2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

A User-level InfiniBand-based File System and Checkpoint Strategy for Burst Buffers

Kento Sato
Dep. of Mathematical and Computing Science
Tokyo Institute of Technology
2-12-1-W8-33, Ohokayama,
Meguro-ku, Tokyo 152-8552 Japan
Email: kent@matsulab.is.titech.ac.jp

Kathryn Mohror, Adam Moody,
Todd Gamblin and Bronis R. de Supinski
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94551 USA
Email: {kathryn, moody20, tgamblin, bronis}@llnl.gov

Naoya Maruyama
Advanced Institute for Computational Science
RIKEN
7-1-26, Minatojima-minami-machi,
Chuo-ku, Kobe, Hyogo, 650-0047 Japan
Email: nmaruyama@riken.jp

Satoshi Matsuoka
Global Scientific Information and Computing Center
Tokyo Institute of Technology
2-12-1-W8-33, Ohokayama,
Meguro-ku, Tokyo 152-8552 Japan
Email: matsu@is.titech.ac.jp

***Abstract*—Checkpoint/Restart is an indispensable fault tolerance technique commonly used by high-performance computing applications that run continuously for hours or days at a time. However, even with state-of-the-art checkpoint/restart techniques, high failure rates at large scale will limit application efficiency. To alleviate the problem, we consider using burst buffers. Burst buffers are dedicated storage resources positioned between the compute nodes and the parallel file system, and this new tier within the storage hierarchy fills the performance gap between node-local storage and parallel file systems. With burst buffers, an application can quickly store checkpoints with increased reliability. In this work, we explore how burst buffers can improve efficiency compared to using only node-local storage. To fully exploit the bandwidth of burst buffers, we develop a user-level InfiniBand-based file system (IBIO). We also develop performance models for coordinated and uncoordinated checkpoint/restart strategies, and we apply those models to investigate the best checkpoint strategy using burst buffers on future large-scale systems.**

***Keywords*-fault tolerance; checkpoint/restart; burst buffer;**

I. Introduction

The growing computational power of high performance computing (HPC) systems enables increasingly larger scientific simulations. However, as the number of system components increases, the overall failure rate of the system increases. Furthermore, the mean time between failures (MTBF) of future systems is projected to be on the order of tens of minutes or hours [1], [2], [3]. In fact, an earlier failure analysis on Hera, Atlas, and Coastal clusters at Lawrence Livermore National Laboratory (LLNL) [4] showed that a production application, the `pF3D` laser-plasma interaction code [5], experienced 191 failures in 5.6-million node-hours. Simple extrapolation assuming a constant node-hour failure rate shows that the estimated MTBF is 1.2 days for a 1,000-node cluster, 2.7 hours for a 10,000-node cluster, and 18 minutes for a 100,000-node cluster. Without fault tolerant techniques and more reliable hardware, applications will be unable to run continuously for even one day on such a large system. Therefore, as we look towards future large-scale systems, fault tolerance is becoming more important [6].

One indispensable technique for fault tolerance is checkpoint/restart (C/R), in which the application periodically writes a snapshot of its state to reliable storage, like a parallel file system (PFS). Then when a failure occurs, the application is restored to its previous state recorded in the snapshot. Although storing checkpoints in the PFS is highly reliable, this method imposes high overhead on application run time at large scales. Multilevel C/R improves on this approach by storing most checkpoints in fast, scalable storage on the compute nodes and only copying a select few to the more reliable PFS [7], [8]. This reduces the overhead of writing checkpoints in the common case, which greatly increases application efficiency. Further efficiency gains can be achieved by combining multilevel C/R with asynchronous I/O [4], [9] or uncoordinated checkpointing [10], [11], [12].

However, even with these state-of-the-art C/R techniques, high failure rates at large scale will significantly limit application efficiency. Our earlier failure analysis study showed that most failures affect a single compute node [4], [7]. To tolerate node failures, multilevel checkpointing libraries apply redundancy schemes to the cached checkpoints across node-local storage (e.g. local SSDs). For example, each checkpoint may be copied to a partner node, or the library may utilize a RAID algorithm and spread redundancy data across multiple compute nodes. This allows the application to recover from node failures assuming the number of nodes lost is less than what is tolerated by the redundancy scheme. However, with higher failure rates, the likelihood of multiple simultaneous node failures increases. If the simultaneous failures affect

978-1-4799-2784-5/14 $31.00 © 2014 IEEE
DOI 10.1109/CCGrid.2014.24

nodes in a shared redundancy set, the cached checkpoints will be lost and the application will need to restart from the PFS. This could mean the application would spend the majority of its time in C/R activities [4]. Thus, writing checkpoints to node-local storage is a scalable, but not reliable solution, and application efficiency may suffer at extreme scales.

Burst buffers have been proposed as node-local storage to alleviate the problems of writing to a shared PFS [13], [14]. Burst buffers are a new tier in the storage hierarchy to fill the performance gap between node-local storage and the PFS. They absorb the bursty I/O requests from applications and thus reduce the effective load on the PFS. System software can manage moving data between the burst buffers and the PFS asynchronously. The storage design is expected to be a promising architecture of future supercomputers.

In this paper, we consider using burst buffers to improve system resiliency. With burst buffers, an application can quickly store checkpoints with increased reliability. We explore how burst buffers can improve efficiency compared to using only node-local storage, and we develop and apply models to investigate the best checkpoint strategy with burst buffers for a wide range of C/R strategies. Our key contributions include:

- An InfiniBand-based file system (IBIO) that exploits the bandwidth of burst buffers;
- A model to evaluate system resiliency given C/R and storage configurations;
- Simulation results showing how system resiliency improves from the use of burst buffers;
- A quantitative examination of the trade-offs between coordinated and uncoordinated multilevel C/R;
- An analysis of which tiers in the storage hierarchy impact system reliability and efficiency; and
- An exploration of burst buffer configurations to discover the best for future large-scale systems.

In the next section, we categorize C/R strategies and describe the targets for our study. In Section III, we introduce our SSD burst buffer machine. In Section IV, we detail IBIO. We model multi-tiered storage and C/R strategies in Section V. In Section VI, we show IBIO performance, and in Section VII we simulate system efficiency given C/R strategies and storage configurations. In Section VIII we detail related work, and we conclude in Section IX.

II. C/R Strategies

Over the years, many C/R strategies have been studied. These techniques can be roughly categorized into single or multilevel, synchronous or asynchronous, and coordinated or uncoordinated C/R. We explain each C/R strategy and their advantages and disadvantages. We describe our target checkpointing strategies.

A. Single vs. Multilevel Checkpointing

The simplest approach for C/R is to write all checkpoints to a single location, such as the PFS [15], [16]. However, when a large number of compute nodes write their checkpoints to a PFS, contention for shared PFS resources leads to low I/O throughput. Multilevel checkpointing is an approach for alleviating this bottleneck [7]. Earlier failure analysis showed most failures on current supercomputers affect a single compute node, which does not necessarily require writing checkpoints to the reliable, but slow PFS [7]. For example, only 15% of failures on the Hera, Atlas and Coastal systems at LLNL required checkpoints on the PFS for restarts. The study also showed multilevel C/R can benefit application efficiency in the face of higher failure rates and increased relative overhead of checkpointing to the PFS that may occur on future systems. Therefore, in this study we only target multilevel C/R.

