Modeling the Impact of Checkpoints on Next-Generation Systems

Ron A. Oldfield
Sandia National Laboratories[*]
raoldfi@sandia.gov

Sarala Arunagiri
The University of Texas at El Paso[†]
sarunagiri@utep.edu

Patricia J. Teller
The University of Texas at El Paso[†]
pteller@utep.edu

Seetharami Seelam
IBM TJ Watson Research Center
sseelam@us.ibm.com

Maria Ruiz Varela
The University of Texas at El Paso[†]
mdruizvarela@miners.utep.edu

Rolf Riesen
Sandia National Laboratories[*]
rolf@sandia.gov

Philip C. Roth
Oak Ridge National Laboratory[‡]
rothpc@ornl.gov

Abstract

The next generation of capability-class, massively parallel processing (MPP) systems is expected to have hundreds of thousands of processors. For application-driven, periodic checkpoint operations, the state-of-the-art does not provide a solution that scales to next-generation systems. We demonstrate this by using mathematical modeling to compute a lower bound of the impact of these approaches on the performance of applications executed on three massive-scale, in-production, DOE systems and a theoretical petaflop system. We also adapt the model to investigate a proposed optimization that makes use of "lightweight" storage architectures and overlay networks to overcome the storage system bottleneck. Our results indicate that (1) as we approach the scale of next-generation systems, traditional checkpoint/restart approaches will increasingly impact application performance, accounting for over 50% of total application execution time; (2) although our alternative approach improves performance, it has limitations of its own; and (3) there is a critical need for new approaches to fault tolerance that allow continuous computing with minimal impact on application scalability.

[*]Sandia National Laboratories is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

[†]Work at UTEP is supported by the Department of Energy under Grant No. DEFG02-04ER25622, Sandia National Laboratories under Contract No. 579987 (PR 882412), and an IBM SUR grant.

[‡]This research is sponsored by the Office of Advanced Scientific Computing Research; U.S. Department of Energy. The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. De-AC05-00OR22725.

1. Introduction

Today's high-end massively parallel processing (MPP) systems have tens of thousands of compute nodes. For example, consider the following MPP systems currently in use at three Department of Energy (DOE) laboratories. The "Red Storm" system, a Cray XT3 machine at Sandia National Laboratories (SNL) [6], has almost thirteen thousand compute nodes; "Jaguar", a Cray XT3/4 hybrid machine at Oak Ridge National Laboratory (ORNL), has more than eleven thousand compute nodes; and the IBM BlueGene/L [33] at Lawrence Livermore National Laboratory (LLNL), has over sixty-four thousand compute nodes. All of these machines are expected to be the computation target of large-scale applications that consume large fractions of the system. For example, in the original allocation policy description for Red Storm, 80% of the compute-node hours were to be used by applications that use a minimum of 40% of the compute nodes.

The massive scale of current and next-generation MPP systems and their supported applications present significant challenges related to fault tolerance. Some challenges arise because current "in-practice" approaches to fault tolerance do not match well with the expected demands or us-

24th IEEE Conference on Mass Storage Systems and Technologies (MSST 2007)
0-7695-3025-7/07 $25.00 © 2007 IEEE

age models of these systems. For example, the most commonly used approach to fault tolerance is "checkpoint-to-disk". Using this approach, an application (or system) periodically outputs to disk an amount of data that is sufficient to restart the application after a failure. As the size of applications grow with the number of compute nodes on which they execute, the cost of the checkpoint-to-disk approach increases due to a combination of three reasons. First, scientific applications (in particular, applications at the DOE laboratories) often use a large fraction of the memory available on each of the compute nodes, preventing the use of client-side caching to overlap computation and I/O. Second, without caching, the rate at which checkpoint data can be output is, at best, the speed of the storage system, which is typically an order of magnitude slower than the interconnection network. Third, as the number of employed compute nodes increases, so does the probability of application failure; this causes the application to checkpoint more frequently. The compounding effect of the increase in checkpoint data and the increase in the frequency of checkpoint operations with the number of employed compute nodes results in checkpoint operations for large-scale applications that generate bursts of I/O that can overwhelm an I/O system and severely impact the execution time of the application. The significance of these points is illustrated by the fact that even on today's systems, the I/O generated by checkpoints consumes nearly 80% of the total I/O usage [23]. Given the expense of the checkpoint-to-disk approach and the trends to develop systems of ever-increasing size, the community must seriously evaluate alternatives to traditional disk-based checkpointing.

The checkpoint-to-memory approach [31] is one such alternative. Its goal is to reduce the application execution time associated with checkpoint operations, i.e., the perceived checkpoint latency or checkpoint overhead, as we refer to it in this paper. In contrast, the checkpoint latency is the time required to write checkpoint data to persistent storage. When some of the checkpoint latency can be hidden, the impact on application execution time, i.e., the checkpoint overhead, is reduced. For the checkpoint-to-memory approach, this is done by having compute nodes use their local memories to manage the state of an application executed on other compute nodes. Because network and memory (network/memory) bandwidths are typically much faster than storage system bandwidths, this approach significantly reduces the checkpoint overhead. However, checkpoint-to-memory approaches may have problems with parity computation [27]: If an application node computes the parity, the advantage gained by using the network/memory bandwidth, instead of the storage bandwidth, decreases significantly [24]. But the biggest problem with checkpoint-to-memory approaches for large-scale scientific applications is the amount of memory resources required by the compute nodes. Even in today's systems, many large-scale scientific applications are constrained by the size of the memory on the compute nodes. Consequently, checkpoint-to-memory approaches are impractical for large-scale systems.

A number of other approaches are proposed in the literature, for example [7], but they have not been adopted by the scientific community. This lack of interest among developers of large-scale applications is likely due to the fact that the scale of systems is not yet large enough to justify a change. As we show in Section 3.3, through mathematical modeling and analysis, checkpoint operations only become an I/O problem and, thus, an application performance problem, when MPP systems reach scales at and beyond the largest existing systems. Because of this, the application-directed, checkpoint-to-disk approach still is the most widely used fault-tolerance approach in high-performance computing (HPC). However, if technology trends continue along the same path being followed today, we soon will have systems with hundreds of thousands (perhaps millions) of compute nodes. Accordingly, if the reliability of hardware and systems software do not improve substantially, checkpoint-to-disk soon will become impractical for large-scale applications.

In this paper, using Daly's model [10], we approximate the impact of the checkpoint-to-disk approach on the execution time of large-scale applications run on current and next-generation MPP systems, i.e., SNL's Red Storm, LLNL's BlueGene/L, ORNL's Jaguar, and a target petaflop system modeled after Red Storm. We then use the execution time model, with our own model of checkpoint latency, to demonstrate the performance impact of an alternate approach to checkpointing [19] that employs lightweight storage architectures [20] and overlay networks [15, 29], which we call the LWFS+overlay approach. Although the analysis of the results indicate that the LWFS+overlay approach has a smaller impact on application execution time than does checkpoint-to-disk, it indicates a critical need for new research in checkpoint/restart approaches for massive-scale systems.

The contributions of this paper include:

- a general analytical model for checkpoint overhead that takes into account per-node link bandwidth, bi-section network bandwidth, and storage system bandwidth;
- a proposed method to reduce checkpoint overhead by using lightweight storage architectures and overlay networks;
- a refined analytical model that approximates the overhead associated with checkpoint operations that use lightweight storage architectures and overlay networks;
- the incorporation of our general and refined models for checkpoint overhead into Daly's equation for the optimal checkpoint interval; and

- an approximation and analysis of the modeled checkpoint overheads for three, in-production, massive-scale systems and one proposed petaflop system.

The remainder of the paper is organized as follows. Section 2 provides background information on checkpoint/restart approaches and discusses related modeling efforts. Section 3 presents Daly's model, describes our model of checkpoint overhead, and presents an analysis of checkpoint overheads for four massive-scale systems. Section 4 describes and presents performance results for checkpoint operations that use lightweight storage architectures and overlay networks. Finally, Sections 5 and 6 discuss future work and summarize the paper.

