# Burst SSD Buffer: Checkpoint Strategy at Extreme Scale

Kento Sato[1,2] Satoshi Matsuoka[1] Adam Moody[3] Kathryn Mohror[3] Todd Gamblin[3]
Bronis R. de Supinski[3] Naoya Maruyama[4]

***Abstract:*** Checkpointing is an indispensable fault tolerance technique, commonly used by HPC applications that run continuously for hours or days at a time. However, when checkpointing extreme scale systems, the bursty nature of the I/O pattern of checkpointing overburdens file systems and also causes huge overhead to be added to an application's runtime. In order to alleviate the overhead and achieve fast checkpoint/restart, we propose a highly-resilient mini-SSD-based burst buffer system, and explore a checkpoint strategy on the system based on our checkpointing model.

## 1. Introduction

The growing computational power of high performance computing (HPC) systems enables increasingly larger scientific simulations. However, as the number of system components increase, the overall failure rate of systems increases. Further, the mean time between failures (MTBF) of future systems is projected to be on the order of tens of minutes or hours [7], [10], [31]. In fact, an earlier failure analysis on Hera, Atlas and Coastal clusters at Lawrence Livermore National Laboratory (LLNL) [29] showed that a production application, the `pF3D` laser-plasma interaction code [4], experienced 191 failures out of 5-million node-hours. If we simply scale out the system while keeping the failure rate constant, the estimated MTBF is about 1.2 days for a 1,000-node cluster, 2.9 hours for a 10,000-node cluster, and 17 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 extreme scale systems, fault tolerance is becoming more important [8].

One indispensable fault tolerance technique is checkpoint/restart. The application writes a snapshot of its state to a reliable parallel file system (PFS) so that if a failure occurs, the application can restore the state from the snapshot. Although storing checkpoints in the PFS is highly reliable, this straightforward method can impose huge overheads on application run times at large scales. Multilevel checkpoint/restart is an approach to this problem [3], [23]. Multilevel checkpointing libraries generally cache checkpoints in in-system storage such as RAM or other node-local storage, and copy a select few to the more reliable PFS. This reduces the overhead of writing checkpoints in the common case, which can greatly increase application efficiency. In addition, combining multilevel checkpoint/restart with asynchronous I/O [1], [25], [30] or uncoordinated checkpointing [5], [11], [28] can result in more efficient execution in the presence of failures. However, even with these state-of-the-art checkpoint/restart techniques, high failure rates at extreme scale may limit the ability of the techniques to improve application efficiency. Using only distributed node-local storage for caching checkpoints is scalable. However, the approach is not viable at extreme scale due to node-local storage being generally unreliable in the event of failures.

The LLNL failure analysis study showed that most failures affect a single compute node. To tolerate node failures, multilevel checkpointing libraries generally apply redundancy schemes to the cached checkpoints. 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 used. However, with higher failure rates, the likelihood of multiple simultaneous node failures increases. If the simultaneous failures affect 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 checkpoint/restart activities [30]. Thus, even with such state-of-the-art checkpoint/restart techniques, application efficiency may suffer at extreme scales.

Burst buffers have been proposed as in-system storage to alleviate the problems of writing to a shared PFS [19], [20]. Burst buffers are a new tier in the storage hierarchy to fill the performance gap between node-local storage and the PFS. They can absorb the bursty I/O requests from applications and thus can reduce the effective load on the PFS. System software can manage moving data between the burst buffers and the PFS asynchronously to applications and can coordinate data movement across jobs. In this paper, we consider using burst buffers to improve system re-

[1] Tokyo Institute of Technology, 2-12-1-W8-33, Ohokayama, Meguro-ku, Tokyo 152-8552 Japan
[2] Research Fellow of Japan Society for the Promotion of Science
[3] Lawrence Livermore National Laboratory, Livermore, CA 94551 USA
[4] RIKEN, 7-1-26, Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo, 650-0047 Japan

siliency. With burst buffers, an application can store checkpoints both on node-local storage and burst buffers for increased efficiency and reliability. In this paper, we explore how burst buffers can improve efficiency compared to using only node-local storage. Our key contributions include:

- A model to evaluate system resiliency given a checkpoint/restart and storage configuration
- Simulation results showing how burst buffers can improve system resiliency compared to using only node-local storage
- A quantitative examination of the trade-offs between coordinated and uncoordinated multilevel checkpoint/restart
- An analysis of which tiers in the storage hierarchy impact system reliability and efficiency
- An exploration of burst buffer configurations to discover the best for extreme scale.

