

# Investigating Lightweight Storage and Overlay Networks for Fault Tolerance

Ron A. Oldfield  
Sandia National Laboratories

*Abstract*—The next generation of capability-class massively parallel processing (MPP) systems is expected to have tens-to-hundreds of thousands of processors, with individual applications consuming large fractions of the system. In such an environment, it is critical to have fault-tolerance mechanisms that allow continuous computing with minimal performance impact on the application. Unfortunately, the current “in-practice” approaches to fault tolerance do neither. This paper analyzes the performance impact of exiting approaches on next-generation systems and describes a new project at Sandia National Laboratories to investigate the use of “lightweight” storage architectures and overlay networks for fault tolerance. The combined use of these technologies has the potential to significantly reduce the I/O impact of checkpoint operations on application performance and allow applications to recover from component failures without a complete restart and with minimal use of compute node resources.

## I. INTRODUCTION

Today’s high-end MPP machines have tens of thousands of nodes. For example, “Red Storm”, the Cray XT3 machine at Sandia National Laboratories [2] has over ten thousand nodes, and the IBM BlueGene/L [17] installed at Lawrence Livermore National Laboratory, has over sixty-four thousand compute nodes. Both machines are expected to be used for large scale applications. For example, 80% of the node-hours of Red Storm are allocated to applications that use a minimum of 40% of the nodes.

The massive scale of current and next-generation MPP machines and their supported applications presents significant challenges related to fault tolerance. The primary problem is that the current “in-practice” approaches do not match well with the expected demands or usage models for these systems. For example, the most commonly used method to defend against application failure is a “checkpoint to disk”. In this approach, the application (or system) periodically outputs enough data to restart the application after a failure. As applications increase in size, the checkpoint-to-disk approach becomes increasingly expensive for several reasons. First, scientific applications (Sandia applications in particular) often use a large fraction of the available memory on a compute node. Thus, the amount of data for a checkpoint increases linearly with the number of compute nodes. Second, although the checkpoint data increases with the number of nodes, the rate at which the data can be output remains fixed. A burst of I/O for a checkpoint can overwhelm the I/O system causing substantial delays. Finally, most applications cannot survive a node failure, so the probability of failure increases with the number of nodes, causing the application to increase the frequency of checkpoints. Even on today’s systems, the I/O generated from checkpoints consumes nearly 80% of the total I/O usage [10]. The combination of the expense of checkpoint-to-disk approach and the trends to develop systems of ever-increasing size, may force the community to seriously evaluate alternative approaches.

Checkpoint-to-memory [7], [12], [16] is one such alternative to checkpoint to disk. The goal is to reduce the checkpoint overhead by having compute nodes manage the state of other compute nodes in their local memory. Since network and memory bandwidth is typically much faster than storage, this approach significantly reduces the time to perform a checkpoint. One problem with in-memory approaches is the parity computation [13]. If an application processor computes the parity, the bandwidth advantage of network/memory over storage decreases significantly [11]. However, the biggest problem with in-memory approaches and large-scale scientific applications are the memory resources required by the compute nodes. Since many large-scale scientific applications are already resource constrained on the compute node, in-memory approaches are seen as impractical for large systems.

A number of other approaches exist in research [3], [5], [6], but have never been accepted by the scientific community. The best explanation for the lack of interest among developers of large-scale applications is that the scale of the systems have not been large enough to justify a change. As we show in Section V, checkpoints only become a problem for the I/O system when systems reach scales at and beyond the largest existing MPP systems. Because of this, application-directed checkpoint to disk still dominates as the most widely used approach in HPC. However, if the technology trends continue along the same path, we will soon have systems with hundreds of thousands of nodes. If reliability of hardware and systems software do not improve substantially, checkpoint-to-disk will soon become impractical for large applications.

In this paper, we approximate, through analytic models, the performance impact of the checkpoint-to-disk approach on current and next-generation systems. We then describe a recently funded project to investigate lightweight storage architectures [9] and overlay networks [8] for fault tolerance. Our project has three phases: develop an I/O efficient checkpoint library that uses lightweight storage to dump checkpoint data direct to storage devices, integrate overlay networks to improve I/O performance by buffering state from bursty checkpoint operations, and investigate algorithms that use overlay networks to manage and restore state on failed compute nodes.