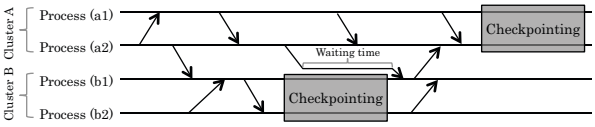

Fig. 1. Indirect global synchronization on uncoordinated C/R

B. Synchronous vs. Asynchronous Checkpointing

Checkpointing libraries can write checkpoints either synchronously or asynchronously. Synchronous checkpointing methods write checkpoints such that all processes write their own checkpoints concurrently, and are blocked until the checkpoint operation completes [15], [7]. In asynchronous checkpointing [16], [4], [8], the methods write checkpoints to the PFS in the background of application computation, which can reduce checkpointing overhead experienced by applications. With asynchronous checkpointing, after each process writes its checkpoint data to RAM or node-local storage, it can continue its computation. Another process or thread reads the checkpoint from the storage location, and writes it to the PFS. Although asynchronous checkpointing can increase an application's runtime due to resource contention from the background checkpoint transfer process, it resolves the blocking problem of synchronous checkpointing.

Intuitively, one would expect asynchronous checkpointing to be more efficient than synchronous checkpointing. However, our earlier study showed that simple asynchronous checkpointing can inflate an application's runtime and can be worse than synchronous checkpointing [4]. But with our asynchronous checkpointing system using RDMA, we minimized the inflated overhead, and showed that the asynchronous checkpointing is more efficient given current and future failure rates, and expected checkpointing overhead. Thus, in this paper, we explore only asynchronous checkpointing.

C. Coordinated vs. Uncoordinated Checkpointing

Last, we consider whether C/R is coordinated or uncoordinated. With coordinated C/R, all processes globally synchronize before taking checkpoints to ensure the checkpoints are consistent and that no messages are in flight. Coordinated C/R is applicable to a wide range of applications. However, at large scales the global synchronization can cause overhead due to propagation of system noise at extreme scale [17]. In

addition, when checkpointing to or restarting from a PFS, tens of thousands of compute nodes concurrently write or read checkpoints, which can cause large overhead due to contention. Meanwhile, uncoordinated checkpointing [10] does not require global synchronization, and allows processes to write/read checkpoints at different times, which lowers checkpoint overhead. However, with uncoordinated checkpointing there may be messages in flight from one process to another when a checkpoint is taken. To handle this, uncoordinated checkpointing libraries log messages, which has its own overhead problem. This protocol can cause the so-called *domino effect* preventing an application from rolling-back to the last checkpoint at restart [18] without message logging.

Earlier uncoordinated checkpoint techniques [11], [12] reduce the message logging overhead by partitioning processes into clusters, and only logging the inter-cluster communications. Although the clustering approach can reduce message logging overhead while minimizing the number of processes that need to restart on failure, application runtime is still inflated by the logging overhead. In addition, if we apply uncoordinated checkpointing to MPI applications, *indirect global synchronization* can occur. In Figure 1, for example, process (a2) in cluster (A) wants to send a message to process (b1) in cluster (B), which is writing its checkpoint at that time. Process (a2) waits for process (b1) because process (b1) is doing I/O and can not receive or reply to any messages, which keeps process (a1) waiting to checkpoint with process (a2). If such a dependency propagates across all processes, it results in indirect global synchronization. Many MPI applications exchange messages between processes in a shorter period of time than is required for checkpoints. In uncoordinated checkpointing, we assume applications restart with uncoordinated manner (partial restart), but globally synchronize before checkpointing like coordinated checkpointing. Thus, as in the model specified in Section V, we estimate the times for uncoordinated checkpointing and for coordinated checkpointing in the same model.

D. Target C/R Strategies

As discussed previously, multilevel and asynchronous software-level approaches are more efficient than single and synchronous C/R respectively. However, there is a trade-off between coordinated and uncoordinated checkpointing given an application and the configuration. In this work, we compare the efficiency of multilevel asynchronous coordinated and uncoordinated C/R. However, because we have already found that these software-level approaches may be limited in increasing application efficiencies at extreme scale [4], we also consider storage architecture approaches.

III. Storage designs

Our goal is to achieve a more reliable system with more efficient application executions. Thus, we consider not only a software approach via C/R techniques (software-level approach), but also consider different storage architectures (architecture-level approach). In this section, we introduce an mSATA-based SSD burst buffer system (*Burst buffer system*), and explore the advantages by comparing to a representative current storage system (*Flat buffer system*).

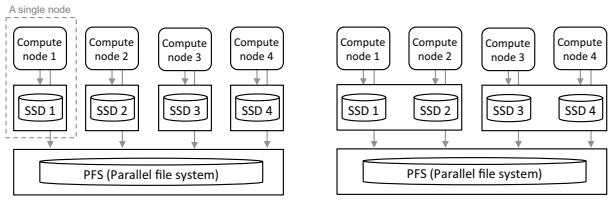

Fig. 2. (a) Left: Flat buffer system (b) Right: Burst buffer system

A. Current Flat Buffer System

In a flat buffer system (Figure 2 (a)), each compute node has its dedicated node-local storage, such as an SSD, so this design is scalable with increasing number of compute nodes. Several supercomputers employ this flat buffer system [19], [20], [21]. However this design has drawbacks: unreliable checkpoint storage and inefficient utilization of storage resources for partial restart. Storing checkpoints in node-local storage is not reliable because an application cannot restart its execution if a checkpoint is lost due to a failed compute node, which is the most common failure [4], [7]. For example, if compute node 1 in Figure 2 (a) fails, a checkpoint on SSD 1 will be lost because SSD 1 is connected to the failed compute node 1. In addition, storage devices can be underutilized with partial restart by uncoordinated C/R. While the system can limit the number of processes to restart, i.e., perform a partial restart, in a flat buffer system, local storage is not utilized by processes which are not involved in the partial restart. For example, if compute node 1 and 3 are in a same cluster for uncoordinated C/R, and restart from a failure, the bandwidth of SSD 2 and 4 will not be utilized.

B. Burst Buffer System

To solve the problems in a flat buffer system, we consider a burst buffer system [22]. A burst buffer is a storage space to bridge the gap in latency and bandwidth between node-local storage and the PFS, and is usually shared by a subset of compute nodes. Additional nodes are required for burst buffers, and the increasing number of nodes may inflate the overall failure rate. However, a burst buffer can offer a system many advantages including higher reliability and efficiency over a flat buffer system. A burst buffer system is more reliable for checkpointing because burst buffers are located on a smaller number of dedicated I/O nodes, so the probability of lost checkpoints is decreased. In addition, even if a large number of compute nodes fail concurrently, an application can still access the checkpoints from the burst buffers. A burst buffer system provides more efficient utilization of storage resources for partial restart of uncoordinated C/R because processes involving restart can exploit higher storage bandwidth. For example, if compute node 1 and 3 are in the same cluster, and both restart from a failure, the processes can utilize all SSD bandwidth (Bandwidth of SSD 1 and 2 for node 1, SSD 3

and 4 for node 3) unlike a flat buffer system. This capability accelerates the partial restart of uncoordinated C/R.

