lambda c} + 2t_{ckpt}^{pfs} t_{ckpt}^{bb} \quad . \tag{5.8}$$

This implies

$$t_{ckpt}^{opt} = \sqrt{\frac{2t_{ckpt}^{bb}}{\lambda c} + 2t_{ckpt}^{pfs} t_{ckpt}^{bb}} \quad . \tag{5.9}$$

The optimal checkpoint interval in our checkpoint model further includes a rigorous analysis of failures logs. The study analyzes the system logs collected from three real-world HPC systems from Desh [Das18a] over a period of six months. Using the Desh approach, the most common sequences of phrases in logs that may lead to failure are considered. Our assumption in this work is that any sequence of phrases, so-called failure chains, results in an actual failure. The time difference between the first phrase and the last phrase in a chain is considered as the lead time. Figure 5.2 shows the distribution of lead times as box plots for different failure instances (sequence 1-10), each of which occurs repeatedly in these logs. Failure sequence ID and number of occurrences in the logs are on the x-axis. The y-axis represents the lead time in seconds. Mean lead time, which is used as failure prediction lead time, is on the left side of each boxplot. We observe that most failures are bounded by the whiskers, only few outliers exist, with the exception of failures sequences 2 and 3. In the following experiments, we consider the mean lead time during simulation for simplicity. Notice

that this is quite conservative as we only cover 50% of the correctly predicted failures, as opposed to more aggressive assumptions with migration in 70% of the predicted failure cases [Wan08].

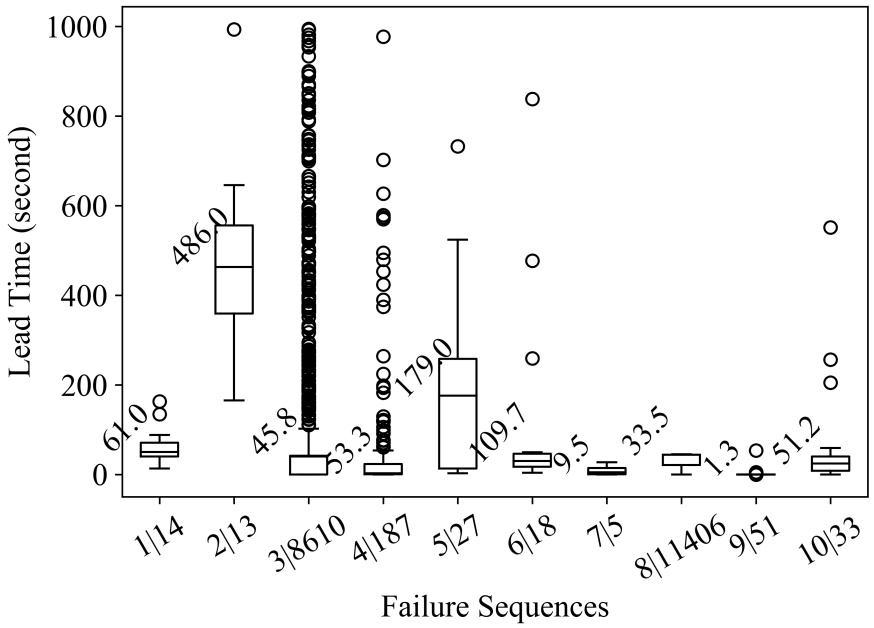

Figure 5.2 Failure Prediction Lead Time Distribution

Our study suggests that approximately 44 percent of the failures can be predicted with lead times in excess of the time required to live migrate a process from a faulty node to a new and healthy node. By considering a 44 percent decrease in the rate of failures, we further improve the optimal checkpoint interval.

From the failure analysis of logs, we obtain that σ percent of failures can be predicted with a lead time in excess of θ seconds (we use a threshold of 41 seconds for Summit) and thus can be avoided with proactive live migration. Hence, the failure rate is further reduced by σ percent and Eq. (5.9) can be re-written as Eq. (5.10).

$$t_{ckpt}^{opt} = \sqrt{\frac{2t_{ckpt}^{bb}}{\lambda c\,(1-\sigma)} + 2t_{ckpt}^{pfs}\,t_{ckpt}^{bb}} \tag{5.10}$$

CHAPTER

6

I/O PERFORMANCE MODEL

As our checkpoint model is developed for large HPC systems like Summit, an I/O performance model for the PFS is built based on Summit. PFS I/O performance plays an important part in checkpoint solutions. With higher I/O performance (latency and bandwidth), checkpoints can be stored faster. There are two reasons for the need for a PFS I/O performance model. First, the optimal checkpoint interval calculation depends on the latency to store a checkpoint to the PFS. Second, I/O performance is known to be variable due to I/O contention between different jobs. Even on modern HPC systems, I/O bandwidth for an application is severely impacted by concurrent I/O operations performed by other applications. This causes significant variability in I/O performance.

On Summit, IBM's SpectrumScale $GPFS^{TM}$ PFS handles application I/O using IBM's $GL4^{TM}$ Elastic Storage Servers as I/O nodes. The I/O subsystem evaluation in [Vaz18] shows an aggregate bandwidth of 2.5 TB/sec can be realized. However, the evaluation measures the performance of the I/O node server. It does not measure the I/O performance realized within an application. The objective here is to characterize the *actual* I/O performance seen by an application.