2. Background and Related Work

This section provides background information and a summary of related work that justify our choice of models, assumptions, and research direction. The following four subsections discuss terminology used in this paper, checkpoint/restart mechanisms, models of some of these mechanisms, and research that targets the reduction of checkpoint overhead and latency.

2.1. Terminology

For clarity, we define the following terminology:

Checkpoint data: An application's state, i.e., data that are sufficient to restart the application in case of a failure.

Checkpoint: An operation performed by an application or a system to save the application's state. This paper focuses on application-directed, rather than system-directed, checkpoint operations.

Checkpoint latency: The amount of time required to write checkpoint data to persistent storage.

Checkpoint overhead: The amount of application execution time used to perform a checkpoint operation. Note that this may not be the same as the checkpoint latency.

Checkpoint interval: The application execution time between two consecutive checkpoint operations. The optimal checkpoint interval results in the minimum application execution time, including the time to redo the work performed between the last checkpoint and a failure.

Compute node: A device that is attached to the interconnection network of a supercomputer and is specifically designed to perform computation for large-scale applications. A compute node contains one or more processors and, on massive-scale systems, is often diskless.

Processor: A processing "core" of a compute node. In this paper, we assume that the application uses each employed compute-node processor for application computation. We do not assume symmetric multiprocessing capabilities on a compute node.

2.2. Checkpoint/Restart Mechanisms

Depending on when processes checkpoint, with respect to when other processes checkpoint, checkpointing mechanisms are classified as coordinated, uncoordinated, or communication-induced. In coordinated checkpointing, an application's processes must arrive at a consistent state before a checkpoint operation is performed; this makes the approach complex with respect to synchronization, but it simplifies recovery. In uncoordinated checkpointing, processes perform checkpoint operations independent of one another, thus, the restart operation is more complex. However, since there is no coordination with other processes, there is no associated synchronization overhead and, as a result, uncoordinated checkpointing is faster than coordinated checkpointing. However, uncoordinated checkpointing is prone to rollback propagation, which requires processes to store multiple checkpoint states, and, consequently, requires garbage collection in order to eliminate unnecessary checkpoint data. In communication-induced checkpointing, communication patterns trigger checkpoint operations. At times during application execution, checkpointing is uncoordinated, while at other times, communication patterns trigger processes to perform required checkpoint operations. For large-scale applications, coordinated checkpointing is the most widely used fault-tolerance mechanism. It is favored over uncoordinated checkpointing, for which better performance comes at the cost of increased restart complexity, memory overhead, and undesirable side effects, such as the domino effect [11]. Checkpointing methods also are classified based on the type of stable storage that is used to save application state, e.g., persistent or volatile storage, and the initiator of checkpoint operations, e.g., application-directed or system-initiated checkpointing. This paper focuses on large-scale applications and MPP systems. In this context, we study coordinated, application-directed, periodic, checkpoint-based, fault-tolerance methods that write application state to persistent storage.

2.3. Models of Checkpoint/Restart Mechanisms

Several models that define the optimal checkpoint interval have been proposed in the literature. Young proposed a first-order model that defines the optimal checkpoint interval in terms of checkpoint overhead and mean time to interrupt (MTTI) [39]. Young's model does not

consider failures during checkpointing and recovery, while Daly's extension of Young's model, a higher-order approximation, does [10]. In addition to considering checkpoint overhead and MTTI, the model in [32] includes sustainable I/O bandwidth as a parameter and uses Markov processes to model the optimal checkpoint interval. The model in [22] uses useful work, i.e., computation that contributes to job completion, to measure system performance. The authors claim that Markov models are not sufficient to model useful work and propose the use of Stochastic Activity Networks (SAN) to model coordinated checkpointing for large-scale systems. Their model considers synchronization overhead, failures during checkpointing and recovery, and correlated failures. This model also defines the optimal number of processors that maximize the amount of total useful work. Vaidya models the checkpointing overhead of a uniprocess application. This model also considers failures during checkpointing and recovery [36]. To evaluate the performance and scalability of coordinated checkpointing in future large-scale systems, [13] simulates checkpointing on several configurations of a hypothetical petaflop system. Their simulations consider the node as the unit of failure and assume that the probability of node failure is independent of its size, which is overly optimistic.

Although the models presented in this section differ in many respects, all but [22] assume that system and processor failures are independent and exponentially distributed; however, a recent study of failures on systems at Los Alamos National Laboratory suggests that emperical evidence may not match this assumption [30]. The analysis in our paper is based on Daly's model, which also makes this assumption.

2.4. Reducing Checkpoint Overhead

Several techniques target the reduction of checkpoint overhead, i.e., the time added to application execution time as a result of checkpointing. Some of these techniques are meant to hide some of the checkpoint latency and, thus, reduce checkpoint overhead. Copy-on-write checkpoint algorithms take advantage of the low-latency of memory; they copy checkpoint data to a separate memory address space via virtual-memory, page-protection hardware. Once a memory-to-memory transfer is complete, the checkpoint data are saved to stable storage while application execution continues. Copy-on-write algorithms can be improved by adding a buffering capability to enable the overlapping of memory-to-memory transfers of checkpoint data and the writing of the data to stable storage [18]. Although copy-on-write implementations slightly increase checkpoint latency, they decrease checkpoint overhead [12]. Since applications executing on MPP systems use large fractions of the available memory, copy-on-write and checkpoint-to-memory approaches [26], discussed briefly in Section 1, are not suitable for such systems.

The following techniques explicitly target the reduction of checkpoint latency. The use of RAID techniques has been proposed to store coordinated checkpoint data more efficiently [25]. RAID-inspired techniques, such as checkpoint mirroring, N+1 parity, and Reed-Solomon coding, are aimed at minimizing the impact of checkpointing on shared resources, e.g., I/O and network bandwidth, and on reducing checkpoint latency and recovery time [35]. Incremental checkpointing aims at reducing the size of checkpoint data by saving only the memory that has been touched since the last checkpoint operation. Page-based incremental checkpointing requires paging support from hardware and the operating system. Page-based techniques might not scale well on large MPP systems since even if only one bit in a page changes, the entire page must be saved; also, paging is not made use of on many MPP systems. Hash-based, as opposed to page-based, techniques are able to identify bytes changed in a page. This feature is used in [1] to propose an adaptive incremental checkpointing algorithm that aims at minimizing the amount of checkpoint data saved to stable storage. This algorithm uses a secure hashing function to dynamically identify a block corresponding to the approximate number of bytes changed in memory.

The LWFS+overlay approach to checkpointing, introduced in Section 1 and discussed in more detail in Section 4, reduces checkpoint overhead by buffering checkpoint data at network bandwidths rather than storage bandwidths. An important benefit of this approach for applications targeted at MPP systems is that it does not require additional memory resources on the compute nodes, a limitation that makes checkpoint-to-memory, asynchronous checkpoint, and incremental checkpoint approaches impractical for many large-scale applications [24]. The techniques described above that specifically target checkpoint latency and are applicable to MPP systems can be incorporated into the LWFS+overlay approach.

3. Modeling Checkpoint Performance

The model used in this paper is based on a detailed mathematical model of wall clock application execution time on a computer system that exhibits Poisson single component failures [8, 9, 10]. In this model, which was constructed by John Daly, execution time includes the time to perform checkpoints and the time to redo the work performed between the last checkpoint and a failure, i.e., *rework time*. For long-running applications, the execution time (T) is:

$$T = Me^{R/M}\left(e^{(\tau+\delta)/M} - 1\right)\frac{T_s}{\tau} \text{ for } \delta << T_s, \quad (1)$$

0-7695-3025-7/07 $25.00 © 2007 IEEE

where

T_s = Time spent doing application computation,
τ = Time spent doing work between checkpoints, i.e., *checkpoint interval*,
δ = Time to output checkpoint data, i.e., *checkpoint overhead*,
M = Mean time between interruptions (MTTI) of the application, and
R = Rework time.