TABLE I
NODE SPECIFICATION

CPU	Intel Core i7-3770K CPU (3.50 GHz x 4 cores)
Memory	Cetus DDR3-1600 (16 GB)
M/B	GIGABYTE GA-Z77X-UD5H
SSD	Crucial m4 CT256M4SSD3 (256GB, mSATA) (Peak read: 500 MB/s, Peak write: 260 MB/s)
SATA converter	KOUTECH IO-ASS110 mSATA to 2.5' SATA Device Converter with Metal Frame
RAID Card	Adaptec ASR-7805Q Single

To build a reliable burst buffer system at extreme scale, minimizing the number of system components is critical while maximizing high I/O throughput under minimal budget. To explore the bandwidth we can achieve by a single node with only commodity devices, we developed an mSATA-based SSD test system. The detailed specification is shown in Table I. The theoretical peak throughputs of sequential read and write operation of the mSATA-based SSD is 500 MB/sec and 260 MB/sec, respectively. We aggregate the eight SSDs into a RAID card, and connect the RAID cards via PCIe 3.0 (x8). The theoretical peak performance of this configuration is 4 GB/sec for read and 2.08 GB/sec for write in total. To use the storage systems as burst buffers, the mSTA-based SSDs must be accessed via a high-speed network (e.g., InfiniBand) with a network-based file system. However, simple methods cannot exploit the bandwidth. For example, if we use NFS with IPoIB for the network-based file system, the useful bandwidth is only 1 GB/s for both read and write (details in Section VI). A new network-based file system is required to exploit the PCIe-attached high bandwidth storage.

IV. USER-LEVEL INFINIBAND-BASED FILE SYSTEM FOR BURST BUFFERS

To exploit the bandwidth of burst buffers, we developed a user-level InfiniBand-based file system and I/O API called IBIO. Our earlier work showed that I/O operations with concurrent multiple threads can effectively exploit the high bandwidth of PCIe-attached storage [23] as well as the PFS [4]. Thus, we parallelize the operations of IBIO with multiple reader and writer threads. The current API of IBIO includes `open`, `write`, `read` and `close`. The interfaces are identical to POSIX except that of `open`. The IBIO `open` requires a *hostname* as well as a *pathname* so that IBIO clients can access any files on any IBIO servers. IBIO `open` sends a query to the IBIO server to open the file, and it returns the file descriptor (`fd`).

Figure 3 and Figure 4 show the design of IBIO `write` and `read`. When a client on a compute node writes its file to remote storage via the IBIO `write` call (Figure 3), the IBIO client divides the data into smaller chunks, and it transfers the chunks using an RDMA API we developed in prior work [4] (Step 1). The RDMA transfer API enables one-sided

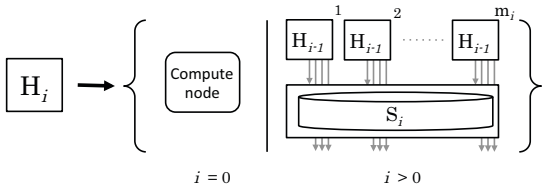

Fig. 5. Recursive structured storage model

communications from the client to the server using a low-level, user-space InfiniBand API called *ibverbs*. Once the IBIO server thread receives a write request from a client, the IBIO server reads the chunk into the selected buffer according to the `fd` by using an RDMA read (Step 2). Then, the IBIO server creates a *writer thread* to asynchronously write the chunk to the file (Step 3) so that the IBIO server thread can receive subsequent chunks from IBIO clients while writing the chunk. When all chunks for the write call are written, the writer threads inform the IBIO server (Step 4), and the IBIO server informs the IBIO client that the write is complete (Step 5).

Since communications from server to client are required for IBIO read operations, we extended the RDMA transfer API from prior work to support bi-directional communications. When an IBIO client reads a file from remote storage via IBIO `read` (Figure 4), the IBIO client sends the read request containing the `fd` to the IBIO server (Step 1). Once the IBIO server receives the read request, the IBIO server thread identifies the file according to the `fd`, and it creates a *reader thread* to handle the read request (Step 2). Then the IBIO server thread waits for the next request. The reader thread reads a chunk of the file with file descriptor `fd` into its buffer (Step 3). The reader thread requests that the IBIO server thread sends the chunk to the IBIO client (Step 4), and then reads the next chunk. When all chunks for the read call have been read, the reader thread informs the IBIO server, and the IBIO server informs the IBIO client that the read is complete (Step 5).

V. MODELING

As described in Sections II and III, each checkpoint strategy and storage architecture have advantages and disadvantages. Here we discuss the model we developed to identify the best checkpoint strategy for a given configuration.

A. Recursive Structured Storage Model

We introduce a recursive structured storage model to generalize storage architectures to describe both flat and burst buffer systems with a single model. Figure 5 shows the recursive structured storage model based on a *restricted context-free grammar*. A tier i hierarchical entity, H_i, has storage S_i shared by m_i upper hierarchical entities, H_{i-1}. We denote $H_{i=0}$ as a compute node. If each tier of hierarchical storage is shared as $\{m_1, m_2, \ldots, m_N\}$ within N-tired hierarchical storage, we denote the storage architecture as $H_N\{m_1, m_2, \ldots, m_N\}$. For example, the flat buffer system in Figure 2 (a) can be

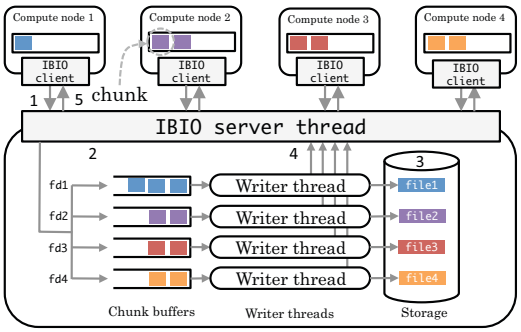

Fig. 3. IBIO Write: four IBIO clients and one IBIO server

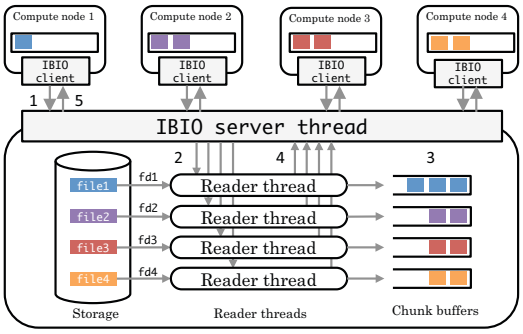

Fig. 4. IBIO Read: four IBIO clients and one IBIO server

represented as $H_2\{1,4\}$. It has 2 levels of storage: the node-local storage is not shared, so $m_1 = 1$; however, the PFS is shared across all compute nodes, so $m_2 = 4$. In the same manner, the burst buffer system in Figure 2 (b) can be represented as $H_2\{2,2\}$. The total number of compute nodes can be calculated as $\prod_{i=1}^{2} m_i = 4$ nodes.