In the next section, we categorize checkpoint/restart strategies and describe our targets for our study. In Section 3, we introduce an mSATA-based SSD burst buffer machine, and show preliminary results. In Section 4, we model multi-tiered hierarchical storage and checkpoint/restart strategies. In Section 5, we describe our experimental setup and in Section 6 we simulate system efficiency given checkpoint/restart strategies and storage configurations, and describe a solution for bulding reliable system/arichitecture towards extreme scale.

## 2. Checkpoint/Restart Strategies

Over the years, many checkpoint/restart strategies have been studied. These techniques can be roughly categorized into single or multi-level, synchronous or asynchronous, and coordinated or uncoordinated checkpoint/restart. We explain each checkpoint/restart strategy and their advantages and disadvantages.

### 2.1 Single vs. Multi-Level Checkpointing

The simplest approach for checkpoint/restart is to write all checkpoints to a single location, such as the PFS [33], [35]. 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 [3], [23]. Earlier failure analysis [23] showed most failures on current supercomputers affect a single compute node, which does not necessarily require writing checkpoints to the reliable, but slow PFS. 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 checkpoint/restart 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 checkpoint/restart.

### 2.2 Synchronous vs. Asynchronous Checkpointing

Checkpointing libraries can write checkpoints either synchronously or an asynchronously. Synchronous checkpointing libraries [23], [35] write checkpoints such that all processes write their own checkpoints concurrently, and are blocked until the checkpoint operation completes. In asynchronous checkpointing

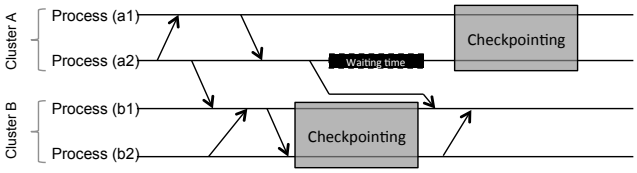


**Fig. 1** Indirect global synchronization

[29], [33], the library writes 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, which inflates an application runtime to a certain extent, can be worse than synchronous checkpointing [29]. 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.

### 2.3 Coordinated vs. Uncoordinated Checkpointing

Last, we consider whether checkpoint/restart is coordinated or uncoordinated. With coordinated checkpoint/restart, all processes globally synchronize before taking checkpoints to ensure the checkpoints are consistent and that no messages are in flight. Coordinated checkpoint/restart is applicable to a wide range of applications. However, at large scales the global synchronization can cause overhead due to propagation of system noise [14]. 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 [5] 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 [9] without message logging.

Earlier uncoordinated checkpoint techniques [11], [28] 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 log-

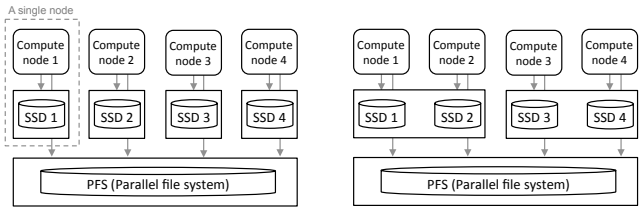


**Fig. 2** (a) Left: Flat buffer system (b) Right: Burst buffer system

ging overhead. In addition, if we apply uncoordinated checkpointing to MPI applications, *indirect global synchronization* can occur. 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) in Figure 1. 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, so we assume uncoordinated checkpointing time is same as coordinated checkpointing one in the model in Section 4.

### 2.4 Target Checkpoint/Restart Strategies

As discussed previously, multilevel and asynchronous approaches are more efficient than single and synchronous checkpoint/restart 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 checkpoint/restart. However, because we have already found that these approaches may be limited in increasing application efficiencies at extreme scale [29], we also consider storage architecture approaches.

## 3. 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 checkpoint/restart techniques, but also consider different storage architectures. 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*).

### 3.1 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 [13], [22], [24]. However this design has drawbacks: unreliable checkpoint storage and inefficient utilization of storage resources. Storing checkpoints in node-local storage is not reliable because an application can not restart its execution if a checkpoint is lost due to a failed compute node. 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. Storage devices can be underutilized with uncoordinated checkpointing and message logging. 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, and restart from a failure, the bandwidth of SSD 2 and 4 will not be utilized. Compute node 1 can write its checkpoints on the SSD of compute node 2 as well as its own SSD in order to utilize both of the SSDs on restart, but as argued earlier distributing checkpoints across multiple compute nodes is not a reliable solution.

Thus, future storage architectures require not only efficient but reliable storage designs for resilient extreme scale computing.