## II. MODELING CHECKPOINTS

To get some perspective on the performance impact of the checkpoint-to-disk approach, we constructed a simple analytic model of the checkpoint operation and applied it to current and next-generation systems at Sandia. Our model uses the following parameters:

- $\tau_{opt}$  = Optimal checkpoint interval  
 $n$  = Number of compute nodes used for the application  
 $d$  = Amount of data written by each node  
 $\delta$  = The time to output a checkpoint/restart file  
 $M$  = Mean time to interrupt (MTTI) per node  
 $\beta_s$  = Aggregate storage system throughput  
 $\beta_n$  = Aggregate bi-section network bandwidth  
 $\beta_L$  = One-way network bandwidth per link  
 $\alpha_c$  = Start-up cost of a checkpoint operation  
 $\beta_{chkpt}$  = Perceived bandwidth of a checkpoint operation

We calculate the optimal checkpoint interval using Daly's equation for  $\tau_{opt}$  [4]

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

Daly's equation is not perfect. For example it assumes an exponential distribution for the time between failures, which may not match empirical evidence [15]. Despite this flaw, the Daly equation is becoming popular among computational scientists as a good approximation to the optimal checkpoint interval. For that reason alone, it is useful in this analysis.

We use an optimistic equation to calculate  $\delta$ , the time required to output the checkpoint/restart file. In earlier papers, authors assume a fixed amount of time for the checkpoint operation. We use a more scientific approach. For our representative architectures (see Section ??), a checkpoint operation is either bound by the aggregate network bandwidth, the bi-section network bandwidth, or the storage system. The following equation for the checkpoint bandwidth,  $\beta_{chkpt}$ , accounts for the three different possibilities,

$$\beta_{chkpt} = \min(n\beta_L, \beta_n, \beta_s).$$

The equation for the checkpoint overhead becomes

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

where  $\alpha_c$  is the start-up cost (e.g., creating files) associated with a checkpoint operation.

The start-up cost of a checkpoint operation depends heavily on the algorithm used for the checkpoint. In POSIX-compliant parallel file systems, consistency semantics and device conflicts contribute to poor performance when writing to a shared file. To compensate, many Sandia applications create a file-per-process for checkpoint/restart files. The second approach improves write performance, but creates a significant overhead associated with sending thousands of simultaneous create operations through a centralized metadata server. See Figure 2 for measured results of the create and write phase of a checkpoint operation.

The equation for  $\delta$  assumes that each processor writes the same amount of state to the checkpoint/restart file(s) and that the parallel file system is perfectly scalable. These are overly optimistic assumptions, but they still provide a reasonable lower bound on the performance impact of the checkpoint operation.

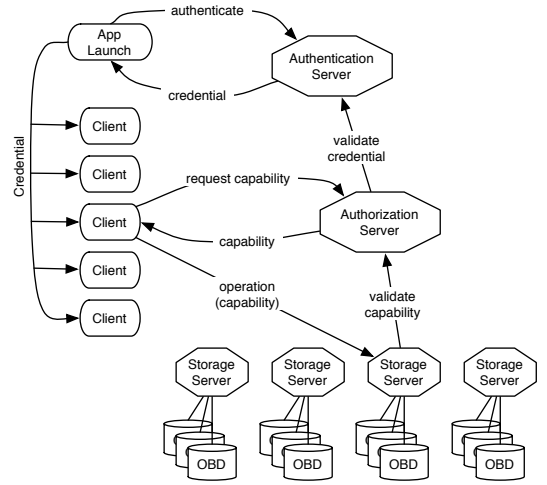


Fig. 1. The LWFS core architecture.

### III. IMPROVING CHECKPOINTS WITH LIGHTWEIGHT STORAGE

Lightweight storage architectures [9] allow secure, direct access to storage, bypassing features of traditional file systems that impose performance bottlenecks. Figure 1 illustrates the core architecture of a lightweight file system (LWFS). The LWFS core only includes mechanisms for security, efficient data transport, and direct access to storage. It does not provide direct support for traditional file system 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.