To characterize the I/O performance of the $GPFS^{TM}$, two experiments are conducted. The first experiment is performed to find the optimal number of MPI processes that can achieve maximum aggregate I/O bandwidth on a single compute node. A compute node on Summit has 42 physical cores, which can be scaled to 168 cores with hardware multithreading (SMT) technology. Further, the physical cores are evenly distributed over two sockets along with DRAM. This experiment measures the I/O bandwidth for different aggregate data transfer sizes over multiple MPI tasks from 1 to 42. These MPI tasks are evenly distributed over the two sockets on the compute node and use POSIX writes to transfer data. I/O buffers are flushed via the fsync() call to ensure that the data is not cached

but rather committed to the devices. Figure 6.1 depicts aggregate I/O bandwidth (y-axis) for different transfer sizes (x-axis) on curves ranging from 1 to 32 processes. These results indicate that 8 MPI tasks on a single compute node result in the maximum I/O bandwidth. Hence, 8 MPI tasks are used to store checkpoints in the PFS model.

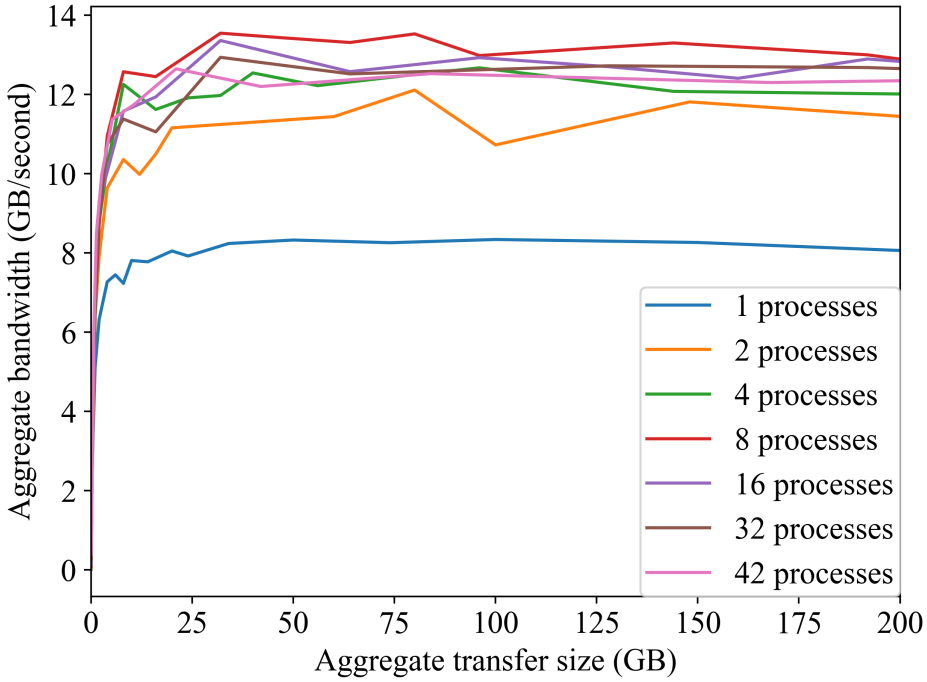

Figure 6.1 I/O Performance on a Single Compute Node

The second experiment assesses the effect of weak scaling on aggregate I/O bandwidth for different sizes of aggregate data transfer per node. 8 MPI tasks are used to perform I/O. Effectively, the I/O performance of the $GPFS^{TM}$ parallel file system is modeled. Figure 6.2 shows the effect of scaling (nodes on y-axis, transfer size on x-axis) on aggregate I/O bandwidth (indicated by the heat map). For strong scaling, a fixed data size (column) is exposed to increasing number of nodes. For weak scaling (diagonal from origin to top right corner), data size and number of nodes are increased at a constant rate. In our simulation, this performance matrix is used to calculate the time required to store checkpoint data in the PFS. Our simulation is based on the assumption that the aggregate bandwidth of a job is not affected by the I/O traffic generated by other running applications for now. For evaluation purposes, we assume the same performance matrix for the I/O read operations.