### 3.2 Burst Buffer System

To solve the problems in a flat buffer system, we consider a burst buffer system [21]. A burst buffer is a storage space to bridge the gap in latency and bandwidth between node-local storage and the PFS, and is shared by a subset of compute nodes. Although additional nodes are required, 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 buffer. A burst buffer system provides more efficient utilization of storage resources for partial restart of uncoordinated checkpointing 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 unlike a flat buffer system. This capability accelerates the partial restart of uncoordinated checkpoint/restart.

**Table 1** Node specification

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

To explore the bandwidth we can achieve with only commodity devices, we developed an mSATA-based SSD test system. The detailed specification is shown in Table 1. The theoretical peak of sequential read and write throughput 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 two the RAID cards via PCE-express(x8) 3.0. The theoretical peak performance of this configuration is 8 GB/sec for read and 4.16 GB/sec for write in total. Our preliminary results showed that actual read bandwidth is 7.7 GB/sec (96% of peak) and write bandwidth is 3.8 GB/sec (91% of peak) [32] . By adding two more RAID cards, and connecting via high-speed interconnects, we expect to be able to build a burst buffer machine using only commodity devices with 16 GB/sec of read, and 8.32 GB/sec of write throughput.

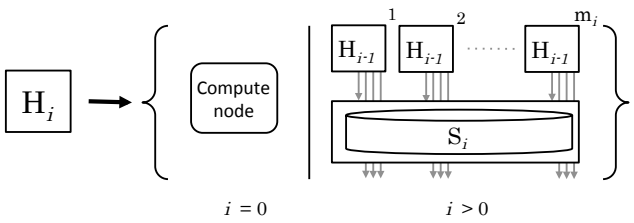


**Fig. 3** Recursive structured storage model

For efficiency exploration, we use the read and write throughput of our test system.

## 4. Modeling

As described in Sections 2 and 3, each checkpoint strategy and storage architecture have advantages and disadvantages. To discover which checkpoint strategy is best for given a storage architecture, we developed a model of the checkpoint strategies and storage architectures.

### 4.1 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 3 shows the recursive structured storage model based on a *context-free grammar*. A tier $i$ hierarchical entity, $H_i$, has a 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 $\{m_1, m_2, \ldots, m_N\}$ in an $N$-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 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 2** 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 isolate storage from compute nodes, which means the model does not distinguish between node-local storage and network-attached storage. Instead, we differentiate the storage levels using performance parameters. Table 2 shows a list of the performance parameters. We consider only sequential I/O because typically the I/O pattern of checkpoint/restart is sequential. Note that $r_i$ and $w_i$ are not the throughput of the storage but the throughput between compute nodes and the storage location. For example, if tier $i$ storage has $r$, $w$ MB/sec of throughput, but is connected via slow network, $l < r$, $w$, then the parameters become $r_i = w_i = l$ MB/sec. Using these performance parameters, we estimate checkpoint/restart time.

### 4.2 Modeling of Checkpoint/Restart 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 [29]. For simplicity, if multiple compute nodes concurrently access a single storage location, we assume the read/write throughput per node scales down according to the number of concurrently accesses. The model relies on an existing multilevel asynchronous checkpoint/restart model [29], so we also include that model's assumptions.

Checkpoint overhead ($O_i$) and restart overhead ($R_i$) are the increased execution time of an application because of checkpointing and restarting, respectively. Checkpoint latency ($L_i$) is the time to complete a checkpoint. If a checkpoint strategy conducts erasure encoding, such as XOR, the checkpoint overhead and latency also include the encoding time. Checkpoint overhead and latency are to clarify the differences between synchronous and asynchronous checkpointing. During synchronous checkpointing, so checkpoint overhead and latency is equal, i.e., $O_i = L_i$, because each process is blocked until the checkpointing is completed. With asynchronous checkpointing, checkpoint overhead can be generally reduced because asynchronous checkpointing incurs only initialization overhead, so checkpoint overhead is equal or smaller than checkpoint latency, i.e., $O_i < L_i$.

First, 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 $E_i = D \cdot e_i$ where $D$ is the checkpoint size per compute node, and $e_i$ is encoding throughput. The actual checkpointing time ($C_i$), i.e., sequencial write time, can be simply 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., $\prod_{i=1}^{N} m_i$. With uncoordinated checkpointing, we assume the checkpointing time is identical to coordinated checkpointing time because of indirect global synchronization as described in Section 2.3. Because $\prod_{k=i+1}^{N} m_k$ is the number of storage locations $S_i$, $\left\lceil \frac{M}{\prod_{k=i+1}^{N} m_k} \right\rceil$ represents the max number of compute nodes per storage location $S_i$.