By minimizing the application execution time in this equation, Daly in [10] derives the following approximation for the optimal checkpoint interval, τ_{opt-D}, which depends on the checkpoint overhead (δ) and the mean time between interruptions (MTTI) of the application (M).

$$\tau_{opt-D} = \begin{cases} \sqrt{2\delta M}\left[1+\frac{1}{3}\left(\frac{\delta}{2M}\right)^{\frac{1}{2}}+\frac{\delta}{9\cdot 2M}\right]-\delta & \delta < 2M \\ M & \delta \geq 2M. \end{cases} \quad (2)$$

3.1. Checkpoint Overhead Model

In the remainder of the paper, we use Daly's models of application execution time and optimal checkpoint interval to estimate the performance impact of application-directed, checkpoint-to-disk strategies that either write directly to the storage system (the current state-of-the-art) or write to an intermediate level of memory, from which the checkpoint data is automatically written to persistent storage. In either case, the checkpoint overhead, δ, which does not consider network or storage system contention, is approximated by the following model:

$$\delta = \alpha_c + \frac{nd}{\beta_{chkpt}},$$

where

α_c = Start-up cost (e.g., creating files) associated with a checkpoint operation,
n = Number of processors used by the application,
d = Amount of data written by each processor,
β_{chkpt} = Perceived bandwidth of a checkpoint operation, i.e., $\min(n\beta_L, \beta_n, \beta_s)$,
β_L = One-way network bandwidth per link,
β_n = Aggregate bisection network bandwidth, and
β_s = Aggregate storage system bandwidth.

The start-up cost of a checkpoint operation, α_c, heavily depends on the employed checkpoint algorithm, e.g., whether a shared file or a file per process is employed. In POSIX-compliant parallel filesystems, consistency semantics and device conflicts contribute to poor performance when writing to a shared file. Considering this negative performance impact, many DOE applications create a file per process for checkpoint/restart files. This improves write performance but creates a significant overhead associated with sending thousands of simultaneous create operations through a centralized metadata server [20].

With respect to the amount of data written by each processor, i.e., d, this model for checkpoint overhead assumes that each processor writes the same amount of state to the checkpoint/restart file(s) and that the parallel filesystem is perfectly scalable. These are overly optimistic assumptions but they still provide a reasonable lower bound on the performance impact of checkpoint operations.

The perceived checkpoint latency in the model takes into account the fact that the compute nodes of most MPP systems are diskless, meaning that all I/O requests travel through a communication network to access the storage devices. Thus, a checkpoint operation is bound by the aggregate network link bandwidth, bisection network bandwidth, or storage system bandwidth. The model accounts for these three different checkpoint bandwidth (β_{chkpt}) possibilities.

3.2. Checkpoint Overhead Model Parameters

In this paper, we model the performance of a representative, scientific, parallel application on four MPP architectures: SNL's Red Storm, LLNL's BlueGene/L (BG/L), ORNL's Jaguar, and a theoretical petaflop system, all scaled to 128K compute-node processors, where a node is comprised of multiple cores/processors. The representative parallel application employs all of the compute-node processors in a system and checkpoints half of each processor's available memory at periodic intervals equal to the optimal checkpoint interval computed using Daly's model. The memory available to each processor is the total memory of a compute node divided by the number of processors. Anecdotal evidence gathered from conversations with computational scientists who use DOE systems indicates that checkpoint data comprises 10-50% of the data in each compute node's memory; we chose to use 50% to cover the most data-intensive case.

Table 1 shows the model parameters for each of the four systems studied. The $n_{max} \times cores$ parameter is the number of compute-node processors in the system. It is the product of the number of nodes and the number of processors per node in the system[1]. d_{max} is the total memory available to

[1] Our model assumes that the application uses every compute-node processor as if it were independent; we do not assume SMP compute nodes. This is known as "virtual node" mode on the Cray XT systems at SNL and ORNL.

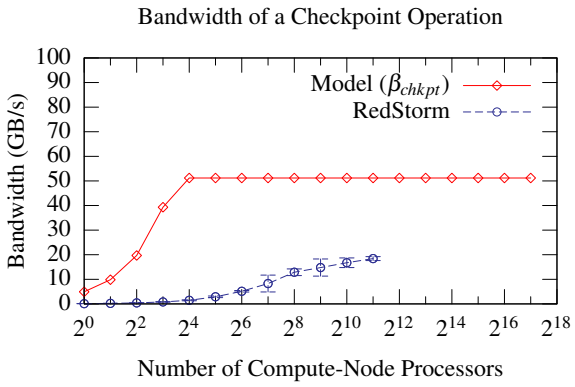

Figure 1. Comparison between the modeled and actual performance of checkpointing for LLNL's IOR benchmark writing to a file per process on SNL's Red Storm.

a processor. M_{dev} is the expected MTTI of any compute node in the system; see Section 3.2.5 for a description of how it is calculated. Our parameters for link and bisection bandwidths (β_n, β_L) indicate hardware rates reported by vendors; they do not include the overheads associated with message headers, encoding, or integrity checks (e.g., checksums), which may be required for production use.

The following sections describe the various systems and justify the parameter values used in our model.

3.2.1. SNL Red Storm Parameters The Red Storm at SNL is a Cray XT3 system comprised of 12,960 dual-core compute nodes and 320 dual-core I/O nodes. The I/O nodes are split evenly between two 100 TB filesystems, one for classified work and one for unclassified work, each of which is supposed to provide 50GB/s sustained throughput to the storage devices. However, as indicated by recent experiments, the I/O nodes deliver filesystem performance that is well below the targeted throughput [38]. Using the LLNL IOR benchmark writing to a file per process, our independent experiments (shown in Figure 1) compare the actual performance of checkpointing on Red Storm, in terms of throughput in GB/s, with the modeled performance. The modeled performance, which reflects hardware rates, further emphasizes that Red Storm's achieved filesystem performance is well below hardware rates.

The Red Storm network consists of a 3-D mesh with a per-link (one-way) peak bandwidth of 4.8GB/s and a peak minimum bisection bandwidth of 2.3TB/s. See [5] for a detailed description of the architecture of the SNL Red Storm, as well as a presentation of the employed design philosophy.

3.2.2. LLNL BlueGene/L Parameters The IBM BlueGene/L (BG/L) at LLNL has 64K dual-core compute nodes and 1K I/O nodes. Each I/O node is a Lustre [4] client that connects to an 896TB Lustre cluster. The Lustre cluster consists of 224 "Object Storage Servers" (OSSs), each attached to a Data Direct Network 8500 RAID controller through a 2Gb/s link. LLNL's BG/L uses a separate network for storage and computation. The storage network consists of 1,024 1Gb/s interfaces and can provide a potential I/O bandwidth of 128GB/s to the storage system. Note, however, that the Lustre system was designed to provide 45GB/s theoretical bandwidth between BG/L and the storage cluster. The hope is to obtain 80% of the theoretical bandwidth but, to date, only about 22GB/s has been achieved by LLNL's IOR benchmark [28].

3.2.3. ORNL Jaguar Parameters The Jaguar at ORNL is a Cray XT system with 11,590 nodes, each comprised of a 2.6 GHz dual-core AMD Opteron and 4GB memory. Of these nodes, 11,508 are used exclusively for computation, 10 are service and login nodes, and the remaining 72 are used for I/O to serve three Lustre filesystems–one 300TB system and two 150TB systems. Each I/O node has four Lustre object-storage targets (OSTs). Two OSTs serve the 300TB system, while the other two serve the 150TB systems. The expected peak block-I/O bandwidth to the 300TB system is 45GB/s.

Like the Red Storm at SNL, each node is connected to a Cray SeaStar router through HyperTransport technology. The SeaStar routers are interconnected in a 3D-torus. Each router provides six network links (one for each neighbor in the torus), each with a peak (one-way) link bandwidth of 3.8GB/s. The peak bisection bandwidth is 1.4TB/s. See [37] for a performance evaluation of the XT3 at ORNL. Details about the recent upgrade of the ORNL XT system can be found at http://info.nccs.gov/resources/jaguar.