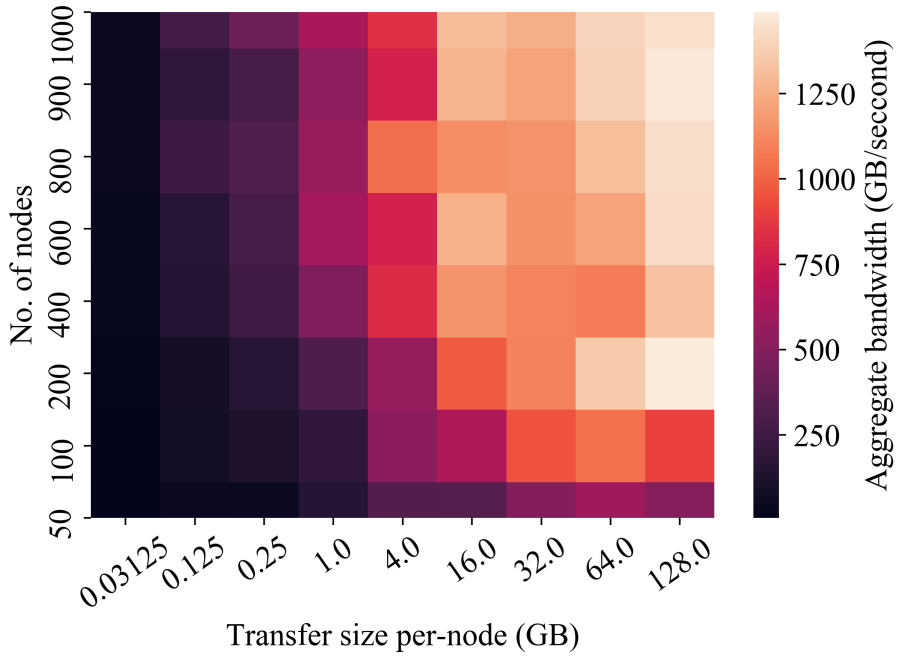

Figure 6.2 Impact of Scaling on I/O bandwidth

CHAPTER

7

EVALUATION

7.1 Simulation Framework

Our simulation framework developed using SimPy [Tea] emulates the system-level characteristics of Summit and the execution based on traces of the real-world scientific applications in Table 7.1. The aim is to evaluate our C/R model while simulating the run-time environment closely resembling the corresponding real-world HPC system. The applications used for our evaluation cover a wide range of job size, checkpoint data size, and total computation time.

Table 7.1 HPC Workload Characteristics

Application	Number of Nodes	Checkpoint Size (GB)	Computation Time (hour)
CHIMERA	2272	163840	360
XGC	1515	37926	240
S3D	505	5120	240
GYRO	126	50	120
POP	126	26	480
VULCAN	64	0.83	720

Previous studies [Tiw14; Sch10] have found that the mean time between failure (MTBF) distributions follow a Weibull distribution. In experiments, three sets of Weibull distribution parameters

from Wan et al. [Wan17] are considered (see Table 7.2) to generate failures during the simulation. The shape parameter with value less than one suggests that the failure rate decreases over time and that after a failure another failure is likely to happen within a short period of time. Since the actual failure statistics of Summit are unavailable, it is assumed that the failure distribution of OLCF's Titan also applies to Summit. LANL systems 8 and 18 are used to test our C/R model under different failure distributions. In experiments, failures are simulated following a given Weibull distribution. Upon a simulated node failure, a node is selected randomly from the set of all nodes, and the failure is simulated on that node. The number of failures for a job depends on the number of nodes an application run uses, i.e., given the MTBF rate, larger jobs (with more nodes) are subject to a high total number of failures. Simulation are repeated 1000 times, and the measurements are averaged over those 1000 runs.

Table 7.2 Weibull Distributions for Failure Generation

HPC System	Shape	Scale
LANL System 8 (164 nodes)	0.7111	67.375
LANL System 18 (1024 nodes)	0.8170	6.6293
OLCF Titan (18868 nodes)	0.6885	5.4527

7.2 C/R Model Evaluation

The evaluation compares four models to study the impact of the BBs and our failure prediction and analysis model assisted by adaptive safeguard checkpoints and proactive live migration:

- *Model A:* The base model does not incorporate any of these techniques. It also does not use BBs. We use the Young's standard formula [You74] as a basis to derive the optimal checkpoint interval.
- *Model B:* This model utilizes only BBs without any of the other techniques.
- *Model C:* This model utilizes the BBs, and failure analysis and prediction to decide on adaptive safeguard checkpoints.
- *Model D:* This model utilizes the BBs, and both adaptive safeguard checkpoints and live migration based on failure analysis and prediction.

Figure 7.1 depicts the overhead of fault tolerance in percent (y-axis) normalized to the base model A (first bar) for each application (x-axis) compared to models B, C and D.

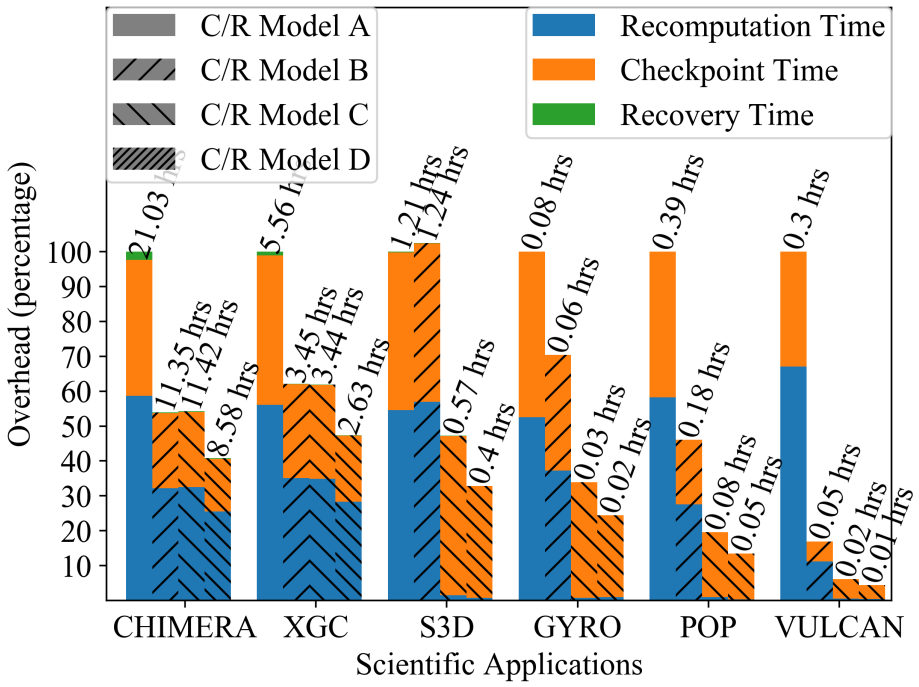

Figure 7.1 Reduction in Overhead with Failure Distribution from OLCF's Titan

Observation 1: Application overhead is reduced by ≈53%-95% due to (1) the reduction in checkpoint time with the assistance of BBs (model B), (2) reduction in failure rate given failure analysis