TABLE II
TIER i STORAGE (S_i) PERFORMANCE PARAMETERS

r_i	Sequential read throughput from compute nodes ($H_{i=0}$)
w_i	Sequential write throughput from compute nodes ($H_{i=0}$)
m_i	The number of a upper hierarchical entities (H_{i-1}) sharing S_i

In this model, we do not distinguish between node-local storage and network-attached storage. Instead, we differentiate the storage levels using performance parameters. We consider only sequential read/write bandwidth because typically the I/O pattern of C/R is sequential. Note that the read and write bandwidth values are not the peak performance of the storage but the effective throughput between compute nodes and the storage location. For example, if tier i storage has a read bandwidth of r, but the network-based file system delivers a bandwidth of $\hat{r} < r$, then the model parameter is set as $r_i = \hat{r}$. Using these performance parameters, we estimate C/R time. However, as we show in Section VI, We can minimize the performance gap between local and remote read/write accesses with IBIO.

B. Modeling of C/R Strategies

Given the storage performance parameters of each tier, we model level i *checkpoint overhead* (O_i), *checkpoint latency* (L_i), and *restart overhead* (R_i) in a multilevel checkpointing library [4]. For simplicity, if multiple compute nodes concurrently access a single storage location, we assume the read/write throughput scales down linearly with the number of concurrent accesses.

Checkpoint overhead O_i is the increased execution time of an application because of checkpointing. Checkpoint latency L_i and restart overhead R_i are the times to complete a checkpoint and restart respectively. If a checkpoint strategy conducts erasure encoding, such as XOR [7] and Reed-Solomon encoding [8], the checkpoint overhead and latency also include the encoding overhead and latency. We differentiate between checkpoint overhead and latency to show the differences between synchronous and asynchronous checkpointing. During synchronous checkpointing, checkpoint overhead and latency are equal, i.e., $O_i = L_i$, because each process is blocked until the checkpoint is completed. Asynchronous checkpointing, meanwhile, incurs only initialization overhead, so checkpoint overhead is equal or smaller than checkpoint latency, i.e., $O_i < L_i$.

We model level i checkpoint overhead and latency as

$$O_i = \begin{cases} C_i + E_i & \text{(synchronous checkpointing)} \\ I_i & \text{(asynchronous checkpointing)} \end{cases}$$

$$L_i = C_i + E_i$$

where C_i denotes actual checkpointing time, E_i denotes encoding time, and I_i denotes initialization time for asynchronous checkpointing. If the level i checkpointing does not encode checkpoints, E_i becomes 0; otherwise we model the encoding time as $E_i = D/e_i$ where D is the checkpoint size per compute node and e_i is encoding throughput. The actual checkpointing time C_i, i.e., sequential write time, is calculated as

$$C_i = \begin{cases} D \times M/w_i & (i = N) \\ D \times \left\lceil \frac{M}{\prod_{k=i+1}^{N} m_k} \right\rceil / w_i & (otherwise) \end{cases}$$

where M denotes the total number of checkpointing compute nodes, i.e., $M = \prod_{i=1}^{N} m_i$. With uncoordinated checkpointing, we assume the checkpointing time is identical to coordinated checkpointing time because of indirect global synchronization as described in Section II-C. Because $\prod_{k=i+1}^{N} m_k$ is the number of storage locations S_i, the quantity $\left\lceil \frac{M}{\prod_{k=i+1}^{N} m_k} \right\rceil$ represents the max number of compute nodes per storage location S_i.

When restarting with uncoordinated checkpointing, the restart overhead is different from coordinated checkpointing. We model the restart overhead R_i, i.e., sequential read time, as:

$$R_i = \begin{cases} D \times K/r_i & (i = N) \\ D \times \left\lceil \frac{K}{\prod_{k=i+1}^{N} m_k} \right\rceil / r_i & (otherwise) \end{cases}$$

where K is the number of restarting compute nodes. With coordinated restart, all compute nodes concurrently read their checkpoints, so K is identical to the total number of compute nodes M. With uncoordinated restart, only the cluster which includes failed compute nodes will read its checkpoints and restart. Here, K is the cluster size, and we assume that each compute node in a cluster is distributed across $S_{i>N}$ storage locations with a topology-aware process mapping technique.

C. Multilevel Asynchronous C/R Model

Our multilevel asynchronous C/R model [4] computes the expected runtime $\hat{T}$ given the checkpoint overheads at each storage level $O = \{O_1, O_2, \ldots\}$, the checkpoint latencies $L = \{L_1, L_2, \ldots\}$, the restart overheads $R = \{R_1, R_2, \ldots\}$, the failure rates $F = \{F_1, F_2, \ldots\}$, the checkpoint frequencies $V = \{v_1, v_2, \ldots\}$, and the checkpoint interval T. Here, v_i is the number of level i checkpoints within each level i + 1 period. For example, if an application writes fifteen level 1 checkpoints for every level 2 checkpoint, and five level 2 checkpoints for every level 3 checkpoint, V is $\{15, 5, 1\}$. Given these parameters, we can compute the optimal checkpoint interval by minimizing $\hat{T}$, where $f(O, L, R, F, V, T) \Rightarrow \hat{T}$.

To evaluate the checkpoint strategies given a storage configuration, we use *efficiency* defined as

$$\textit{efficiency} = \frac{\textit{ideal time}}{\textit{expected time}} = \frac{I}{\hat{T}}.$$

I is the minimum run time assuming the application spends no time in checkpointing activities and encounters no failures. So I is simply computed as:

$$\begin{aligned} I &= T \times (v_1 + 1) \times \cdots \times (v_{N-1} + 1) \\ &= T \cdot \prod_{i=1}^{N-1} (v_i + 1). \end{aligned}$$

We use this metric to compare the checkpoint strategies. Our earlier study provides more details of the model [4].

VI. Performance of IBIO

For a burst buffer system, it is important to exploit the high bandwidth of the burst buffers through network access. To evaluate this property of IBIO we conducted sequential read and write tests using the mSATA-based SSD system described in Section III-B. Figure 6 shows sequential read and write throughputs of local I/O, and I/O with IBIO and NFS. We connected the mSATA-based SSD system to an InfiniBand network with a Mellanox FDR HCA (Model No.: MCX354A-FCBT) for remote access evaluations of IBIO and NFS. We use 64 MB for the maximal chunk size to read and write files to and from the NFS and IBIO servers to maximize the sequential read and write throughputs. In NFS, the parameters are configured by `rsize` and `wsize` options.