3.2.4. Petaflop System Although no true petaflop capability-class systems exist, Tomkins presents a "conservative" description of the system requirements for this next class of systems in [34]. The architecture of the Red Storm follow-on remains basically the same as its predecessor with improvements in the network, storage system, processors, and memory capacity. A petaflop Red Storm system will consist of over 50K compute nodes. Our table shows two processors per node, but we do not know exactly how many processors per compute node the system will have. If the trend toward multi-core systems continues, it would not be surprising to see up to 64 cores/node. Applications will have to execute on 25K or more compute nodes with over 50% efficiency and have an I/O throughput to the filesystem of 500GB/s. Each compute node will

Table 1. Parameter values for the studied MPPs .

Parameter	Red Storm	BlueGene/L	Jaguar	Petaflop
$n_{max} \times cores$	$12,960 \times 2$	$65,536 \times 2$	$11,590 \times 2$	$50,000 \times 2$
d_{max}	1GB	0.25GB	2.0GB	2.5GB
M_{dev}	5 years	5 years	5 years	5 years
β_s	50GB/s	45GB/s	45GB/s	500GB/s
β_n	2.3TB/s	360GB/s	1.8TB/s	30TB/s
β_L	4.8GB/s	1.4GB/s	3.8GB/s	40GB/s

need at least 5GB of memory and the network will need a per-link (one-way) bandwidth of 40GB/s with a bisection bandwidth of 30TB/s.

3.2.5. MTTI for Large-scale Applications The specifications of both Red Storm and the proposed petaflop system expect a MTTI of over 50 hours for the whole system, including software and hardware failures. Given our assumption of an exponential failure distribution, achieving 50 continuous hours of service without any failures on a 128K-processor application means that the per-processor MTTI has to be $128K \times 50$ hours, or 748 years! However, a recent paper from Schroeder, et al. [30], which documents a variety of different types of interrupts registered for MPP systems at Los Alamos National Laboratory (LANL), shows that with software interrupts caused by either the operating system or application libraries, even the most reliable systems achieve an MTTI of no more than five years/processor. Based on failure rates reported for the ASCI-Q supercomputer at LANL, Elnozahy, et al. [13] suggest an even lower value of single-node MTTI of one year, as a "conservative" estimate. Since we do not have definitive failure rates for the systems studied, we use a generous five-year processor failure rate (MTTI); this is optimistic, considering the failure rates reported in the literature.

3.3. Model Results

For the systems studied, using Daly's model and our model for checkpoint overhead, this section presents results regarding the optimal checkpoint interval, the throughput of the checkpoint operation, and the checkpoint overhead as a percentage of application execution time. Figure 2 depicts the optimal checkpoint interval as a function of the number of compute-node processors. Since the probability of application failure is directly proportional to the number of employed processors, as the figure shows, the application increases the frequency of checkpoint operations, i.e., the optimal checkpoint interval decreases, as the number of employed processors increases to account for the increased probability of failure.

Figure 3 illustrates the modeled throughput of a checkpoint operation of our representative parallel application

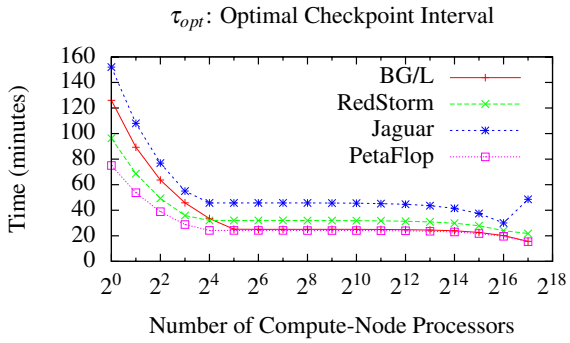

Figure 2. Optimal checkpoint interval as a function of the number of compute-node processors.

executed on the MPPs studied. As shown in the figure, for any job executing on more than 32 compute-node processors, the execution time of a checkpoint operation is governed by storage system performance. In contrast, for smaller jobs, the storage system can keep up with the demands of checkpointing because the aggregate node-link bandwidth of the compute-node processors does not exceed the storage system's ability to consume data. On all systems, the execution time of a checkpoint operation is never bound by the bisection bandwidth.

Figure 4 shows the aggregate execution time spent performing all checkpoint operations as a percentage of the overall application execution time. The following formula is used to calculate this percentage:

$$\frac{T_\delta}{T_s + T_\delta},$$

where T_s is the amount of compute time required for the application, i.e., the solve time without checkpointing, and T_δ is the total execution time spent performing checkpoint operations. It is this fraction that matters to the computational scientist; it provides an upper bound on the scalability of the application. According to our model, even when a checkpoint operation executes at hardware rates, an application that executes on 64K processors can achieve no better than 70% efficiency, even on the hypothetical petaflop

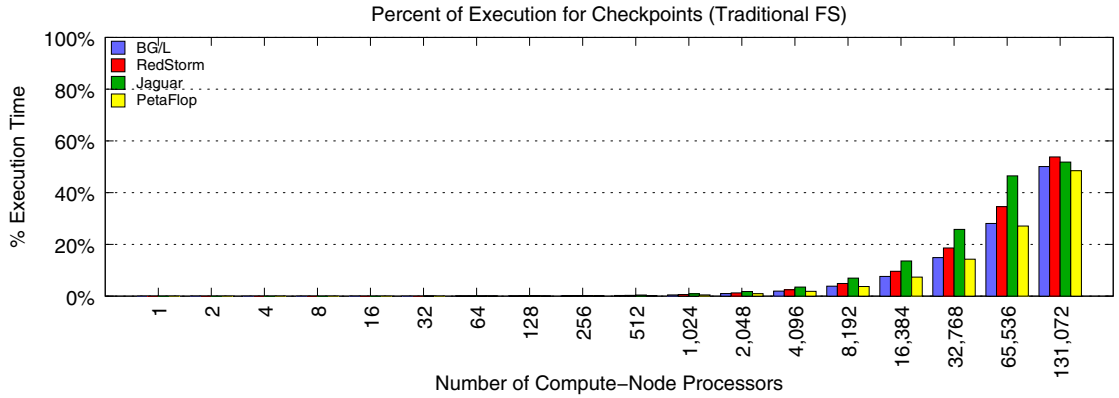

Figure 4. Aggregate checkpoint overhead as a percentage of application execution time.

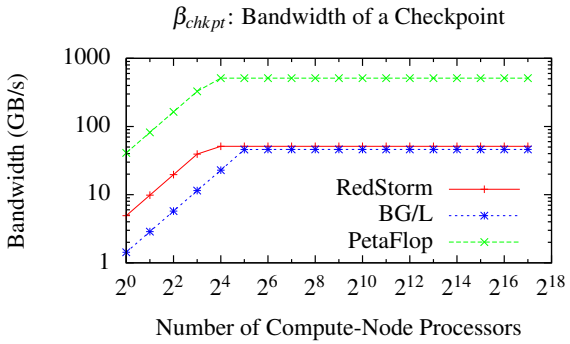

Figure 3. Modeled throughput of a checkpoint operation as a function of the number of compute-node processors.

system. The dramatic increase in checkpoint overhead as the system size increases demonstrates the need to investigate alternative approaches to checkpoint/restart.

Our analysis also provides substantial evidence to explain why checkpoint-to-disk has been an acceptable solution to date. On capacity systems, with small job sizes, system-directed checkpoints, where the system checkpoints the entire memory footprint, are viable. For applications that scale to more than 4K processors, application-directed checkpoint-to-disk solutions are sufficient if the application can select the data that needs to be dumped, opting to re-calculate portions of lost data. However, as applications scale beyond 32K processors, application-directed checkpoint operations begin to dominate execution time, severely limiting application scalability.