The LWFS architecture is well suited for application checkpoints. As Figure 2 illustrates, the imposed consistency semantics of traditional file systems hinders performance to shared files, but the alternative (file per process) generates an unnecessarily large number of operations to a centralized metadata server. With LWFS, it is possible to design a library such that each client process allocates its own object on a storage server for checkpoint data. After all clients dump their state, the application selects one process to construct the necessary metadata to represent the distributed dataset and associate it with a name in an external naming service. This approach avoids the expensive overhead of a file-per-process case, while still achieving near physical bandwidths to the storage system. With respect to the model for a checkpoint operation, this only effects the start-up cost. Based on the measured results from Figure 2-c, we set  $\alpha_c = n/60,000$  for LWFS, a conservative guess to the cost of allocating objects on a large system.

### IV. IMPROVING CHECKPOINTS WITH OVERLAY NETWORKS

The LWFS provides a direct-to-storage option for checkpoints that can modestly improve I/O performance for a checkpoint operation. Another way to improve I/O performance is to buffer data on intermediate nodes in an "overlay networks" [8] between the client and the storage system. In this way, the application can transfer much of the application state off the node at

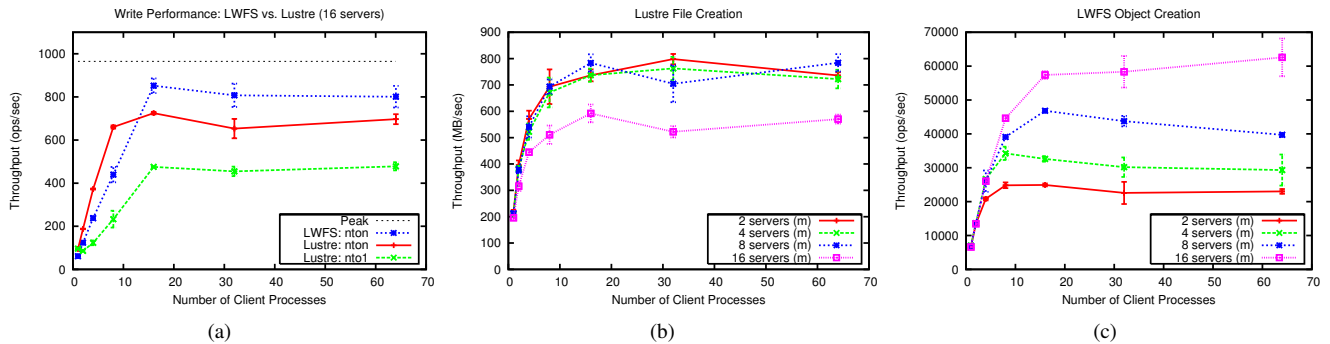


Fig. 2. The left figure (a) shows write performance of *nto1* (shared file) and *nton* (file per process). The right figures (b and c) show throughput (ops/sec) of creating files using Lustre and creating objects (in parallel) using LWFS. All experiments were performed on Sandia’s Darkstar cluster.

network bandwidths, rather than storage bandwidths. To model this impact, we modify the equation for checkpoint time to

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

where  $k$  is the amount of data transferred over the network before the transfer becomes bound by the storage system and  $\beta_N = \min(n\beta_L, \beta_n)$  is the minimum of the aggregate link bandwidth and the bi-section bandwidth of the network. The variable  $k$  is the sum of  $\mu$ , the combined memory in the overlay network, and the amount of data transferred to storage while  $\mu$  was being transferred to the overlay networks. Thus,

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

The left half of the equation for  $\delta$  represents the time spent bound by the network. The right half (for  $dn > k$ ) represents the time spent bound by the storage system.

Note that systems like BlueGene and RedStorm already pass data through “I/O nodes” that act as intermediate-nodes. On BG/L, there are 1024 nodes that act as file-system client that simply forward calls to the file system. On RedStorm there are 256 nodes (128 each side), each attached to storage devices. In both cases, these nodes run Linux. One goal of this project is to investigate how to use these nodes for more application-specific purposes (buffers for example), 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.

It is also important to note that overlay nodes do not impact the checkpoint interval. If an overlay node dies, it does not impact the application beyond reducing the number of buffers available to the application. In addition, this approach does

TABLE I  
PARAMETER VALUES FOR MPPS

Parameter	Red Storm	BlueGene/L	Petaflop
$n_{max}$	10,368 × 2	65,536 × 2	50,000
$d_{max}$	2 GB	0.5 GB	5 GB
$M$	5 yr	5 yr	5 yr
$\beta_s$	50 GB/s	45 GB/s	500 GB/s
$\beta_n$	1.15 TB/s	360 GB/s	30 TB/s
$\beta_L$	3 GB/s	1.4 GB/s	40 GB/s

not require additional memory resources on the compute node, a limitation that makes checkpoint-to-memory, asynchronous checkpoints, and incremental checkpoint approaches impractical for large systems [11].