When restarting with uncoordinated checkpointing, the restart overhead is different than that of coordinated checkpointing. We model the restart overhead ($R_i$), i.e., sequencial 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

**Table 3** 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/sec, 10 GB/sec} | {16 GB/sec, 10 GB/sec} |
| $\{w_1, w_2\}$ | {260 MB/sec, 10 GB/sec} | {8.32 GB/sec, 10 GB/sec} |
| $\{e_1, e_2\}$ | {400 MB/sec, N/A} | |
| $D$ | 5 GB | |
| $\{F_1, F_2\}$ | $\{2.13 \times 10^{-6},\ 4.27 \times 10^{-7}\}$ | $\{2.13 \times 10^{-6},\ 1.33 \times 10^{-8}\}$ |

$M$. With uncoordinated restart, only the number compute nodes that failed will perform restart. Here, the size of the cluster of failed nodes is $K$, and we assume each compute node in a cluster is distributed across $S_{i>N}$ storage locations with a topology-aware process mapping technique.

### 4.3 Multilevel Asynchronous Checkpoint/Restart Model

Our multilevel asynchronous checkpoint/restart model [29] 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 ($T$), i.e., $f(O, L, R, F, V, T) \Rightarrow \hat{T}$. Thus, we can compute the optimal checkpoint frequency an interval by minimizing $\hat{T}$. $v_i$ 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 $\{15, 5, 1\}$.

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

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

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

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

The efficiency metric indicates the fraction of time an application spends only in computation. We use this metric to compare the checkpoint strategies. Our earlier study [24], [29] provides more details of the model.

## 5. Experimental Setup

In this section, we describe our experimental setup including configuration details for checkpoint/restart and storage, and how we determine the failure rates to use in our model.

### 5.1 Checkpoint/Restart and Storage Configuration

In this study, we evaluate multilevel checkpoint/restart on a 2-tiered storage system. Table 3 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. The burst buffer system has 34 burst buffer nodes, each of which is shared by 32 compute nodes. Our burst buffer prototype achieved almost theoretical peak performance with two RAID cards, but for this exploration, we assume that each burst buffer node has four RAID cards (read BW 16 GB/sec, write BW 8.32 GB/sec), and are connected via a high speed interconnect which does not create a bottleneck in bandwidth. For a fair comparison, we set the aggregate I/O throughput of each tier of storage to the same values for both the flat buffer and the burst buffer systems.

For uncoordinated checkpointing, we use 16 nodes for the cluster size ($K$). An earlier study [12], [28] 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. 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 1 compute node for uncoordinated restart.

### 5.2 Failure Rate Estimation

Failure rates ($F$) are based on a failure analysis study using the Scalable Checkpoint/Restart (SCR) Library [23]. 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 rates of a single compute node requiring `LOCAL` is $1.96 \times 10^{-10}$, `XOR` is $1.77 \times 10^{-9}$, and `PFS` is $3.93 \times 10^{-10}$.

In a flat buffer system, each failure rate is calculated by multiplying the failure rate by the number of compute nodes, 1088 nodes. This leads to failure rates of $2.14 \times 10^{-7}$ for `LOCAL`, $1.92 \times 10^{-6}$ for `XOR`, and $4.27 \times 10^{-7}$ for `PFS`. Actually, if a level-$i$ failure rate is lower than a level-$i + 1$ one, the optimal level $i$ checkpoint count is zero because level $i$ can recover a level $i + 1$ checkpoint, which is written more frequently than level $i$. Thus, we do not consider `LOCAL` checkpointing for the simulation. We evaluate the two level checkpoint/restart case where level 1 is `XOR`, and level 2 is `PFS`, with failure rates of $\{F_1,\ F_2\} = \{2.14 \times 10^{-7} + 1.92 \times 10^{-6},\ 4.27 \times 10^{-7}\}$.

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. In a burst buffer system, 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 checkpoint/restart 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 $6.67 \times 10^{-8}$. Meanwhile, checkpoint data is stored on fewer nodes, which decreases

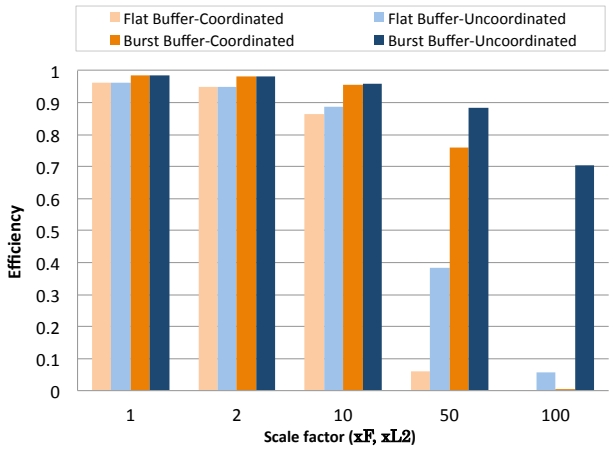


**Fig. 4** Efficiency of multilevel coordinated and uncoordinated checkpoint/restart on a flat buffer system and a burst buffer system