4. Reducing Checkpoint Overhead

According to Daly's model, the optimal checkpoint interval is dependent on two variables: the MTTI and the overhead of a checkpoint operation. We focus our efforts on ways to reduce the checkpoint overhead as a means to improve overall application performance. There are a number of ways to achieve this goal, some of which are mentioned in Sections 1 and 2. In this paper, we focus on exploring the use of lightweight filesystems [20] and overlay networks [15] to reduce checkpoint overhead, and we evaluate the impact of this approach, which we call LWFS+overlay in this paper, using Daly's model.

4.1. Lightweight Filesystems

Lightweight filesystems [20] allow secure, direct access to storage, bypassing features of traditional filesystems that impose performance bottlenecks. Figure 5 illustrates the core architecture of a lightweight filesystem (LWFS). The LWFS core architecture consists of a small set of services and mechanisms to provide security, efficient data transport, and direct access to storage. It does not provide direct support for traditional filesystem services like naming, consistency/conflict management, or organizational information that describes data distribution. If the application requires these services, the user includes the necessary library services at link time.

Figure 6 illustrates why the LWFS architecture, a lightweight storage architecture, is well suited for application checkpoints. These results were obtained on SNL's Darkstar cluster, an I/O-development system that consists of 64 dual-core Opteron compute nodes and 16 dual-core Opteron I/O nodes configured to mimic the I/O-node configuration of Red Storm. First, referring to the figure, consider the negative effect that the consistency semantics of traditional filesystems have on the performance associated with shared-file access (labeled n-to-1 in the figure). Also, consider the alternative, i.e., the file-per-process approach (labeled n-to-n), which generates an unnecessarily large number of operations targeted at a centralized metadata server. In contrast, with LWFS, it is possible to design a library that permits each client process to allocate on a

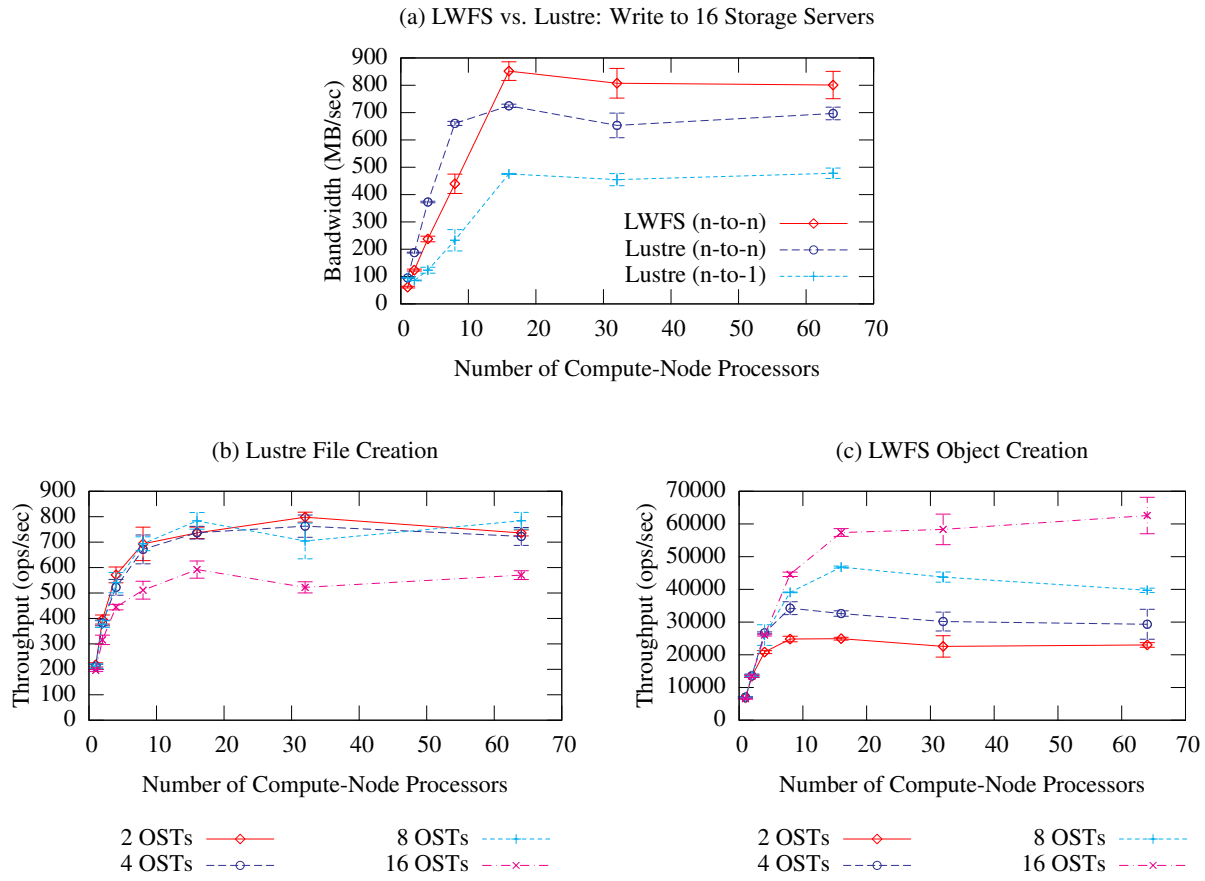

Figure 6. The first figure (a) shows n-to-1 (shared-file) and n-to-n (file-per-process) write performance of Lustre compared to the n-to-n write performance of LWFS. The second and third figures (b and c) show throughput (ops/sec) of creating files using Lustre and creating objects (in parallel) using LWFS. All experiments were performed on SNL's Darkstar cluster.

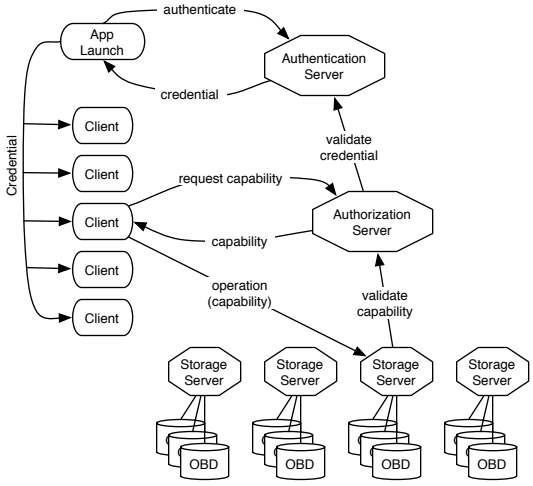

Figure 5. The LWFS core architecture.

storage server its own object for checkpoint data. After all clients dump their states, the application collectively generates the necessary metadata to represent the distributed data set; then it selects one client to associate that metadata with a name in an external naming service. This approach avoids the expensive overhead of a file-per-process case, while achieving near physical bandwidths to the storage system. With respect to the checkpoint model, this only effects the start-up cost, α_c. Based on the measured results from Figure 6(c), for LWFS, we set $\alpha_c = n/60,000 seconds$; this is a conservative estimate of the cost of allocating objects on a large system.

4.2. Overlay Networks

LWFS architectures provide a direct-to-storage option for checkpoints that can improve I/O performance when compared to using a traditional "heavyweight" filesystem. However, these improvements will be modest if the data

throughput is bound by the bandwidth of the storage system.

One way to relieve this storage system bottleneck is to exploit available processing and memory resources in an overlay network [15, 29]. Overlay networks provide a mechanism that allows an application to move potentially performance-limiting I/O operations off the compute nodes; this allows compute nodes to perform I/O operations unencumbered by the associated overheads and potential serialization imposed by the I/O operations. There are a number of interesting uses for overlay networks. For example, overlay networks could (1) filter input data in an application-specific way, saving network bandwidth and compute-node memory [16, 3, 14, 17]; (2) efficiently route data to compute nodes using data-dependent mapping functions, for example in applications with data-dependent decomposition of unstructured data [16]; and (3) process in-flight data sets to transform data into a format that matches the needs of the computation or a particular data distribution, for example to convert time-series data into frequency data for seismic imaging [21].