We find that the read and write throughputs of IBIO are almost identical to the local throughputs with 8 processes or more. When the number of read processes is 4 or less, we see performance degradation of read operations on IBIO (Read-IBIO) compared to the local read access (Read-Local). In IBIO read, the client first sends a message to request the first 64 MB chunk of the file, and the constant request latency is added to the read time. Because of the constant latency, the read throughput decreases with a smaller number of clients. However, when applications write and read checkpoints, multiple processes concurrently access the storage, so applications can exploit the bandwidth. In contrast, when IBIO clients write data to a file, the clients send the first chunk with the first request message. Thus, the write throughput does not decrease with the fewer processes.

We also evaluate the sequential read and write performance of NFS. Because NFS runs on IPoIB, NFS incurs as much as 49.7% and 35.3% performance degradation compared to local read and write, respectively, while IBIO incurs only 4.3 % over local read and 2.7 % over local write. NFS can not exploit the FDR network bandwidth, because IPoIB becomes a bottleneck. In contrast, IBIO uses the low-level InfiniBand API (*ibverbs*). Thus IBIO can minimize data transfer overhead even over networks.

TABLE III
Simulation configuration

	Flat buffer system	Burst buffer system
$H_2\{m_1, m_2\}$	$H_2\{1, 1088\}$	$H_2\{32, 34\}$
$\{r_1, r_2\}$	{500 MB/s, 10 GB/s}	{16 GB/s, 10 GB/s}
$\{w_1, w_2\}$	{260 MB/s, 10 GB/s}	{8.32 GB/s, 10 GB/s}
$\{e_1, e_2\}$	{400 MB/s, N/A}	
D	5 GB	
$\{F_1, F_2\}$	{2.14e-6, 4.28e-7}	{2.63e-6, 1.33e-8}

VII. Resiliency Exploration

In this section, we evaluate the trade-offs of different checkpointing and storage configurations.

A. Experimental Setup

We describe our experimental setup including configuration details for C/R and storage configurations, and how we determine the failure rates to use in our model.

1) C/R and Storage Configuration: In this study, we evaluate multilevel C/R on a 2-tiered storage system. Table III shows the base configuration. The system sizes (H_i) are based on the Coastal cluster at LLNL, which is an 88.5 TFLOP theoretical peak system consisting of 1,088 batch nodes. We use 500 MB/s for local read throughput, and 260 MB/s for local write throughput for the Flat buffer system. The burst buffer system has 34 burst buffer nodes, each of which is shared by 32 compute nodes. In this exploration, we simulate that we aggregate the 32 storage into the single burst buffer node, and connect the burst buffer nodes via a high speed interconnect which does not create a bottleneck in bandwidth to simulate future extreme scale system. With IBIO, applications can remotely access the burst buffers with almost the same throughput as local access throughput. Thus,

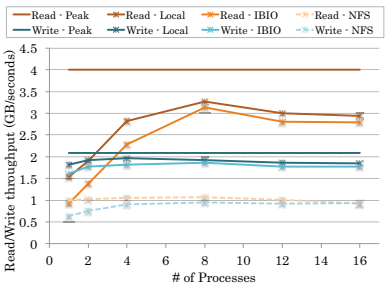

Fig. 6. Sequential read and write throughput of local I/O, and I/O with IBIO and NFS via FDR InfiniBand networks

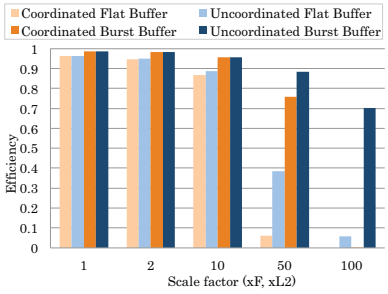

Fig. 7. Efficiency of multilevel coordinated and uncoordinated C/R on a flat buffer system and a burst buffer system

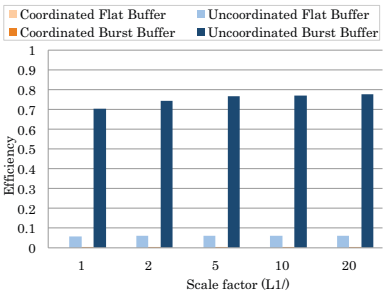

Fig. 8. Efficiency in increasing level-1 C/R performance in x100 failure rate: L1 C/R time/scale factor

we use 16 GB/s for read throughput, and 8.32 GB/s for write throughput in the burst buffer system.

For uncoordinated checkpointing, we use 16 nodes for the cluster size (K). Earlier studies showed that the optimal cluster size is from 32 to 128 processes, i.e., 4 to 16 nodes for a 8-core Coastal compute node, to provide a good trade-off between the size of the clusters and the amount of messages to log for most applications [11], [24]. Because the cluster size is small enough to assign a compute node to a single burst buffer node, $\left\lceil \frac{K}{\prod_{k=2}^{2} m_k} \right\rceil$ is computed as 1 compute node for uncoordinated restart.

We use asynchronous checkpointing for PFS, and synchronous checkpointing for XOR. For the encoding rate, we only provide an encoding rate (e_1) for level 1 (XOR) because PFS does not need encoding.

2) Failure Rate Estimation: Failure rates (F) are based on a failure analysis study using a multilevel C/R library called the Scalable C/R (SCR) Library [7]. SCR provides several checkpoint options: LOCAL, XOR, and PFS. With LOCAL, SCR simply writes the checkpoint data to node-local storage. In this case, if one of the checkpoints is lost due to a failure, an application would not be able to restart its execution. So, SCR provides XOR, which is a RAID-5 strategy that computes XOR parity across subgroups of processes so that SCR can restore the lost checkpoint data. SCR also provides PFS to keep checkpoint data on the most reliable storage level, the PFS. The failure analysis study shows that the average failure rate (# of failures/second) per a single compute node requiring LOCAL is 1.96×10^{-10}, XOR is 1.77×10^{-9}, and PFS is 3.93×10^{-10}.

In a flat buffer system, each failure rate is calculated by simply multiplying the failure rate by the number of compute nodes, 1088 nodes. This leads to failure rates of 2.14×10^{-7} (= $1.96 \times 10^{-10} \times 1,088$) for LOCAL, 1.92×10^{-6} (= $1.77 \times 10^{-9} \times 1,088$) for XOR, and 4.28×10^{-7} (= $3.93 \times 10^{-10} \times 1,088$) for PFS. The failure rates are identical to the measured ones of the LLNL Coastal cluster. Actually, if the level-i failure rate is lower than the level-$i+1$ rate, the optimal level i checkpoint count is zero because a level i failure can be recovered with a level $i+1$ checkpoint, which is written more frequently than level i [21] . Thus, we do not consider LOCAL checkpointing for the simulation. We evaluate the two level C/R case where level 1 is XOR, and level 2 is PFS, with failure rates of $\{F_1,\ F_2\} = \{2.14 \times 10^{-6}, 4.28 \times 10^{-7}\}$ (See Table IV).