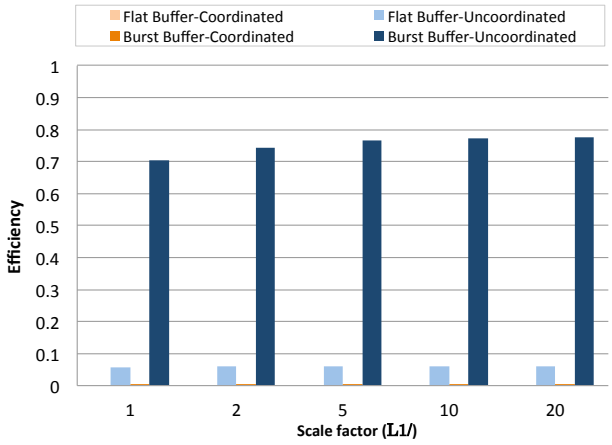


**Fig. 5** Efficiency in increasing level-1 checkpoint/restart performance

the failure rate requiring PFS for recovery. The level 2 failure is calculated as $1.33 \times 10^{-8}$. Thus, the failure rate of each level is $\{F_1,\ F_2\} = \{2.14 \times 10^{-7} + 1.92 \times 10^{-6} + 6.67 \times 10^{-8}, 1.33 \times 10^{-8}\}$ for the burst buffer system. $F_1$ increases because the burst buffer system requires additional nodes for the burst buffer.

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.

## 6. Resiliency Exploration

In this section, we evaluate the trade-offs of different checkpointing and storage configurations. In particular, we evaluate the system efficiency with increasing failure rates and checkpoint costs; the allowable message logging overhead for uncoordinated checkpointing; the effect of improving the performance at different levels of the storage hierarchy; and the optimal ratio of compute nodes to burst buffer nodes.

### 6.1 Efficiency with Increasing Failure Rates and Checkpoint 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) checkpoint costs by factors of 1, 2, 10, 50 and 100, and compare the efficiencies of multilevel coordinated and uncoordinated checkpoint/restart on a flat buffer system and on a burst buffer system. We do not change the level 1 (XOR) checkpoint cost; because it is node-local storage, its performance will scale with increasing system size.

Figure 4 shows application efficiency under increasing failure rates and checkpoint costs. When we 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 always achieves a higher efficiency than the flat buffer system. 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 checkpoint/restart 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 exploit the bandwidth of both burst buffers and the PFS, and accelerate restart time.

### 6.2 Allowable Message Logging Overhead

The efficiencies shown in Figure 4 do not include message logging overhead. We consider this factor in Table 4 which shows the message logging overhead allowed in uncoordinated checkpointing to achieve a higher efficiency than coordinated checkpointing. As in Figure 4, we increase both the failure rates and level 2 checkpointing 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.

### 6.3 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. 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 5 and 6 show efficiency with increasing performance of level 1 and 2 checkpoint/restart, using failures rates at 100 × current rates. We see that improvement of level 1 checkpoint/restart does not impact efficiency for either flat buffer or burst buffer systems. However, as shown in Figure 6, increasing the performance of the PFS does