and prediction (3) combined with adaptive safeguard checkpoints (model C) and, in addition, (4) live migration (model D).

The overhead consists of three components depicted as stacked bars. Recovery time is required to restart an application after a node failure. Since only the new node requires to access the PFS to recover the checkpoint without contention while others recover from their local BB, the recovery time is insignificant and hardly visible in the graphs for all the models except A. We observe a ≈60%-99% decrease in recovery time for models B, C, and D.

Observation 2: Checkpoint overhead is reduced by the use of BBs and asynchronous checkpoints in ≈30%-82% of the cases for all the applications except for S3D. It is reduced further by ≈5%-30% due to the reduced failure rate with our failure analysis model.

The checkpoint time is the duration during which application execution is suspended while storing the checkpoints on permanent storage. In model A, the PFS is the permanent storage. BBs are used for the other models. We observe that due to higher latency to checkpoint the PFS in model A, all applications except for S3D spend more time in storing the checkpoints without BBs. For S3D and its checkpoint size, the PFS I/O bandwidth is almost equal to the I/O bandwidth of BBs. The checkpoint size for S3D is ≈10 GB per node. Following our I/O performance model (see Section 6), the aggregate bandwidth per node starts to saturate when the transfer size per node is around ≈10-12 GB. For smaller applications such as GYRO, POP, and VULCAN, the latency to write checkpoints to PFS is higher and results in more checkpoint overhead. Also, for larger applications such as CHIMERA and XGC, the higher latency for PFS writes is due to larger PFS contention.

Observation 3: While re-computation time dominates the base model, it can be reduced by ≈50%-55% for applications with large checkpoint sizes and up to ≈ 98% for those with smaller checkpoints, both for models B and C. BBs do not help in reducing re-computation overhead in the presence of failures for smaller applications.

The re-computation time is the duration spent by an application to re-compute the portion of execution to reach the point where execution was interrupted when the node failed. It contributes the largest fraction of overhead for all the applications in the base model. We observe that lack of BBs in the base model A results in a larger checkpoint interval (see Table 7.3) causing more losses due to failures. With BBs, the shorter checkpoint interval provides better tolerance from non-predicted failures. This reduces the re-computation overhead by ≈29%-83% for all applications except for S3D as its checkpoint interval is unaffected by BBs. In fact, it takes almost twice the time to store a checkpoint for S3D in model B compared to model A without the additional benefit of BBs. So the small increase in re-computation overhead for S3D in model B is because of the increased computation loss when a failure occurs during asynchronous bleed-off time.

We also observe that as the aggregate checkpoint size decreases the re-computation time also decreases. For CHIMERA and XGC, with larger checkpoints, the additional reduction in re-computation time is ≈12%. For the remaining applications except for S3D, the additional decrease is significantly higher, namely between 16% and 69%. This is an indirect effect of reduced latency to store checkpoints. The reduced time to store checkpoints just to the BB on the critical path and then bleeding

them off asynchronously helps our C/R model to handle more failures with proactive actions, i.e., adaptive safeguard checkpoints and live migration. For S3D, the reduction is ≈98% because of live migration or adaptive checkpoints, depending on the model. This suggests that smaller applications can achieve significant reductions in re-computation time without BBs.

Observation 4: Models C and D yield a similar reduction in re-computation time for applications with smaller checkpoint size, i.e., the added benefit of live migration is insignificant for these smaller checkpoints while live migration was relevant for larger checkpoints.

For applications such as S3D, GYRO, POP, and VULCAN, the difference in reductions between the adaptive safeguard checkpointing (model C) and adding live migration on top (model D) is less than 2%. Adaptive safeguard checkpoints result in less than 1% reduction in re-computation time for CHIMERA and XGC, respectively, which is due to higher latency to store the larger checkpoints.

There is a negligible change of less than 0.002% in time spent in storing checkpoints with adaptive safeguard checkpoints (model C) compared to the model B. What is more significant is the reduced failure rate resulting from the failure analysis model, which yields a ≈29% decrease in checkpoint time in adaptive safeguard checkpoints plus live migration (model D). The low variation in reduced checkpoint time across applications is due to the adaptive checkpoints that shift the schedule of periodic (base) checkpoints (see Figure 4.4C).

Observation 5: The reduction in checkpoint time further reduces the amount of checkpoint data written to BBs in an application by ≈ 29%, which increases the lifetime of BBs as they are subject to wear-out.