For the purpose of improving the I/O performance of checkpoint operations, a simple use for overlay networks is to buffer checkpoint data for applications that have compute-node memory constraints. This approach allows the application to checkpoint some, if not all, of its state at network bandwidths rather than storage bandwidths. On all of the systems studied in this paper, bisection bandwidth is at least an order of magnitude greater than the peak storage system bandwidth. An important benefit of this approach is that it does not require additional memory resources on the compute nodes, a limitation that makes checkpoint-to-memory, asynchronous checkpoint, and incremental checkpoint approaches impractical for many large-scale applications [24]. Note that for applications that are not memory constrained, using overlay nodes purely as buffers is not practical. These applications will benefit more from using an asynchronous checkpoint approach that exploits buffers in the compute nodes to overlap I/O and computation.

To model the use of overlay networks for checkpoint operations, we compute the checkpoint overhead as follows:

$$\delta = \alpha_c + \begin{cases} \frac{dn}{\beta_N} & dn \le k \\ \frac{k}{\beta_N} + \frac{(dn-k)}{\beta_s} & dn > k \end{cases}, \qquad (3)$$

where k is the effective memory that can be used at the network bandwidth, i.e., the amount of checkpoint data transferred over the network before the transfer becomes bound by the storage system bandwidth, and $\beta_N = \min(n\beta_L, \beta_n)$ is the minimum of the aggregate link bandwidth and the bisection bandwidth of the network. The variable k is the sum of μ, the combined memory in the overlay network, and the amount of data transferred to storage while μ is being transferred to the overlay network. Thus,

$$\begin{aligned} k &= \mu + \mu\left(\frac{\beta_s}{\beta_N}\right) + \mu\left(\frac{\beta_s}{\beta_N}\right)^2 + \cdots \\ &= \mu \sum_{i=0}^{\infty} \left(\frac{\beta_s}{\beta_N}\right) \\ &= \mu\left(\frac{1}{1-\frac{\beta_s}{\beta_N}}\right). \end{aligned}$$

As shown in Equation 3, when $(dn > k)$, the terms $\frac{k}{\beta_N}$ and $\frac{(dn-k)}{\beta_s}$ represent the time spent bound by the network bandwidth (β_N) and the time spent bound by the storage system bandwidth (β_s), respectively.

The memory in overlay nodes can be used to provide buffers for bursts of I/O. The size of the checkpoint data and the size of this memory determine how much the system can hide the checkpoint overhead. However, this introduces a lower bound on the checkpoint interval that can be employed in the system, τ_{lb}, given by

$$\tau_{lb} = \frac{\mu}{\beta_s} \min\left(1, \frac{nd}{k}\right). \qquad (4)$$

Recall that k is size of the effective memory that can be written at the network bandwidth rate and $k > \mu$, where μ is the size of the memory in the overlay network.

For a system, as described in this section, that uses the memory in an overlay network as a buffer for checkpoint data, we define the optimal checkpoint interval, τ_{opt} as

$$\tau_{opt} = \max\left(\tau_{opt-D}, \tau_{lb}\right),$$

where τ_{opt-D} is Daly's optimal checkpoint interval defined by Equation 2 and τ_{lb} is defined by Equation 4.

Note that systems like Red Storm and BlueGene/L already pass data through intermediate "I/O nodes", which run Linux. On BG/L, there are 1,024 I/O nodes, one for every 64 compute nodes, that act as filesystem clients, which simply forward calls to the back-end filesystem. On Red Storm there are 320 nodes, each attached to storage devices. One goal of our future work is to investigate how to use these nodes for more application-specific purposes (for example, buffers), rather than just interfaces to the I/O system. We also want to explore using application-dedicated nodes for this purpose, and eventually investigate opportunities to use these nodes to manage state in a way that allows recovery from individual node failure without restarting the entire application.

We reiterate a point made in the introduction: We do not explore how failures in the overlay network or the underlying storage system affect our model. However, we believe that it is possible to design libraries that use overlay net-

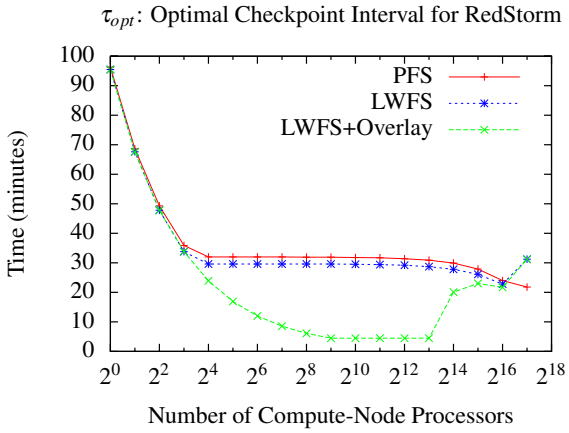

Figure 7. Optimal checkpoint interval for PFS, LWFS, and LWFS+overlay on Red Storm.

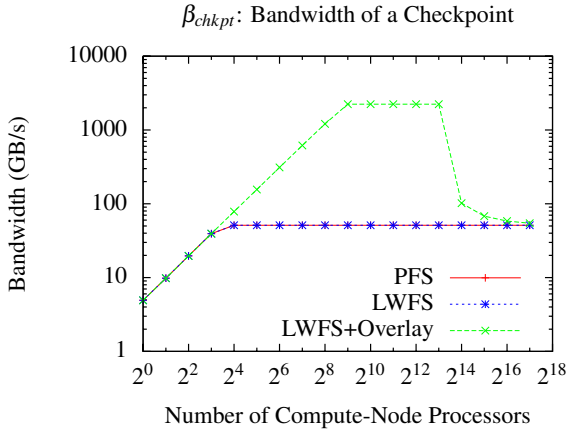

Figure 8. Throughput of LWFS+overlay checkpoint operation on Red Storm.

work memory to buffer checkpoint data and transfer it to the storage system in such a way that it has minimal impact on the probability of application failure (i.e., employing the memory in overlay networks will not influence the calculation for M_{app}). This is another topic for future work.

4.3. Results

For the Red Storm system at SNL, we use the models derived in Sections 3, 4.1, and 4.2 to estimate the potential benefit of LWFS+overlay on checkpoint performance. The results for Red Storm are presented in Figures 7 and 8, as well as in 9 and 10, which present results for the other three systems. With respect to Figures 7 and 8, the results for the other three systems are similar. Due to space constraints, we omit full results for all systems. In all of our models, we assume that there are 1,024 intermediate-node processors in the overlay network (less than 1% of the total compute-node processors), each with double the memory of a normal compute-node processor.

Figure 7 shows Daly's optimal checkpoint interval for our representative parallel application executed on Red Storm with the normal parallel file system (PFS), LWFS, and LWFS with an overlay network. An interesting side-effect of Daly's equation is that a reduction in checkpoint overhead also reduces the checkpoint interval. Initially a decrease in application execution time due to a reduction in checkpoint frequency seemed counter-intuitive. However, when you consider rework time (the time required to recompute data lost due to failures), which is included in Daly's model, it makes sense that as checkpoint operations become cheaper, the application performs checkpoints more often in order to reduce the amount of work lost due to failures. A detailed exploration of this phenomenon is presented in [2].

The effective throughput of an LWFS+overlay checkpoint operation on Red Storm, illustrated in Figure 8, behaves as we expected. The additional memory provided by the intermediate nodes in the overlay network causes the throughput to be bounded by the aggregate link bandwidth, then by the bisection bandwidth, and finally, when the size of a checkpoint exhausts the memory on the intermediate nodes, by the storage bandwidth. As shown in Figure 9, for all the MPP systems studied, this has a dramatic effect on the percentage of execution time spent checkpointing. Figure 10 is another bar graph that shows, for all the MPP systems studied, the relative difference between the standard filesystem approach and the LWFS+overlay approach. We calculate the relative difference as

$$\frac{P_{fs} - P_{overlay}}{P_{fs}},$$

where P_{fs} is the percentage of execution time for the standard filesystem approach, and $P_{overlay}$ is the percentage of execution time for the LWFS+overlay approach. The relative difference plot gives insight into how much better one approach is than the other. Below 8K processors, LWFS+overlay reduces the checkpoint overhead to less than 1% of the total application execution time. At 16K processors, checkpoint data exhausts the memory in the overlay network and the operations again become bound by the storage system bandwidth.