**Table 4** 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 | ≈ 100% | 100 | 99.9% |

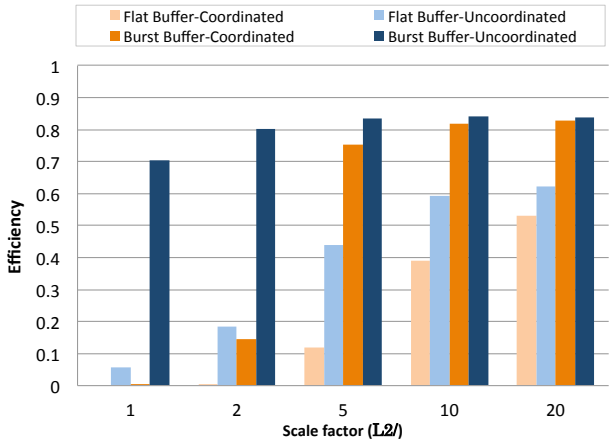


**Fig. 6** Efficiency in increasing level-2 checkpoint/restart performance

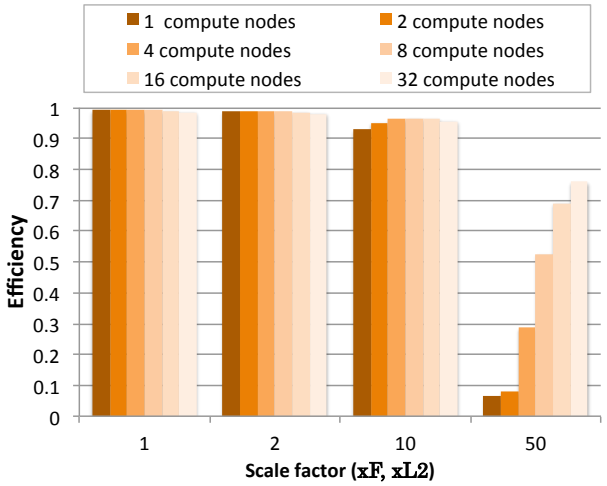


**Fig. 7** Coordinated: Efficiency in different ratios of compute nodes to a single burst buffer nodes with coordinated checkpoint/restart

impact system efficiency. We can achieve over 80% efficiency with both coordinated and uncoordinated checkpoint/restart on the burst buffer system with improved PFS performance of 10 and 20 ×. These results tell us that level 2 checkpoint/restart 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 improvement of system reliability for failures requiring level 2 checkpoint is important.

### 6.4 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 failure rate of the system and add to system cost. 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 checkpoint costs while keeping I/O throughput of a single burst buffer node constant. Figures 7 and 8 show the results with coordinated and uncoordinated checkpoint/restart. We see that the ratio is not significant up to scale factors of 10 ×. However, at a scale factor of 50 ×, 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.

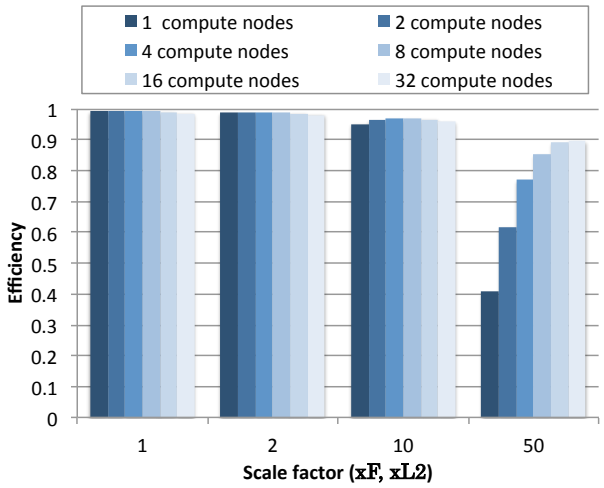


**Fig. 8** Uncoordinated: Efficiency in different ratios of compute nodes to a single burst buffer nodes with uncoordinated checkpoint/restart

## 7. Related Work

Fast checkpoint/restart is important for an application running for days and weeks at extreme scale to achieve efficient execution in the presence of failures. Multilevel checkpoint/restart [3], [23] is an approach for increasing application efficiency. Multilevel checkpoint libraries utilize multiple tiers of storage, such as node-local storage and the PFS. Uncoordinated checkpoint/restart [5], [11], [28] works effectively when coupled with multilevel checkpoint/restart. 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 [2], [6], [26], and checkpoint compression [15], [16]. However, such combined approaches are limited in their ability to improve application efficiency at extreme scale because checkpoint/restart 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. [27] 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. [1] achieved high I/O throughput by using additional nodes. As we observed, optimizing performance 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. To deal with bursty I/O requests, Liu et al. [21] proposed a storage system design that integrates SSD buffers on I/O nodes. The system achieved high aggregate I/O bandwidth. However, to the best our knowledge, our work is the first focus-

ing on a co-designed approach for increasing both I/O throughput and reliability with burst buffers at extreme scale.

Wickberg et al. [34] 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 I/O scheduler. Kannan et al. [17] also presented a data staging approach using active NVRAM (Non-volatile RAM) [18] technology. These studies focused on only I/O throughput. However, 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.

## 8. Conclusion

In this work, we explored the use of burst buffer storage for scalable checkpoint/restart on future extreme scale systems. We developed a model to explore both checkpointing strategies and storage architectures. We used our model to evaluate multilevel checkpointing on flat storage systems that are currently available on today's machines and hierarchical storage systems using burst buffers. We also modeled the performance of different checkpointing strategies, specifically coordinated checkpointing where all processes checkpoint simultaneously, and uncoordinated checkpointing where subsets of processes coordinate a checkpoint instead of the whole job.