Figure7.2 depicts the aggregate amount of daily write traffic to the BB (y-axis) normalized to model B across all applications (x-axis) for the three models. Since model A excludes BBs, it is not under consideration. It indicated a ≈29% reduction in writes for model D. On Summit, each BB device has a daily write limit of 8 TB given its designation to last for 5 years. Model D effectively increases the longevity of the BBs by ≈41% assuming uniform daily writes across all the devices and assuming that BBs are used for checkpointing, only. Notice that model C is *not* subject to BB write reduction as safeguard checkpoints are committed to the BB. Once committed, the next periodic checkpoint is offset relative to the last safeguard checkpoint, which explains why model C only results in an insignificant change in writes of less than 0.5% over model D.

Observation 5: Adaptive safeguard checkpointing plus live migration (model D) results in a larger average checkpoint interval than other models.

This is evident from Table 7.3, which lists run-time application characteristics such as average checkpoint interval, and the average number of failures observed in 1000 simulation runs. The larger checkpoint interval of models A and B increases the risk of longer re-computation upon failure. However, failure prediction reduces that risk in models C and D via adaptive safe guard checkpointing and, additionally, live migration. The average number of failures for an application depends on its execution length. Longer jobs and jobs with larger checkpoints increase the risk of suffering from more failures for a given MTBF rate. For model D (adaptive safeguard checkpointing plus live migration), the average number of failures is lower as failures are avoided proactively

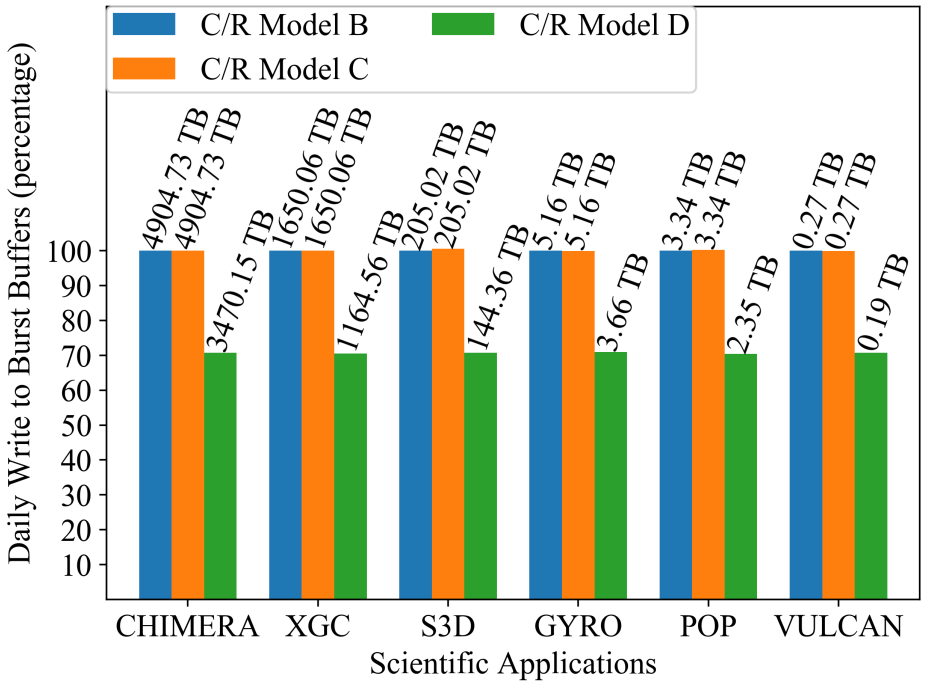

Figure 7.2 Reduction in Daily Writes to Burst Buffers

circumvented by live migration.

Table 7.3 Simulation Statistics

Application	**Average Checkpoint Interval (hour)**			**Average No. of Failures**			
	Model A	***Model B/C***	***Model D***	***Model A***	***Model B***	***Model C***	***Model D***
CHIMERA	1.43	0.79	1.13	15.81	14.83	15.15	8.49
XGC	0.87	0.55	0.78	7.03	6.82	6.98	3.87
S3D	0.60	0.61	0.83	2.24	2.35	2.26	1.30
GYRO	0.33	0.23	0.33	0.28	0.29	0.26	0.17
POP	0.42	0.19	0.27	1.08	1.11	1.15	0.59
VULCAN	0.45	0.07	0.10	0.89	0.87	0.83	0.48

7.3 Impact of Failure Locality

To study impact of locality of failures, our C/R model was exposed to two other failure distributions derived from data from LANL systems 8 and 18 [Sch06]. Here, we only compare the base model A and the adaptive safeguard checkpoint plus live migration model D.

Observation 6: Reductions in overheads for model D is robust across different Weibull failure distributions.

Figures 7.3 and 7.4 depict the reduction in overhead on the same x- and y-axes as before (Fig. 7.1 for Titan), yet for Systems 8 and 18, respectively. The reduction in overhead follows a similar pattern for all the three failure distributions. For LANL System 8 with its failure distribution, the decrease in overhead is ≈54%-95% while System 18 results in ≈52%-94% reduced overhead. Furthermore, the same pattern of increasing gains with decreasing checkpoint sizes is observed. This result is significant as it demonstrates that our model generalizes to other failure distributions. In principle, our C/R model can be deployed on any HPC system that supports BBs, live migration, and failure analysis plus prediction. It also shows that orchestrating failure prediction within a C/R model to drive decisions about when to checkpoint and when to live migrate (see Fig. 4.2) reduces the impact of of failures, shortens application execution over simpler failure models, and prolongs BB lifetime.