## V. ANALYSIS

We estimated checkpoint overheads for the standard checkpoint-to-disk approach for several different MPP architectures. Our model represents an application that checkpoints half of its available memory at intervals that match the Daly’s optimal checkpoint interval. Each plot shows the performance of the same application running on Sandia’s RedStorm, LLNL’s BG/L, and a theoretical Petaflop machine, all scaled up to 128K nodes.

Table I shows values for the model parameters for the Red Storm system at Sandia, the BlueGene/L system at Lawrence Livermore, and a theoretical Petaflop system.

### A. LLNL BlueGene Parameters

A recent newsletter from the BlueGene Consortium details the projected performance of the I/O system on LLNL’s BlueGene system [14]. The BlueGene/L (BGL) at LLNL consists of 64K compute nodes and 1K I/O nodes. Each I/O node is a Luster client that connects to an 896 Terabyte Luster Cluster. The Luster cluster consists of 224 “Object Storage Servers” (OSS), each attached to Data Direct Network 8500 RAID controller through a 2Gb/s link. The BGL at LLNL uses a separate network for storage and computation. The storage network consists of 1024 1Gb/s interfaces and can provide a potential I/O bandwidth of 128 GB/s to the storage system. However, the Lustre system was designed to provide 45 GB/s theoretical bandwidth between

BGL and the storage cluster. They hope to obtain 80% of theoretical, but to-date, they have only been able to achieve around 22 GB/s using LLNL’s IOR benchmark [14].

### B. Sandia Red Storm Parameters

There are no published reports on the performance of Lustre on Sandia’s Red Storm System. We gathered the parameters for this analysis through [1], a paper that describes the architecture and design philosophy behind the Red Storm at Sandia. Sandia’s Red Storm consists of 10,368 compute nodes and 256 I/O nodes. The I/O nodes are split evenly between two files systems (one for classified and one for unclassified) that are each supposed to provide 50 GB/s throughput to the storage devices. The network consists of a 3-D mesh with a per-link (bi-directional) bandwidth of 6 GB/s and a minimum bi-section bandwidth of 2.3 TB/s. For this analysis, we are concerned about one-way traffic, so we cut the per-link and bi-section bandwidth in half.

### C. Petaflop System

Although no true Petaflop capability class systems exist, Tomkins presents a “conservative” description of the system requirements for this next class of system in [18]. 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 (applications will have to execute on 25K or more nodes with over 50% efficiency) and have an I/O throughput to the file system of 500 GB/s. Each compute node will need at least 5 GB of memory and the network will need a per-link bandwidth of 80 GB/s with a bi-section bandwidth of 61 TB/s.

### D. MTTI of large scale systems

The specifications of both the Red Storm and the proposed Petaflop system require a MTTI of over 50 hours, including software and hardware failures. While this is lofty goal, no system of any scale has been able to achieve such reliability. A recent paper from Schroeder et al.[15] documents a variety of different types of interrupts registered for MPP systems at Los Alamos National Laboratories. They found that even the most reliable systems achieved MTTI of no longer than 5 years/device when you include software interrupts caused by either the operating system or application libraries.

### E. Results

Figure 3 shows the optimal checkpoint interval as a function of the number of compute nodes. Since the probability of application failure is directly proportional to the number of compute nodes used, the application has to increase the frequency of checkpoints to account for the increased probability of failure. There is a kink in the checkpoint period plots when the checkpoint performance goes from being bound by the network to being bound by the storage system.