In a burst buffer system, we use 34 burst buffer nodes, and assume the failure rate of a burst buffer node is identical to a compute node. On a compute node failure, an application does not lose checkpoint data because the checkpoint data is not in compute nodes. However, if a burst buffer node fails, checkpoint data on the failed burst buffer nodes is lost. Thus, we also use two level C/R where level 1 is XOR, and level 2 is PFS. Because the total number of nodes increases, failure rate requiring level 1 checkpoint increases according to the number of burst buffer nodes. For 34 burst buffer nodes, the level 1 failure rate is calculated as 6.67×10^{-8} (= $(1.96 \times 10^{-10} + 1.77 \times 10^{-9}) \times 34$). Meanwhile, checkpoint data is stored on fewer nodes, which decreases the failure rate requiring PFS for recovery. The level 2 failure is 1.33×10^{-8} (See Table IV). On compute node failures, application can restart from level 1 checkpoint regardless of the number of failed compute nodes in a burst buffer system. Thus, the failure rate of each level is $\{F_1,\ F_2\} = \{2.63 \times 10^{-6}, 1.33 \times 10^{-8}\}$ for the burst buffer system. Because the burst buffer system uses more nodes for burst buffers, the overall failure rate of the burst buffer system (2.64×10^{-6}) is higher than one of the flat buffer system (2.57×10^{-6}).

B. Efficiency with Increasing Failure Rates and Level 2 C/R Costs

We expect the failure rates and aggregate checkpoint sizes to increase on future extreme scale systems. To explore the effects, we increase failure rates and level 2 (PFS) C/R costs by factors of 1, 2, 10, 50 and 100 from the base configuration in Table III, and compare the efficiencies of multilevel coordinated and uncoordinated C/R on a flat buffer system and on a burst buffer system. We do not change the level 1 (XOR) checkpoint cost; because the total performance of node-local storage and burst buffer will scale with increasing system size. For uncoordinated C/R, we assume that the message logging overhead is 0. We discuss the allowable message logging overhead in Section VII-C.

Figure 7 shows application efficiency under increasing failure rates (xF) and level 2 C/R costs (xL2). When we

TABLE IV
CHECKPOINT LEVELS AND FAILURE RATES

		Flat Buffer System	Burst Buffer System
Level 1	C/R method	XOR on local store	XOR on burst buffer
	Failure rate (# of failures / second)	$(1.96 \times 10^{-10} + 1.77 \times 10^{-9}) \times 1,088$ $= 2.14 \times 10^{-6}$	$F_{flat} + (1.96 \times 10^{-10} + 1.77 \times 10^{-9}) \times 34$ $= 2.63 \times 10^{-6}$
Level 2	C/R method	PFS	PFS
	Failure rate (# of failures / second)	$(3.93 \times 10^{-10}) \times 1,088$ $= 4.28 \times 10^{-7}$	$(3.93 \times 10^{-10}) \times 34$ $= 1.33 \times 10^{-8}$
Overall failure rate (Level 1+2)		$2.57 \times 10^{-6} (= F_{flat})$	2.64×10^{-6}

compute efficiency, we optimize the level-1 and 2 checkpoint frequencies (v_1 and v_2), and the interval between checkpoints (T) to discover the maximal efficiency. The burst buffer system achieves a higher efficiency than the flat buffer system in most cases. The efficiency gap becomes more apparent with higher failure rates and higher checkpoint costs because the burst buffer system stores checkpoints on fewer burst buffer nodes. By using uncoordinated C/R and leveraging burst buffers, we achieve 70% efficiency even on systems that are two orders of magnitude larger. This is because partial restart with uncoordinated checkpointing can limit the number of compute nodes that read checkpoints on burst buffers and the PFS, which accelerates restart time.

C. Allowable Message Logging Overhead

The efficiencies shown in Figure 7 do not include message logging overhead. We consider this factor in Table V which shows how much message logging overhead of uncoordinated checkpointing is allowed in order to achieve a higher efficiency than coordinated checkpointing. As in Figure 7, we increase both the failure rates and level 2 C/R cost by the scale factor shown on each row. We find that the logging overhead must be relatively small, less than a few percent, for scale factors up to 10. However, at scale factors of 50 and 100, very high message logging overheads are tolerated. This shows that uncoordinated checkpointing can be more efficient on future systems even with high logging overheads.

D. Effect of Improving Storage Performance

When building a reliable data center or supercomputer, significant efforts are made to maximize system performance given a fixed budget. It can be challenging to decide which system resources will most affect overall system performance.

TABLE V
ALLOWABLE MESSAGE LOGGING OVERHEAD

Flat buffer		Burst buffer	
scale factor	Allowable message logging overhead	scale factor	Allowable message logging overhead
1	0.0232%	1	0.00435%
2	0.0929%	2	0.0175%
10	2.45%	10	0.468%
50	84.5%	50	42.0%
100	$\approx$ 100%	100	99.9%

To explore how the performance of different tiers of the storage hierarchy impact system efficiency, we increase performance of each tier of storage by factors of 1, 2, 10, and 20. Figures 8 and 9 show efficiency with increasing performance of level 1 and 2 C/R, i.e., decreasing level 1 and 2 C/R time, using failures rates at 100 $\times$ current rates. We see that improvement of level 1 C/R does not impact efficiency for either flat buffer or burst buffer systems. However, as shown in Figure 9, increasing the performance of the PFS does impact system efficiency. We can achieve over 80% efficiency with both coordinated and uncoordinated C/R on the burst buffer system with improved PFS performance of 10 and 20 $\times$. These results tell us that level 2 C/R overhead is a major cause of degrading efficiency, and its performance affects the system efficiency much more than that of level 1. We also find that prevention of level 2 failures is important for future extreme scale systems.

E. Optimal Ratio of Compute Nodes to Burst Buffer Nodes

Another thing to consider when building a burst buffer system is the ratio of compute nodes to burst buffer nodes. A large number of burst buffer nodes can increase the total bandwidth, but the large node counts increase the overall failure rate of the system. To explore the effect of the ratio of compute node and burst buffer node counts, we evaluate efficiency under different failure rates and level 2 C/R costs while keeping I/O throughput of a single burst buffer node constant. Figures 10 and 11 show the results with coordinated and uncoordinated C/R. We see that the ratio is not significant up to scale factors of 10 $\times$. However, at a scale factor of 50 $\times$, a larger number of burst buffer nodes decreases efficiency. Adding additional burst buffer nodes increases the failure rate which degrades system efficiency more than the efficiency gained by the increased bandwidth. Thus, increasing the number of compute nodes sharing a burst buffer node is optimal as long as the burst buffer throughput can scale to the number of sharing compute nodes.