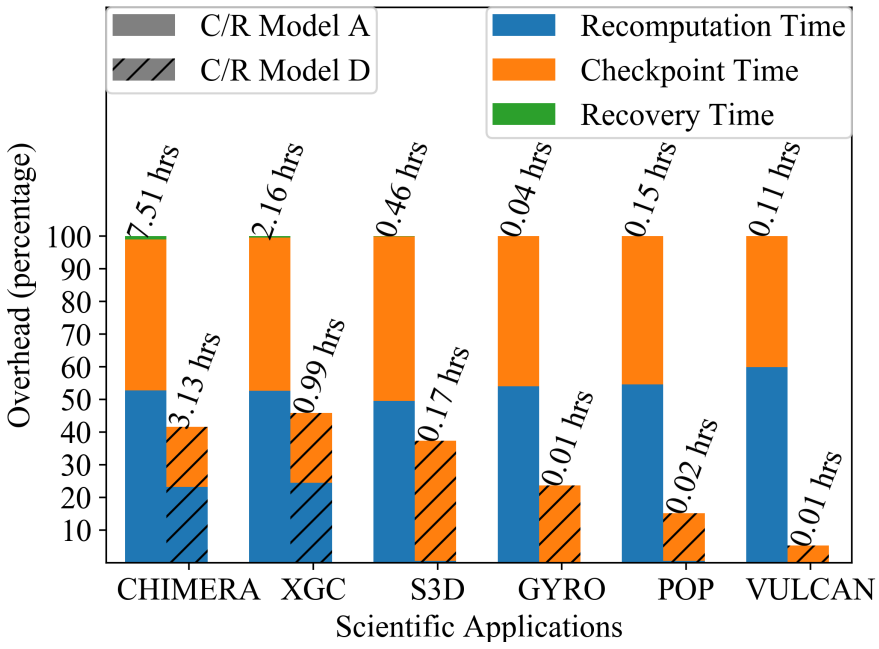

Figure 7.3 Reduction in Overhead with Failure Distribution of LANL System 8

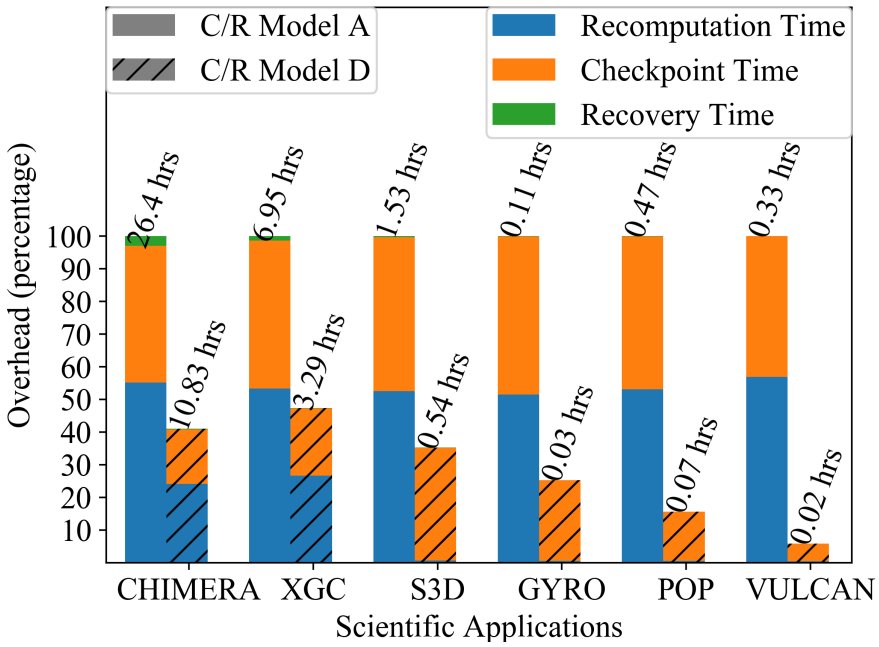

Figure 7.4 Reduction in Overhead with Failure Distribution of LANL System 18

CHAPTER

8

LIMITATIONS AND FUTURE WORK

Our work unifies several advanced methods to tackle the challenges of failures and PFS I/O congestion in HPC systems. However, it currently lacks some of the sensitivity studies such as assessing the impact of failure prediction lead time variability and I/O performance variability.

Further, our model still lacks the ability to handle failures with very low lead time prediction. For smaller sized applications, it is possible to handle failures with low lead time with safeguard checkpoints. However, for larger size applications, this becomes an issue as both live migration and safeguard checkpoint are not enough.

As part of our future work, we plan to improve our model with granular coordination and prioritization to handle the issue of very low lead time prediction. In short, in coordination with the job scheduler, a node with a failure prediction should be prioritized high when storing its checkpoint data on the Parallel File System. Only After that, other nodes can be allowed to store their checkpoint data. Further, we would like to evaluate our current model and the improved model with lead time and I/O performance variability.