5. Future Work

This paper presents results of a preliminary study to understand the impact of application-directed checkpointing on the next generation of massive-scale systems. This section describes our plans to extend this work. In particu-

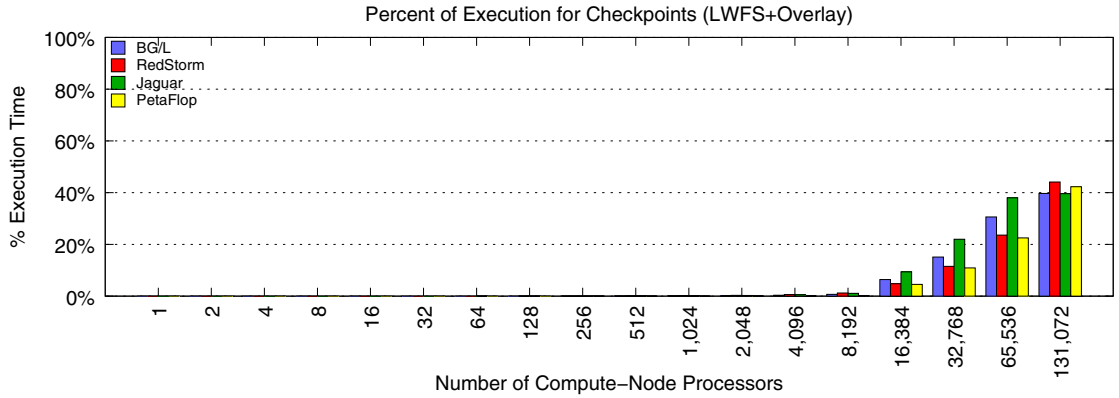

Figure 9. LWFS+overlay checkpoint overhead as a percentage of total application execution time for the MPP systems studied.

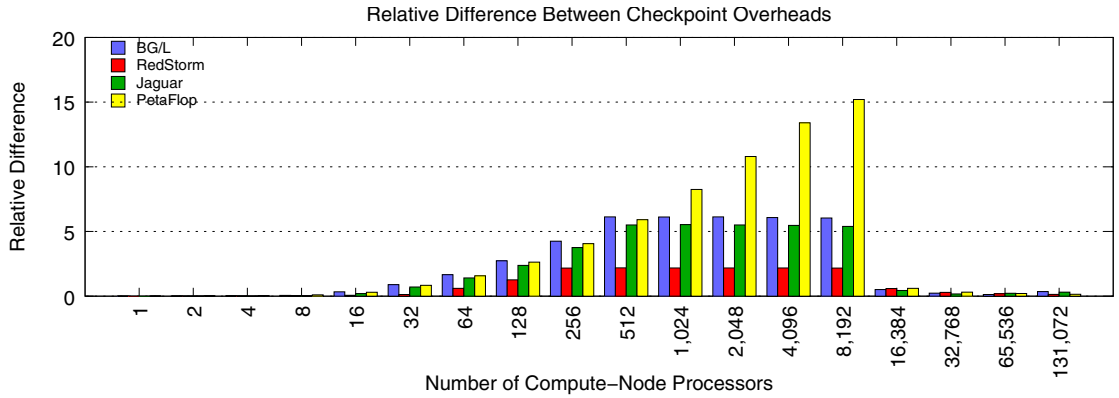

Figure 10. Relative difference between checkpoint overheads for standard parallel filesystems and LWFS+overlay for the MPP systems studied.

lar, we plan to validate our models on a large-scale system, improve the accuracy of our models by including realistic overheads associated with the network and storage system, and investigate other fault-tolerance methods.

Although Section 4 includes validation of standard parallel filesystem and LWFS performance on a modest-sized cluster, we have not yet validated the performance models on a large-scale system. In the next few months, we will actively develop experiments to validate our analytical models on the Red Storm system at SNL.

Our models take an overly optimistic view of application performance to establish a lower bound on the performance impact of application-directed checkpoints. On most systems, however, measured performance is often much worse than hardware rates. A logical next step is to refine our models to include realistic, expected overheads (for production systems) for operations associated with the storage system, network, and other system components.

The LWFS and overlay network approach, LWFS+overlay, presented in Section 4, is just one of a number of interesting approaches to reduce checkpoint overhead. Our results show that when the overlay network contains sufficient memory, the approach is quite effective at reducing checkpoint overhead; however, when the size of the checkpoint data is larger than the memory in the overlay network, LWFS+overlay is no better than a traditional filesystem approach. Our future work includes investigating hybrid approaches that combine the use of overlay networks and client-side buffering. Such an approach would be effective, for example, if compute nodes have a small portion of available memory that, when combined with the memory in the overlay network, is sufficient to buffer all checkpoint data. Our modeling approach gives us a tool that can be used to evaluate and theorize on the value and applicability of some of these approaches on the next generation of systems.

Although our analytical models provide some evidence that we can improve I/O performance for checkpoint operations, it is clear that I/O improvements alone are not enough. Another goal of our research is to investigate al-

gorithms that use the intermediate nodes in an overlay network, along with lightweight storage architectures, for continuous computing, even when application nodes fail. For example, one interesting idea is to use nodes in an overlay network in the same way diskless checkpointing approaches use compute nodes. While the overlay nodes provide the memory lacking on the compute nodes, the parity computation can be "offloaded" to the overlay network, resolving some of the issues discussed in [27].

6. Summary

This paper uses mathematical models to approximate the performance impact of application-directed checkpointing on a representative scientific application running on three existing MPP systems and one theoretical petaflop system. To establish a lower bound on the performance impact of checkpointing, our models assume perfect scalability of the filesystem, no overheads or contention in the network, and periodic checkpoints at the interval defined by Daly's function for the optimal checkpoint interval.

Our experiments investigate three different application-directed, checkpoint approaches: a typical approach that dumps direct to a parallel file system, an approach that dumps to a lightweight file system, and an approach that exploits available memory in an overlay network as a buffer between the application and the storage system.

Our analysis of the traditional approach illustrates two important points. First, for applications that use fewer than 16K processors, checkpoint overhead accounts for less than 10% of the overall application execution time. Since today's massive-scale systems are just now reaching this scale, the impact of the choice of fault-tolerance approach on application execution time has not yet been realized. This explains the general lack of interest we have experienced when discussing checkpoint optimizations with computational scientists at SNL.

The results also show that the compounding effect of larger checkpoint data files and increased checkpoint frequency to account for increased probability of failure, contribute to significant overheads when applications employ more than 16K processors. On some systems this overhead accounts for more than 50% of the total execution time when the application scales beyond 64K processors.

Our analysis of application performance when using overlay networks to buffer checkpoints is encouraging. The results show that when the overlay network contains sufficient memory, the approach is quite effective at reducing the checkpoint overhead. However, as the size of a checkpoint file increases beyond the size of the overlay network memory, the performance attained by using overlay networks approaches that of traditional filesystems. In our experiments, using an overlay network with 1,024 processors (1% of the total number of system processors), each with a memory capacity of twice the normal compute-node processor, when the application uses 16K processors or fewer, all of the data in a checkpoint file can be transferred at network rates rather than storage system rates .

In short, this work provides insight and understanding that will help motivate and guide future efforts related to fault-tolerance research. It is clear from our results that new approaches are needed. Our mathematical models provide a useful tool kit with which to evaluate the viability of experimental fault-tolerance methods for applications executed on massive-scale systems.

References

[1] S. Agarwal, R. Garg, M. S. Gupta, and J. E. Moreira. Adaptive incremental checkpointing for massively parallel systems. In *Proceedings of the 18th Annual International Conference on Supercomputing*, pages 277–286, New York, NY, 2004. ACM Press.