VIII. RELATED WORK

Fast C/R is important for an application running for days and weeks at extreme scale to achieve efficient execution in the presence of failures. Multilevel C/R [7], [8] is an approach for increasing application efficiency. Multilevel checkpoint libraries utilize multiple tiers of storage, such as node-local storage and the PFS. Uncoordinated C/R [10], [11], [12]

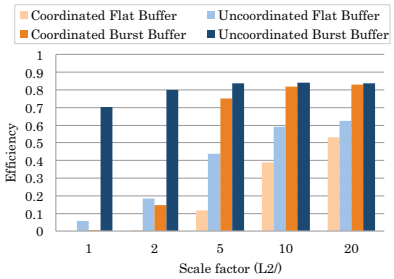

Fig. 9. Efficiency in increasing level-2 C/R performance in x100 failure rate: L2 C/R time/scale factor

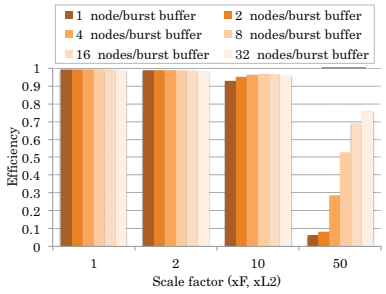

Fig. 10. Coordinated: Efficiency in different ratios of compute nodes to a single burst buffer nodes with coordinated C/R

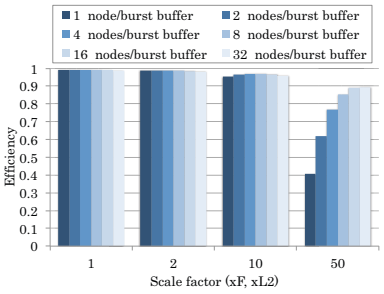

Fig. 11. Uncoordinated: Efficiency in different ratios of compute nodes to a single burst buffer nodes with uncoordinated C/R

works effectively when coupled with multilevel C/R. The approach can limit the number of processes that need to be restarted, i.e., only a partial restart instead of the whole job, which can decrease restart time from shared file system resources, such as a PFS or burst buffer. These techniques can be improved further when coupled with incremental checkpointing [25], [26], and checkpoint compression [27], [28]. Over the years, many C/R strategies have been studied, but there is no studies that model and evaluate the efficiency of the wide-range of C/R strategies.

The state-of-the art C/R strategies themselves are limited in their ability to improve application efficiency at extreme scale because C/R time depends on underlying I/O storage performance. Another approach is to accelerate I/O performance itself by altering the storage architecture. Adding a new tier of storage is one solution. Rajachandrasekar et al. [29] presented a staging server which drains checkpoints on compute nodes using RDMA (Remote Direct Memory Access), and asynchronously transfers them to the PFS via FUSE (Filesystem in Userspace). Hasan et al. [9] achieved high I/O throughput by using additional nodes. To deal with bursty I/O requests, Liu et al. [22] proposed a storage system that integrates SSD buffers on I/O nodes. As we observed, optimizing performance and reliability requires determination of the proper number of burst buffers for a given number of compute nodes. However, a comprehensive study on the problem has not yet been done.

Wickberg et al. [30] introduced an aggregated DRAM buffer on top of the PFS called RAMDISK Storage Accelerator (RSA). RSA constructs a low latency and high bandwidth buffer on the fly, and asynchronously stages in files ahead of execution coupled with an I/O scheduler. Kannan et al. [31] also presented a data staging approach using active NVRAM (Non-volatile RAM) [32] technology. These studies focused on only I/O throughput. As we have seen, storing application's data as well as checkpoints in a fewer number of extra nodes is a reliable solution. Our model evaluates the system efficiency and is useful for designing future storage architectures at extreme scale. To the best our knowledge, our work is the first focusing on a co-designed approach for increasing both I/O throughput and reliability with burst buffers at extreme scale.

IX. Conclusion

In this work, we explored the use of burst buffer storage for scalable C/R, and developed IBIO to exploit the high bandwidth of the burst buffers for future extreme scale systems. We also developed a model to explore the performance difference of checkpointing strategies, specifically coordinated and uncoordinated checkpointing. We used the model to evaluate multilevel checkpointing on flat buffer storage systems that are currently available on today's machines and hierarchical storage systems using burst buffers.

From our exploration, we found that burst buffers are indeed beneficial for C/R on future systems, increasing reliability and efficiency. We also found that the performance of the parallel file system has a high impact on the efficiency of a machine, while increased bandwidth to burst buffers did not affect overall machine efficiency. However, the reliability of burst buffers does impact efficiency, because unreliable buffers mean more I/O traffic to a PFS when multiple burst buffer nodes fail, and checkpoints on the failed burst buffer nodes are lost. Overall, uncoordinated checkpointing was more efficient than coordinated checkpointing, even with high message logging overhead. These findings can benefit system designers in making the trade-offs in performance of components so that they can create efficient and cost-effective machines.

Acknowledgment

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. (LLNL-CONF-645876). This work was also supported by Grant-in-Aid for Research Fellow of the Japan Society for the Promotion of Science (JSPS Fellows) 24008253, and Grant-in-Aid for Scientific Research S 23220003.

References

[1] B. Schroeder and G. A. Gibson, "Understanding Failures in Petascale Computers," *Journal of Physics: Conference Series*, vol. 78, no. 1, pp. 012 022+, Jul. 2007. [Online]. Available: http://dx.doi.org/10.1088/1742-6596/78/1/012022

[2] A. Geist and C. Engelmann, "Development of Naturally Fault Tolerant Algorithms for Computing on 100,000 Processors," 2002.

[3] J. Daly *et al.*, "Inter-Agency Workshop on HPC Resilience at Extreme Scale," February 2012. [Online]. Available: http://institutes.lanl.gov/resilience/docs/Inter-AgencyResilienceReport.pdf

[4] K. Sato, N. Maruyama, K. Mohror, A. Moody, T. Gamblin, B. R. de Supinski, and S. Matsuoka, "Design and Modeling of a Non-Blocking Checkpointing System," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Salt Lake City, Utah: IEEE Computer Society Press, 2012. [Online]. Available: http://portal.acm.org/citation.cfm?id=2389022

[5] R. L. Berger, C. H. Still, E. A. Williams, and A. B. Langdon, "On the Dominant and Subdominant Behavior of Stimulated Raman and Brillouin Scattering Driven by Nonuniform Laser Beams," *Physics of Plasmas*, vol. 5, p. 4337, 1998.

[6] J. Dongarra *et al.*, *Int. J. High Perform. Comput. Appl.*, vol. 25, no. 1, pp. 3–60, Feb. 2011. [Online]. Available: http://dx.doi.org/10.1177/1094342010391989

[7] A. Moody, G. Bronevetsky, K. Mohror, and B. R. de Supinski, "Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. Washington, DC, USA: IEEE Computer Society, Nov. 2010, pp. 1–11. [Online]. Available: http://dx.doi.org/10.1109/sc.2010.18

[8] L. Bautista-Gomez, D. Komatitsch, N. Maruyama, S. Tsuboi, F. Cappello, and S. Matsuoka, "FTI: High Performance Fault Tolerance Interface for Hybrid Systems," in *Proceedings of the 2011 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, Seattle, WS, USA, 2011.