From our exploration, we found that burst buffers are indeed beneficial for checkpoint/restart on future systems, increasing reliability and efficiency. We also found that the performance of the parallel file system has a good deal of 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 the parallel file system. 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.

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

### References

[1] Abbasi, H., Wolf, M., Eisenhauer, G., Klasky, S., Schwan, K. and Zheng, F.: DataStager: Scalable Data Staging Services for Petascale Applications, *Proceedings of the 18th ACM international symposium on High performance distributed computing*, HPDC '09, New York, NY, USA, ACM, pp. 39–48 (online), DOI: 10.1145/1551609.1551618 (2009).

[2] Agarwal, S., Garg, R., Gupta, M. S. and Moreira, J. E.: Adaptive Incremental Checkpointing for Massively Parallel Systems, *Proceedings of the 18th annual international conference on Supercomputing*, ICS '04, New York, NY, USA, ACM, pp. 277–286 (online), DOI: 10.1145/1006209.1006248 (2004).

[3] Bautista-Gomez, L., Komatitsch, D., Maruyama, N., Tsuboi, S., Cappello, F. and Matsuoka, S.: FTI: high performance Fault Tolerance Interface for hybrid systems, *Proceedings of the 2011 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, Seattle, WS, USA (2011).

[4] Berger, R. L., Still, C. H., Williams, E. A. and Langdon, A. B.: 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).

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

[6] Bronevetsky, G., Marques, D., Pingali, K. and Rugina, R.: Languages and Compilers for Parallel Computing, Springer-Verlag, Berlin, Heidelberg, chapter Compiler-Enhanced Incremental Checkpointing, pp. 1–15 (online), DOI: 10.1007/978-3-540-85261-2_1 (2008).

[7] Daly, J. et al.: Inter-Agency Workshop on HPC Resilience at Extreme Scale (2012).

[8] Dongarra, J., Beckman, P., Moore, T., Aerts, P., Aloisio, G., Andre, J.-C., Barkai, D., Berthou, J.-Y., Boku, T., Braunschweig, B., Cappello, F., Chapman, B., Chi, X., Choudhary, A., Dosanjh, S., Dunning, T., Fiore, S., Geist, A., Gropp, B., Harrison, R., Hereld, M., Heroux, M., Hoisie, A., Hotta, K., Jin, Z., Ishikawa, Y., Johnson, F., Kale, S., Kenway, R., Keyes, D., Kramer, B., Labarta, J., Lichnewsky, A., Lippert, T., Lucas, B., Maccabe, B., Matsuoka, S., Messina, P., Michielse, P., Mohr, B., Mueller, M. S., Nagel, W. E., Nakashima, H., Papka, M. E., Reed, D., Sato, M., Seidel, E., Shalf, J., Skinner, D., Snir, M., Sterling, T., Stevens, R., Streitz, F., Sugar, B., Sumimoto, S., Tang, W., Taylor, J., Thakur, R., Trefethen, A., Valero, M., Van Der Steen, A., Vetter, J., Williams, P., Wisniewski, R. and Yelick, K.: The International Exascale Software Project roadmap, *Int. J. High Perform. Comput. Appl.*, Vol. 25, No. 1, pp. 3–60 (online), DOI: 10.1177/1094342010391989 (2011).

[9] Elnozahy, E. N. M., Alvisi, L., Wang, Y.-M. and Johnson, D. B.: A Survey of Rollback-Recovery Protocols in Message-Passing Systems, *ACM Computing Surveys*, Vol. 34, No. 3, pp. 375–408 (2002).

[10] Geist, A. and Engelmann, C.: Development of Naturally Fault Tolerant Algorithms for Computing on 100,000 Processors (2002).

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

[12] Guermouche, A., Ropars, T., Snir, M. and Cappello, F.: HydEE: Failure Containment without Event Logging for Large Scale Send-Deterministic MPI Applications, *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, IEEE, pp. 1216–1227 (online), DOI: 10.1109/ipdps.2012.111 (2012).

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

[14] Hoefler, T., Schneider, T. and Lumsdaine, A.: Characterizing the Influence of System Noise on Large-Scale Applications by Simulation, *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, Washington, DC, USA, IEEE Computer Society, pp. 1–11 (online), DOI: 10.1109/SC.2010.12 (2010).