Figure 4 shows the throughput of the checkpoint operation. For any job larger than 32 nodes, a checkpoint operation is limited to the storage system performance. For small jobs, the storage system can keep up because the aggregate network links of

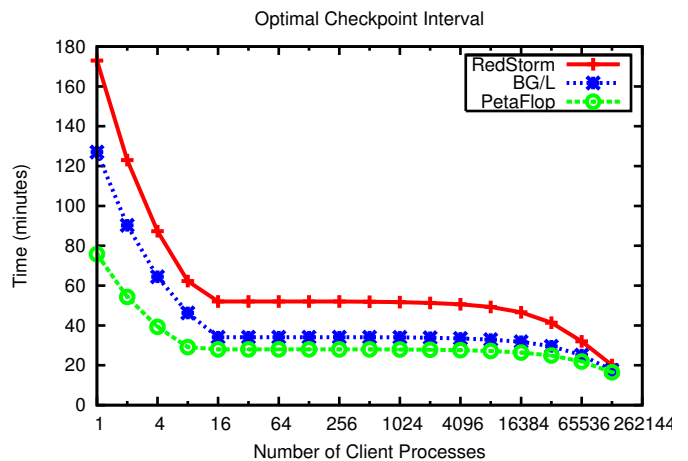


Fig. 3. Optimal checkpoint period as a function of the number of compute nodes.

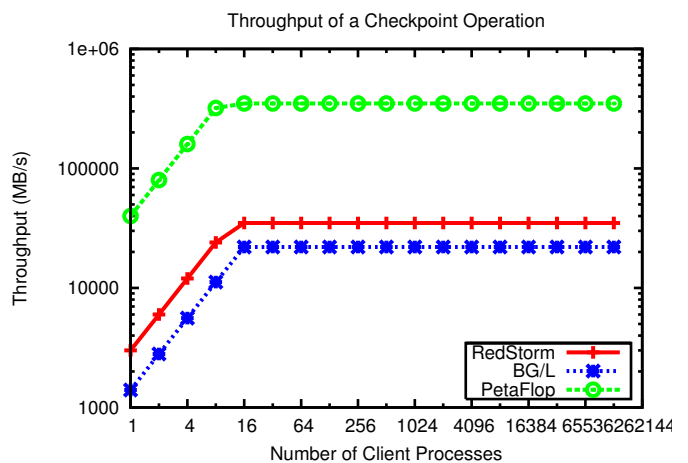


Fig. 4. Throughput of the checkpoint write operation.

the individual compute nodes does not exceed the storage systems ability to consume data. These experiments do not consider contention due to other applications.

Figure 5 shows the overhead of all checkpoints as a percentage of the overall execution time of the application. This is really what matters to the application scientist because it provides an upper bound on the scalability of the application. According to our model, a 64K node application can achieve no better than 70% efficiency even on the Petaflop system. Our model assumes extremely reliable hardware and software, a perfectly scalable file system, and perfectly scalable application (i.e., no communication besides checkpoint I/O). The dramatic increase in checkpoint overhead as the system size increases demonstrates the need to investigate alternative approaches.

Our analysis also provides substantial evidence to explain why checkpoint to disk has been an acceptable solution so far. 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 nodes, application-directed checkpoint to disk solutions are sufficient because the application can be choose the data that needs

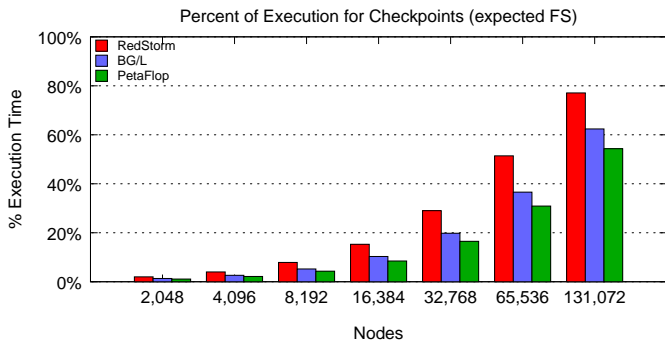


Fig. 5. Overhead of checkpoint as a percentage of execution time for a traditional PFS on RedStorm, BG/L, and PetaFlop systems.