[9] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng, "DataStager: Scalable Data Staging Services for Petascale Applications," in *Proceedings of the 18th ACM international symposium on High performance distributed computing*, ser. HPDC '09. New York, NY, USA: ACM, 2009, pp. 39–48. [Online]. Available: http://dx.doi.org/10.1145/1551609.1551618

[10] A. Bouteiller, T. Herault, G. Bosilca, and J. J. Dongarra, "Correlated Set Coordination in Fault Tolerant Message Logging Protocols," in *Proceedings of the 17th international conference on Parallel processing - Volume Part II*, ser. Euro-Par'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 51–64. [Online]. Available: http://portal.acm.org/citation.cfm?id=2033415

[11] T. Ropars, A. Guermouche, B. Uçar, E. Meneses, L. V. Kalé, and F. Cappello, "On the Use of Cluster-Based Partial Message Logging to Improve Fault Tolerance for MPI HPC Applications," in *Proceedings of the 17th international conference on Parallel processing - Volume Part I*, ser. Euro-Par'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 567–578. [Online]. Available: http://portal.acm.org/citation.cfm?id=2033406

[12] L. B. Gomez, T. Ropars, N. Maruyama, F. Cappello, and S. Matsuoka, "Hierarchical Clustering Strategies for Fault Tolerance in Large Scale HPC Systems," in *Proceedings of the 2012 IEEE International Conference on Cluster Computing*, ser. CLUSTER '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 355–363. [Online]. Available: http://dx.doi.org/10.1109/CLUSTER.2012.71

[13] N. Liu, J. Cope, P. H. Carns, C. D. Carothers, R. B. Ross, G. Grider, A. Crume, and C. Maltzahn, "On the Role of Burst Buffers in Leadership-Class Storage Systems," in *Symposium on Mass Storage Systems and Technologies, MSST 2012*, April 2012.

[14] D. Kimpe, K. Mohror, A. Moody, B. Van Essen, M. Gokhale, R. Ross, and B. R. de Supinski, "Integrated In-System Storage Architecture for High Performance Computing," in *Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers*, ser. ROSS '12, 2012.

[15] J. W. Young, "A First Order Approximation to the Optimum Checkpoint Interval," *Commun. ACM*, vol. 17, pp. 530–531, Sep. 1974. [Online]. Available: http://dx.doi.org/10.1145/361147.361115

[16] N. H. Vaidya, "On Checkpoint Latency," College Station, TX, USA, Tech. Rep., 1995. [Online]. Available: http://portal.acm.org/citation.cfm?id=892900

[17] T. Hoefler, T. Schneider, and A. Lumsdaine, "Characterizing the Influence of System Noise on Large-Scale Applications by Simulation," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–11. [Online]. Available: http://dx.doi.org/10.1109/SC.2010.12

[18] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A Survey of Rollback-Recovery Protocols in Message-Passing Systems," *ACM Computing Surveys*, vol. 34, no. 3, pp. 375–408, 2002.

[19] S. Matsuoka, T. Aoki, T. Endo, H. Sato, S. Takizawa, A. Nomura, and K. Sato, *TSUBAME2.0: The First Petascale Supercomputer in Japan and the Greatest Production in the World*. Chapman & Hall/CRC Computational Science, Apr. 2013, vol. 1, ch. 20, pp. 525–556. [Online]. Available: http://www.crcnetbase.com/doi/book/10.1201/b14677

[20] J. He, A. Jagatheesan, S. Gupta, J. Bennett, and A. Snavely, "DASH: A Recipe for a Flash-based Data Intensive Supercomputer," *ACM/IEEE conference on Supercomputing*, Nov. 2010.

[21] A. Moody, G. Bronevetsky, K. Mohror, and B. R. de Supinski, "Detailed Modeling, Design, and Evaluation of a Scalable Multi-level Checkpointing System," https://library- ext.llnl.gov, Lawrence Livermore National Laboratory, Tech. Rep., Jul. 2010.

[22] N. Liu, C. Jason, C. Philip, C. Christopher, R. Robert, G. Gary, C. Adam, and M. Carlos, "On the Role of Burst Buffers in Leadership-class Storage Systems," in *MSST/SNAPI*, Apr. 2012.

[23] T. Saito, K. Sato, H. Sato, and S. Matsuoka, "Energy-Aware I/O Optimization for Checkpoint and Restart on a NAND Flash Memory System," in *Proceedings of the 3rd Workshop on Fault-tolerance for HPC at extreme scale*, ser. FTXS '13. New York, NY, USA: ACM, 2013, pp. 41–48. [Online]. Available: http://doi.acm.org/10.1145/2465813.2465822

[24] A. Guermouche, T. Ropars, M. Snir, and F. Cappello, "HydEE: Failure Containment without Event Logging for Large Scale Send-Deterministic MPI Applications," in *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*. IEEE, May 2012, pp. 1216–1227. [Online]. Available: http://dx.doi.org/10.1109/ipdps.2012.111

[25] J. S. Plank, M. Beck, G. Kingsley, and K. Li, "Libckpt: Transparent Checkpointing under Unix," Knoxville, TN, USA, Tech. Rep., 1994.

[26] 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*, ser. ICS '04. New York, NY, USA: ACM, 2004, pp. 277–286. [Online]. Available: http://doi.acm.org/10.1145/1006209.1006248

[27] T. Z. Islam, K. Mohror, S. Bagchi, A. Moody, B. R. de Supinski, and R. Eigenmann, "McrEngine: A Scalable Checkpointing System Using Data-Aware Aggregation and Compression," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389020

[28] D. Ibtesham, D. Arnold, K. B. Ferreira, and P. G. Bridges, "On the Viability of Checkpoint Compression for Extreme Scale Fault Tolerance," in *Proceedings of the 2011 international conference on Parallel Processing - Volume 2*, ser. Euro-Par'11. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 302–311. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-29740-3_34

[29] R. Rajachandrasekar, X. Ouyang, X. Besseron, V. Meshram, and D. K. Panda, "Can Checkpoint/Restart Mechanisms Benefit from Hierarchical Data Staging?" in *Proceedings of the 2011 international conference on Parallel Processing - Volume 2*, ser. Euro-Par'11. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 312–321. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-29740-3_35

[30] T. Wickberg and C. Carothers, "The RAMDISK storage accelerator: a method of accelerating I/O performance on HPC systems using RAMDISKs," in *Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers*, ser. ROSS '12. New York, NY, USA: ACM, 2012, pp. 5:1–5:8. [Online]. Available: http://doi.acm.org/10.1145/2318916.2318922

[31] S. Kannan, A. Gavrilovska, K. Schwan, D. Milojicic, and V. Talwar, "Using Active NVRAM for I/O Staging," in *Proceedings of the 2nd international workshop on Petascal data analytics: challenges and opportunities*, ser. PDAC '11. New York, NY, USA: ACM, 2011, pp. 15–22. [Online]. Available: http://doi.acm.org/10.1145/2110205.2110209

[32] S. Kannan, D. Milojicic, V. Talwar, A. Gavrilovska, K. Schwan, and H. Abbasi, "Using Active NVRAM for Cloud I/O," *Open Cirrus Summit*, vol. 0, pp. 32–36, 2011.