CHAPTER

9

CONCLUSION

This work contributes a multi-level C/R model instantiated to resemble a Summit-like large-scale HPC system. (1) It integrates failure analysis (by analyzing system logs from three real-world HPC systems) and a prediction model, which predicts failure types and lead time that are subsequently used in optimal checkpoint interval derivation. (2) It issues adaptive safeguard checkpoints or live migration of an application from a faulty to node to a healthy, reserved node in face of failures. (3) It utilizes BBs to reduce PFS contention during application execution through asynchronous bleed off and for failure recovery by mandating only the replacement node to recover from a failure. (4) It derives a first-order approximation of the optimal checkpoint interval for our multi-level model. (5) It includes a PFS I/O performance model to characterize I/O bandwidth within applications on Summit, which is also used in optimal checkpoint derivation. The failure analysis model indicates that ≈ 44% of failures can be avoided using live migration and yields a ≈ 29% reduction in checkpoint time, which increases the lifetime of BBs. Adaptive safeguard checkpoints and proactive live migration effect the schedule of checkpoints and with multilevel asynchronous checkpoints supported by BBs result in ≈53%-95% reduced application overhead. Our C/R model is shown to be robust across multiple failure distributions.

Past work has studied failure/reliability-awareness to improve application efficiency. However, our C/R model's suitability for contemporary and future large-scale HPC systems, its applicability to wide range of applications, and its fault tolerance using advanced failure analysis and prediction model along with live migration are unprecedented in its combination with efficient and adaptive checkpointing, PFS I/O performance modeling, and robustness to failure distributions, which gives it a significant advantages over prior solutions.

BIBLIOGRAPHY

[Ahn18] Ahn, D. et al. “Flux: Overcoming Scheduling Challenges for Exascale Workflows”. 2018, pp. 10–19.

[Ben17] Benoit, A. et al. “Towards Optimal Multi-Level Checkpointing”. *IEEE Transactions on Computers* **66**.7 (2017), pp. 1212–1226.

[Bhi16] Bhimji, W. et al. “Accelerating Science with the NERSC Burst Buffer Early User Program”. 2016.

[Bou13] Bouguerra, M. et al. “Improving the Computing Efficiency of HPC Systems Using a Combination of Proactive and Preventive Checkpointing”. *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. 2013, pp. 501–512.

[Che13] Chen, Z. “Online-ABFT: An Online Algorithm Based Fault Tolerance Scheme for Soft Error Detection in Iterative Methods”. *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 2013, pp. 167–176.

[Cot06] Coti, C. et al. “Blocking vs. Non-Blocking Coordinated Checkpointing for Large-Scale Fault Tolerant MPI”. *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*. SC ’06. Tampa, Florida: Association for Computing Machinery, 2006, 127–es.

[Das18a] Das, A. et al. “Desh: Deep Learning for System Health Prediction of Lead Times to Failure in HPC”. *Symposium on High Performance Distributed Computing*. 2018, pp. 40–51.

[Das18b] Das, A. et al. “Doomsday: Predicting Which Node Will Fail When on Supercomputers”. *Supercomputing*. 2018, 9:1–9:14.

[Das18c] Das, A. et al. “KeyValueServe: Design and performance analysis of a multi-tenant data grid as a cloud service”. *Concurrency and Computation: Practice and Experience* **30**.14 (2018). e4424 cpe.4424, e4424–n/a.

[Di14] Di, S. et al. “Optimization of Multi-level Checkpoint Model for Large Scale HPC Applications”. *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. 2014, pp. 1181–1190.

[Di17] Di, S. et al. “Toward an Optimal Online Checkpoint Solution under a Two-Level HPC Checkpoint Model”. *IEEE Transactions on Parallel and Distributed Systems* **28**.1 (2017), pp. 244–259.

[Du17] Du, M. et al. “DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning”. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’17. Dallas, Texas, USA: Association for Computing Machinery, 2017, 1285–1298.

[Ell12] Elliot, J. et al. “Combining Partial Redundancy and Checkpointing for HPC”. *International Conference on Distributed Computing Systems*. 2012.

[Eln08] Elnozahy, E. N. M. et al. *System Resilience at Extreme Scale*. Tech. rep. Defense Advanced Research Project Agency (DARPA), 2008.

[Fan15] Fang, A. & Chien, A. A. "How Much SSD Is Useful for Resilience in Supercomputers". *Proceedings of the 5th Workshop on Fault Tolerance for HPC at eXtreme Scale*. FTXS '15. Portland, Oregon, USA: ACM, 2015, pp. 47–54.

[Fer09] Ferreira Kurt, R. R. O. R. S. J. L. J. P. K. K. T. & Brightwell, R. "Increasing fault resiliency in a message-passing environment" (2009).

[Fia11] Fiala, D. et al. "Detection and Correction of Silent Data Corruption for Large-Scale High-Performance". *SC Poster Session*. 2011.

[Gai12] Gainaru, A. et al. "Fault Prediction under the Microscope: A Closer Look into HPC Systems". *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC '12. Salt Lake City, Utah: IEEE Computer Society Press, 2012.

[Gar18] Garg, R. et al. "Shiraz: Exploiting System Reliability and Application Resilience Characteristics to Improve Large Scale System Throughput". *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2018, pp. 83–94.

[Gei03] Geist, A. & Engelmann, C. "Development of Naturally Fault Tolerant Algorithms for Computing on 100,000 Processors" (2003).