[2] S. Arunagiri, S. Seelam, R. A. Oldfield, M. R. Varela, P. J. Teller, and R. Riesen. An analysis of the consequences of a reduction in checkpoint latency for periodic checkpointing systems. Technical Report SAND2007-xxxx, Sandia National Laboratories, 2007.

[3] A. J. Borr and F. Putzolu. High performance SQL through low-level system integration. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 342–349, Chicago, IL, 1988. ACM Press.

[4] P. J. Braam. The lustre storage architecture. Cluster File Systems, Inc. Architecture, Design, and Manual for Lustre, November 2002. http://www.lustre.org/docs/lustre.pdf.

[5] R. Brightwell, W. Camp, B. Cole, E. DeBenedictis, R. Leland, J. Tomkins, and A. B. Maccabe. Architectural specification for massively parallel computers: an experience and measurement-based approach. *Concurrency and Computation: Practice and Experience*, 17(10):1271–1316, March 2005.

[6] W. J. Camp and J. L. Tomkins. The red storm computer architecture and its implementation. In *The Conference on High-Speed Computing: LANL/LLNL/SNL*, Salishan Lodge, Glenedon Beach, Oregon, April 2003.

[7] T.-C. Chiueh and P. Deng. Evaluation of checkpoint mechanisms for massively parallel machines. In *Proceedings of the Annual Symposium on Fault Tolerant Computing*, pages 370–379, Sendai, Japan, June 1996. IEEE Computer Society Press.

[8] J. Daly. A model for predicting the optimum checkpoint interval for restart dumps. *Lecture Notes in Computer Science*, 2660:3–12, August 2003.

[9] J. Daly. A strategy for running large scale applications based on a model that optimizes the checkpoint interval for restart dumps. In *Proceedings of the 26th International Conference on Software Engineering*, pages 70–74, Edinburgh, Scotland, UK, May 2004.

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

0-7695-3025-7/07 $25.00 © 2007 IEEE

[11] E. N. Elnozahy, L. Alvisi, Y. M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3):375–408, September 2002.

[12] E. N. Elnozahy, D. B. Johnson, and W. Zwaenepoel. The performance of consistent checkpointing. In *Proceedings of the 11th Symposium on Reliable Distributed Systems*, pages 39–47, Houston, TX, October 1992. IEEE Computer Society Press.

[13] E. N. Elnozahy and J. S. Plank. Checkpointing for peta-scale systems: A look into the future of practical rollback-recovery. *IEEE Transactions on Dependable and Secure Computing*, 1(2):97–108, April–June 2004.

[14] E. Franke and M. Magee. Reducing data distribution bottlenecks by employing data visualization filters. In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing*, pages 255–262, Redondo Beach, CA, August 1999. IEEE Computer Society Press.

[15] A. Gavrilovska, K. Schwan, O. Nordstrom, and H. Seifu. Network processors as building blocks in overlay networks. In *Proceedings of the 11 th Symposium on High Performance Interconnects (HOTI03)*, pages 83–88, August 2003.

[16] D. Kotz. Expanding the potential for disk-directed I/O. In *Proceedings of the 1995 IEEE Symposium on Parallel and Distributed Processing*, pages 490–495, San Antonio, TX, October 1995. IEEE Computer Society Press.

[17] T. Kurc, C. Chang, R. Ferreira, and A. Sussman. Querying very large multi-dimensional datasets in ADR. In *Proceedings of SC99: High Performance Networking and Computing*, Portland, OR, November 1999. ACM Press and IEEE Computer Society Press.

[18] K. Li, J. S. Naughton, and J. S. Plank. Low-latency, concurrent checkpointing for parallel programs. *IEEE Transactions on Parallel and Distributed Systems*, 5(8):874–879, August 1994.

[19] R. A. Oldfield. Investigating lightweight storage and overlay networks for fault tolerence. In *Proceedings of the High Availability and Performance Computing Workshop*, Santa Fe, NM, Oct. 2006.

[20] R. A. Oldfield, A. B. Maccabe, S. Arunagiri, T. Kordenbrock, R. Riesen, L. Ward, and P. Widener. Lightweight I/O for scientific applications. In *Proceedings of the IEEE International Conference on Cluster Computing*, Barcelona, Spain, September 2006.

[21] R. A. Oldfield, D. E. Womble, and C. C. Ober. Efficient parallel I/O in seismic imaging. *The International Journal of High Performance Computing Applications*, 12(3):333–344, Fall 1998.

[22] K. Pattabiraman, C. Vick, and A. Wood. Modeling coordinated checkpointing for large-scale supercomputers. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 812–821, Washington, DC, 2005. IEEE Computer Society.

[23] F. Petrini and K. Davis. Tutorial: Achieving Usability and Efficiency in Large-Scale Parallel Computing Systems, August 31, 2004. Euro-Par 2004, Pisa, Italy.

[24] I. R. Philp. Software failures and the road to a petaflop machine. In *1st Workshop on High Performance Computing Reliability Issues (HPCRI)*. Los Alamos National Laboratory, February 2005.

[25] J. S. Plank. Improving the performance of coordinated checkpointers on networks of workstations using RAID techniques. In *Proceedings of the Symposium on Reliable Distributed Systems*, pages 76–85, 1996.

[26] J. S. Plank, Y. Kim, and J. J. Dongarra. Fault-tolerant matrix operations for networks of workstations using diskless checkpointing. *Journal of Parallel and Distributed Computing*, 43(2):125–138, June 1997.

[27] J. S. Plank and K. Li. Faster checkpointing with n+1 parity. In *Proceedings of the 24th International Symposium on Fault-Tolerant Computing*, pages 288–297, Austin, Texas, June 1994.

[28] R. Ross, J. Moreira, K. Cupps, and W. Pfeiffer. Parallel I/O on the IBM Blue Gene/L system. BlueGene Consortium Quarterly Newsletter, 2006.

[29] P. C. Roth, D. C. Arnold, and B. P. Miller. MRNet: A software-based multicast/reduction network for scalable tools. In *Proceedings of SC2003: High Performance Networking and Computing*, Pheonix, AZ, Nov. 2003.

[30] B. Schroeder and G. A. Gibson. A large-scale study of failures in high-performance computing systems. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN2006)*, Philadelphia, PA, June 2006. School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

[31] L. M. Silva and J. G. Silva. An experimental study about diskless checkpointing. In *Proceedings of the 24th EUROMICRO Conference*, pages 395–402, Vasteras, Sweden, August 1998. IEEE Computer Society Press.

[32] R. Subramaniyan, R. S. Studham, and E. Grobelny. Optimization of checkpointing-related I/O for high-performance parallel and distributed computing. In *Proceedings of The International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 937–943, 2006.

[33] T. B. Team. An overview of the BlueGene/L supercomputer. In *Proceedings of SC2002: High Performance Networking and Computing*, Baltimore, MD, November 2002.

[34] J. Tomkins. A conservative path to petaflop computing: The Red Storm architecture scaled to a petaflop and beyond. 4th Annual Workshop on Linux Clusters for Supercomputing, October 2003.

[35] N. H. Vaidya. A case for two-level distributed recovery schemes. *SIGMETRICS Perform. Eval. Rev.*, 23(1):64–73, 1995.

[36] N. H. Vaidya. Impact of checkpoint latency on overhead ratio of a checkpointing scheme. *IEEE Transactions on Computers*, 46(8):942–947, 1997.

[37] J. S. Vetter, S. R. Alam, J. Thomas H. Dunigan, M. R. Fahey, P. C. Roth, and P. H. Worley. Early evaluation of the cray xt3. In *Proceedings of the International Parallel and Distributed Processing Symposium*. Oak Ridge National Laboratory, April 2006.

[38] L. Ward, J. Laros, R. Klundt, and B. Kellog. Red storm IO performance and scaling. Presentation given to Cray Inc., August 2006.

[39] J. W. Young. A first order approximation to the optimum checkpoint interval. *Communications of the ACM*, 17(9):530–531, 1974.