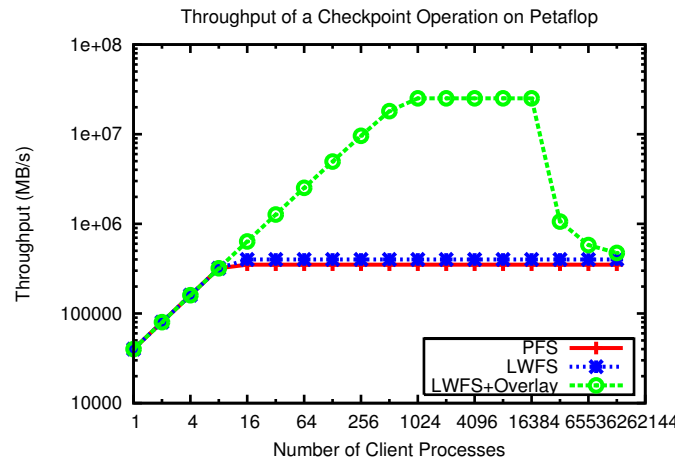


Fig. 7. Throughput of checkpoint operation.

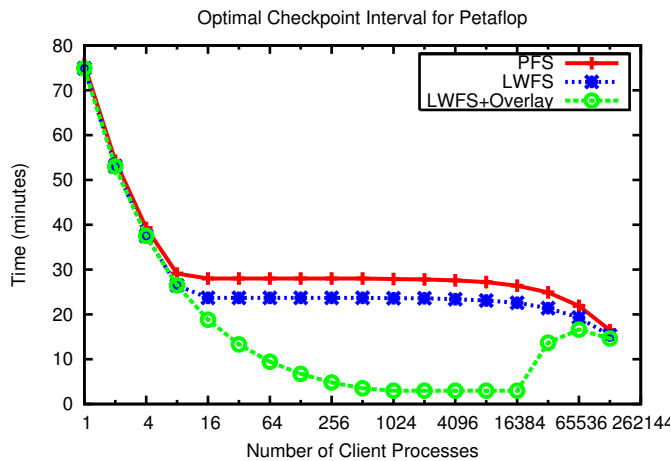


Fig. 6. Optimal checkpoint interval.

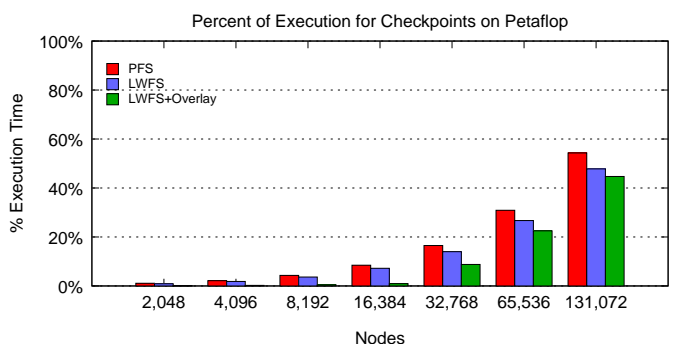


Fig. 8. Checkpoint overhead for PFS, LWFS, and overlay networks on the PetaFlop system.

to be dumped, opting to re-calculate portions of the lost memory. As applications grow beyond 32K in size, application-directed checkpoints begin to dominate the execution time, severely limiting the scalability of the application.

We approximate the potential benefit of LWFS and overlay networks by plotting results from the models derived in Sections II, III, and IV on the PetaFlop system. The results for RedStorm and BG/L are similar, so we decided it was unnecessary to present those results. In these plots, we assume there are 1024 intermediate nodes, each with 10 times the memory of a normal compute node.

Figure 6 shows Daly’s optimal checkpoint interval for the normal PFS, LWFS, and LWFS with an overlay network. An interesting side-effect of Daly’s equation is that a reduction in the time to perform a checkpoint also reduces the checkpoint interval. This is somewhat counter-intuitive and is not reflected in other studies that assume a fixed amount of time for a checkpoint. The effective throughput, illustrated in Figure 7, behaves as we expected. The additional memory provided by the intermediate nodes causes the throughput to be bounded by the aggregate network, then the bi-section bandwidth, and finally the storage bandwidth when the size of a checkpoint exhausts the memory on the intermediate nodes. This has a dramatic effect on the percent of execution time spent for checkpointing shown in Figure 8. The overhead of the overlay network is consistently

10% less than writing using a traditional PFS and 5% less than writing direct to storage through the LWFS.

## VI. EXTENSIONS FOR CONTINUOUS COMPUTING

Although our analytic 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 important goal for this project is to investigate and develop algorithms that use the intermediate nodes and lightweight storage for continuous computing, even when application nodes fail. This work requires research to develop mechanisms that allocate and inject a new compute node into an existing application (allowing the new node to assume the identity of the failed node). We also need to develop algorithms to efficiently bring all of the application nodes to a consistent state after replacing the failed node.