[Geo15] George, C. & Vadhiyar, S. "Fault Tolerance on Large Scale Systems using Adaptive Process Replication". *IEEE Transactions on Computers* **64**.8 (2015), pp. 2213–2225.

[Gup17] Gupta, S. et al. "Failures in Large Scale Systems: Long-Term Measurement, Analysis, and Implications". *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '17. Denver, Colorado: Association for Computing Machinery, 2017.

[Har06] Hargrove, P. H. & Duell, J. C. "Berkeley lab checkpoint/restart (BLCR) for Linux clusters". *Journal of Physics: Conference Series* **46** (2006), pp. 494–499.

[Isk08] Iskra, K. et al. "ZOID: I/O-forwarding Infrastructure for Petascale Architectures". *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. PPoPP '08. Salt Lake City, UT, USA: ACM, 2008, pp. 153–162.

[Khe19] Khetawat, H. et al. "Evaluating Burst Buffer Placement in HPC Systems". *IEEE Cluster*. 2019.

[Kua84] Kuang-Hua Huang & Abraham, J. A. "Algorithm-Based Fault Tolerance for Matrix Operations". *IEEE Transactions on Computers* **C-33**.6 (1984), pp. 518–528.

[Liu12] Liu, N. et al. "On the role of burst buffers in leadership-class storage systems". *2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*. 2012, pp. 1–11.

[Liu08] Liu, Y. et al. “An optimal checkpoint/restart model for a large scale high performance computing system”. *2008 IEEE International Symposium on Parallel and Distributed Processing*. 2008, pp. 1–9.

[Luc14] Lucas, R. et al. “DOE Advanced Scientific Computing Advisory Subcommittee (ASCAC) Report: Top Ten Exascale Research Challenges” (2014).

[Moo10] Moody, A. et al. “Design, Modeling, and Evaluation of a Scalable Multi-Level Checkpointing System”. *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’10. USA: IEEE Computer Society, 2010, 1–11.

[ORN] ORNL. *Spectral Library*. URL: `https://www.olcf.ornl.gov/spectral-library/`.

[Pla93] Plank, J. S. & Li, K. *Faster Checkpointing with N + 1 Parity*. Tech. rep. USA, 1993.

[Sat12] Sato, K. et al. “Design and modeling of a non-blocking checkpointing system”. *SC ’12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. 2012, pp. 1–10.

[Sat14] Sato, K. et al. “A User-Level InfiniBand-Based File System and Checkpoint Strategy for Burst Buffers”. *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. 2014, pp. 21–30.

[Sch06] Schroeder, B. & Gibson, G. “A large-scale study of failures in high-performance computing systems”. *Proceedings of the 2006 International Conference on Dependable Systems and Networks (DSN-2006)*. Philadelphia, PA, 2006.

[Sch07] Schroeder, B. & Gibson, G. A. “Understanding failures in petascale computers”. *Journal of Physics: Conference Series* **78** (2007), p. 012022.

[Sch10] Schroeder, B. & Gibson, G. A. “A Large-Scale Study of Failures in High-Performance Computing Systems”. *IEEE Transactions on Dependable and Secure Computing* **7**.4 (2010), pp. 337–350.

[Tea] Team, S. *SimPy: Discrete-Event Simulation for Python*. URL: `https://pypi.org/project/simpy/`.

[Tiw14] Tiwari, D. et al. “Lazy Checkpointing: Exploiting Temporal Locality in Failures to Mitigate Checkpointing Overheads on Extreme-Scale Systems”. *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 2014, pp. 25–36.

[Vaz18] Vazhkudai, S. S. et al. “The Design, Deployment, and Evaluation of the CORAL Pre-Exascale Systems”. Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC ’18. Dallas, Texas: IEEE Press, 2018.

[Wan17] Wan, L. et al. “Optimizing checkpoint data placement with guaranteed burst buffer endurance in large-scale hierarchical storage systems”. *Journal of Parallel and Distributed Computing* **100** (2017), pp. 16 –29.

[Wan09] Wang, C. et al. *Hybrid Full/Incremental Checkpoint/Restart for MPI Jobs in HPC Environments*. Tech. rep. TR 2009-14. Dept. of Computer Science, North Carolina State University, 2009.

[Wan08] Wang, C. et al. “Proactive Process-Level Live Migration in HPC Environments”. *SC ’08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. 2008.

[Wan12] Wang, C. et al. “Proactive Process-Level Live Migration and Back Migration in HPC Environments”. *Journal of Parallel Distributed Computing* **72**.2 (2012), pp. 254–267.

[Wan14] Wang, T. et al. “BurstMem: A high-performance burst buffer system for scientific applications”. *2014 IEEE International Conference on Big Data (Big Data)*. 2014, pp. 71–79.

[Yoo03] Yoo, A. B. et al. “SLURM: Simple Linux Utility for Resource Management”. *Job Scheduling Strategies for Parallel Processing*. Ed. by Feitelson, D. et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60.

[You74] Young, J. W. “A first order approximation to the optimum checkpoint interval”. *Commun. ACM* **17**.9 (1974), pp. 530–531.

[Yu16] Yu, X. et al. “CloudSeer: Workflow Monitoring of Cloud Infrastructures via Interleaved Logs”. *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS ’16. Atlanta, Georgia, USA: Association for Computing Machinery, 2016, 489–502.