[15] Ibtesham, D., Arnold, D., Ferreira, K. B. and Bridges, P. G.: On the Viability of Checkpoint Compression for Extreme Scale Fault Tolerance, *Proceedings of the 2011 international conference on Parallel Processing - Volume 2*, Euro-Par'11, Berlin, Heidelberg, Springer-Verlag, pp. 302–311 (online), DOI: 10.1007/978-3-642-29740-3_34 (2012).

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

[17] Kannan, S., Gavrilovska, A., Schwan, K., Milojicic, D. and Talwar, V.: Using Active NVRAM for I/O Staging, *Proceedings of the 2nd international workshop on Petascal data analytics: challenges and opportunities*, PDAC '11, New York, NY, USA, ACM, pp. 15–22 (online), DOI: 10.1145/2110205.2110209 (2011).

[18] Kannan, S., Milojicic, D., Talwar, V., Gavrilovska, A., Schwan, K. and Abbasi, H.: Using Active NVRAM for Cloud I/O, *Open Cirrus Summit*, Vol. 0, pp. 32–36 (online), DOI: http://doi.ieeecomputersociety.org/10.1109/OCS.2011.12 (2011).

[19] Kimpe, D., Mohror, K., Moody, A., Van Essen, B., Gokhale, M., Ross,

R. and de Supinski, B. R.: Integrated In-System Storage Architecture for High Performance Computing, *Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '12 (2012).

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

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

[22] Matsuoka, S., Aoki, T., Endo, T., Sato, H., Takizawa, S., Nomura, A. and Sato, K.: *TSUBAME2.0: The First Petascale Supercomputer in Japan and the Greatest Production in the World*, Vol. 1, chapter 20, pp. 525–556 (online), available from ⟨http://www.crcnetbase.com/doi/book/10.1201/b14677⟩, Chapman & Hall/CRC Computational Science (2013).

[23] Moody, A., Bronevetsky, G., Mohror, K. and de Supinski, B. R.: Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System, *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, Washington, DC, USA, IEEE Computer Society, pp. 1–11 (online), DOI: 10.1109/sc.2010.18 (2010).

[24] Moody, A., Bronevetsky, G., Mohror, K. and de Supinski, B. R.: Detailed Modeling, Design, and Evaluation of a Scalable Multi-level Checkpointing System, Technical report, Lawrence Livermore National Laboratory (2010).

[25] Patrick, C. M., Son, S. and Kandemir, M.: Comparative Evaluation of Overlap Strategies with Study of I/O Overlap in MPI-IO, *SIGOPS Oper. Syst. Rev.*, Vol. 42, pp. 43–49 (online), DOI: 10.1145/1453775.1453784 (2008).

[26] Plank, J. S., Beck, M., Kingsley, G. and Li, K.: Libckpt: Transparent Checkpointing under Unix, Technical report, Knoxville, TN, USA (1994).

[27] Rajachandrasekar, R., Ouyang, X., Besseron, X., Meshram, V. and Panda, D. K.: Can Checkpoint/Restart Mechanisms Benefit from Hierarchical Data Staging?, *Proceedings of the 2011 international conference on Parallel Processing - Volume 2*, Euro-Par'11, Berlin, Heidelberg, Springer-Verlag, pp. 312–321 (online), DOI: 10.1007/978-3-642-29740-3_35 (2012).

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

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

[30] Sato, K., Moody, A., Mohror, K., Gamblin, T., de Supinski, B. R., Maruyama, N. and Matsuoka, S.: Design and Modeling of a Non-Blocking Checkpoint System, *ATIP - A*CRC Workshop on Accelerator Technologies in High Performance Computing* (2012).

[31] Schroeder, B. and Gibson, G. A.: Understanding Failures in Petascale Computers, *Journal of Physics: Conference Series*, Vol. 78, No. 1, pp. 012022+ (online), DOI: 10.1088/1742-6596/78/1/012022 (2007).

[32] Shirahata, K., Sato, H. and Matsuoka, S.: Preliminary I/O performance Evaluation on GPU Accelerator and External Memory, *IPSJ SIG Technical Reports 2013-HPC-141* (2013).

[33] Vaidya, N. H.: On Checkpoint Latency, Technical report, College Station, TX, USA (1995).

[34] Wickberg, T. and Carothers, C.: The RAMDISK storage accelerator: a method of accelerating I/O performance on HPC systems using RAMDISKs, *Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '12, New York, NY, USA, ACM, pp. 5:1–5:8 (online), DOI: 10.1145/2318916.2318922 (2012).

[35] Young, J. W.: A First Order Approximation to the Optimum Checkpoint Interval, *Commun. ACM*, Vol. 17, pp. 530–531 (online), DOI: 10.1145/361147.361115 (1974).