Partial application recovery not only involves the integration of LWFS with overlay networks, it also requires integration and cooperation with other systems services that do not yet exist. For example, if a compute node dies, we need to allocate a new compute node, install the previous state (either stored in memory or disk) onto a new node, possibly roll-back all other nodes or update the new node to the current state, then we need to have the new node resume the identity of the failed node. None of these services exist in the systems software for MPP systems.

We do not yet have clear ideas about how to accomplish this goal. This is, after all, a research project. As we continue to investigate the applicability of previous academic solutions to our environment, we hope to develop insight and expertise that will help solve this problem. We will also rely heavily on collaborative relationships with fault-tolerance and algorithms experts to achieve this goal.

## VII. SUMMARY

Our analysis shows that our proposed solution to use LWFS with overlay networks has potential to significantly improve performance, scalability, and reliability of mission-critical DOE scientific applications. Not only will this approach reduce the I/O-induced overhead of checkpointing for applications, but, if successful, it will also allow applications to survive independent node failures, eliminating the implicit assumption that probability of application failure is directly proportional to the number of nodes used by the application. Such an advance will have a tremendous impact on the HPC community because it significantly reduces the burden of fault tolerance on the application, network, and storage system.

## REFERENCES

- [1] Ron Brightwell, William Camp, Benjamin Cole, Erik DeBenedictis, Robert Leland, James Tomkins, and Arthur 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.
- [2] William J. Camp and James 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.
- [3] Tzi-Cker Chiueh and Peitao Deng. Evaluation of checkpoint mechanisms for massively parallel machines. In *Annual Symposium on Fault Tolerant Computing*, pages 370–379, Sendai, Japan, June 1996. IEEE Computer Society Press.
- [4] John Daly. A model for predicting the optimum checkpoint interval for restart dumps. *Lecture Notes in Computer Science*, 2660:3–12, August 2003.
- [5] Geert Deconinck, Johan Vounckx, Rudi Cuyvers, and Rudy Lauwereins. Survey of checkpointing and rollback techniques. Technical Report O3.1.8 and O3.1.12, ESPRIT Project 6731 (FTMPS), June 1993.
- [6] 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, SEP 2002.
- [7] Christian Engelmann and Al Geist. A diskless checkpointing algorithm for super-scale architectures applied to the fast fourier transform. In *In Proceedings of the International Workshop on Challenges of Large Applications in Distributed Environments*, pages 47–52, Seattle, WA, June 2003. IEEE Computer Society Press.
- [8] Ada Gavrilovska, Karsten Schwan, Ola Nordstrom, and Hailemeleket Seifu. Network processors as building blocks in overlay networks. In *Proceedings of the 11 th Symposium on High Performance Interconnects (HOTI'03)*, pages 83–88, August 2003.
- [9] Ron A. Oldfield, Arthur B. Maccabe, Sarala Arunagiri, Todd Kordenbrock, Rolf Riesen, Lee Ward, and Patrick Widener. Lightweight I/O for scientific applications. In *Proceedings of the IEEE International Conference on Cluster Computing*, Barcelona, Spain, September 2006. To appear.
- [10] Fabrizio Petrini and Kei Davis. Tutorial: Achieving Usability and Efficiency in Large-Scale Parallel Computing Systems, August 31, 2004. Euro-Par 2004, Pisa, Italy.
- [11] Ian 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.
- [12] 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 – 38, JUN 1997.
- [13] James S. Plank and Kai 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.
- [14] Rob Ross, Jose Moreira, Kim Cupps, and Wayne Pfeiffer. Parallel I/O on the IBM Blue Gene /L system. BlueGene Consortium Quarterly Newsletter, 2006.
- [15] Bianca Schroeder and Garth 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 15213.
- [16] L. M. Silva and J. G. Silva. An experimental study about diskless checkpointing. In *24th EUROMICRO Conference*, pages 395 – 402, Vasteras, Sweden, August 1998. IEEE Computer Society Press.
- [17] The BlueGene/L Team. An overview of the BlueGene/L supercomputer. In *Proceedings of SC2002: High Performance Networking and Computing*, Baltimore, MD, November 2002.
- [18] Jim 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.