On the Non-Suitability of Non-Volatility

John Bent[*†] Brad Settlemyer[†‡] Nathan DeBardeleben[†‡]
Sorin Faibish[*] Uday Gupta[*] Dennis Ting[*] Percy Tzelnic[*]

Abstract

For many emerging and existing architectures, NAND flash is the storage media used to fill the cost-performance gap between DRAM and spinning disk. However, while NAND flash is the best of the available options, for many workloads its specific design choices and trade-offs are not wholly suitable. One such workload is long-running scientific applications which use checkpoint-restart for failure recovery. For these workloads, HPC data centers are deploying NAND flash as a storage acceleration tier, commonly called burst buffers, to provide high levels of write bandwidth for checkpoint storage. In this paper, we compare the costs of adding reliability to such a layer versus the benefits of not doing so. We find that, even though NAND flash is non-volatile, HPC burst buffers should not be reliable when the performance overhead of adding reliability is greater than 2%.

1 Introduction

Scientific applications running on high-performance computing (HPC) systems typically rely on the storage system to ensure application progress. Because scientific simulations are tightly-coupled across hundreds of thousands of compute cores, and derive their next state from their current state, checkpoints are the most popular technique for ensuring that applications make progress in the face of frequent machine failures. As the memory footprint of high-end machines has increased into the multi-petabyte range, it has become necessary to construct a new storage tier to meet the *bandwidth requirements* of application checkpointing [1, 10, 12]. The existing storage tier (i.e. primarily disk-based parallel file systems) remains but now serves only the *capacity requirements* of HPC workloads.

This new HPC acceleration tier is typically called a *burst buffer*, and sacrifices capacity per dollar to achieve much higher levels of bandwidth per dollar. In this way, these burst buffers are similar to the storage acceleration tiers being deployed within enterprise and cloud facilities as web/database caches, data ingest layers, metadata indices, etc. At present, these storage acceleration tiers are comprised of NAND flash memory, a non-volatile storage technology characterized by fast reads and slower writes (though performance is typically faster than the same access pattern on spinning disks) [11]. Due to the non-volatile storage property of the flash memory within the storage acceleration tier, HPC burst buffers appear to fit as a part of the storage hierarchy as opposed to a part of the memory hierarchy. Further, it seems intuitive that the acceleration tier media should be managed similarly to the way we manage the other layers within the storage hierarchy (parallel file systems and archival storage). Data replication, software-based RAID, and erasure coding in conjunction with PAXOS [8] or heartbeats all seem to be natural design points for implementing an acceleration tier that adequately protects data.

Compute Nodes	100K	–
Compute Cores	1B	–
Checkpoint Data	1EB / day	–
CN:BB Ratio	100:1	–
Burst Time	5 min	CK_{Unr}
Drain Time	60 min	D
Compute Phase	60 min	CO
MTBI	24 hours	N

Table 1: **Exascale Projections.** *Values taken from the Department of Energy Exascale Initative Steering Committee's roadmap to exascale[4] and the variables we use to represent some of them.*

However, this design intuition is incorrect. Burst buffers, although vital to ensuring application progress within the HPC data center, should not be architected to protect data, or even be designed to achieve high degrees of data reliability. Software and firmware techniques that layer distributed fault tolerance schemes on top of the storage acceleration tier are actively harmful. The use of a reliable storage medium, NAND flash, within the burst buffer is strictly an economic decision, and if a volatile memory with lower costs and similar performance characteristics existed, it would be a straightforward optimization to use it within this acceleration tier.

In this paper we explore the mathematical basis for why this storage acceleration tier should not be reliable. For completeness, we also examine the conditions under which a storage acceleration tier should be reliable. Finally, we briefly explore the hypothetical characteristics of an ideal media for storage acceleration tiers, with the hope of influencing future high-performance storage technologies.

2 Acceleration Tier Reliability

IOD, a DOE-sponsored project that researched burst buffers (BB) for exascale computers[7], designed a prototype that did

[*]EMC OCTO; first.last@emc.com
[†]LANL HPC; {bws,ndebard@lanl.gov}
[‡]A portion of this work was performed at the Ultrascale Systems Research Center (USRC) at Los Alamos National Laboratory, supported by the U.S. Department of Energy contract DE-FC02-06ER25750. The publication has been assigned the LANL identifier LA-UR-15-21931

not provide data reliability within the BB tier. Instead the project relied on a disk-based reliable capacity tier for recovery. This decision was based on a cost-benefit analysis which we will now describe in more detail.

For this analysis we rely on several assumptions. First, we assume that the exascale projections in Table 1 are at least as accurate now as they were when they were first published. We also assume that checkpoint data is usually migrated to the capacity storage system to recover acceleration tier capacity (typically only slightly larger than aggregate main memory). Note, however, that migrating all checkpoint data is not required; instead, applications require only that a moderate number of checkpoints exist after workflow completion for validation, verification, analysis and visualization, and uncertainty quantification. We further assume that sufficient read bandwidth exists in the acceleration tier to saturate the capacity tier. More importantly, we also assume that the read operations do not interfere with the write performance of the acceleration tier as it must simultaneously accept new data from the computation while migrating previously saved data to the capacity tier. We assume that read rates and write rates are the same for both the capacity tier and the BB tiers. Finally, we assume that BB interruptions do not impact the application (although this was true for the implementation of the IOD prototype, this property is not always trivial to achieve in practice).

Figure 1 shows state transition diagrams for a typical HPC workflow in which an application alternates between N compute and checkpoint phases with interspersed recovery phases as represented by CO, CK, and R, respectively. The time spent in the CK and R states differs depending on whether the BB tier is reliable. With reliable BB's, recovery can always use the last checkpoint in the BB and therefore the transition from the recovery state always returns to the most recent compute phase. With unreliable BB's, the last checkpoint stored in the BB might be lost in which case the recovery will need to use the last checkpoint stored in the capacity tier thereby returning it to the previous compute phase. This difference appears as the transitions labeled with f_{bb} in Figure 1b.

We now define the variables and equations we use to compare these workflows. Extrapolations from Table 1 show that there are 24 compute phases (represented as N below) and one recovery phase in each day. Accordingly, the specific workflow equations to perform one day's worth of computation for unreliable and reliable BBs respectively are:

$$W_{\text{Unr}} = N * (CO + CK_{\text{Unr}}) + R_{\text{Unr}} \quad (1)$$

$$W_{\text{Rel}} = N * (CO + CK_{\text{Rel}}) + R_{\text{Rel}} \quad (2)$$

where CK_{Unr} and CK_{Rel} are the respective checkpoint costs for unreliable and reliable BBs and R_{Unr} and R_{Rel} are their respective recovery costs.

The first three variables in Equation 1 are defined in Table 1; the fourth, R_{Unr}, is more complicated and we therefore temporarily delay presenting its definition.

Turning to Equation 2, the checkpoint cost with a reliable BB, CK_{Rel}, is closely related to the checkpoint cost with an unreliable BB, CK_{Unr}. For an unreliable BB, the checkpoint data can be striped across all of the available BB storage. Conversely, for a reliable BB, the checkpoint data can be striped across only a subset of the BB storage since the remaining BB storage must be used for parity. We introduce the variable P to represent the multiplier for parity space overhead. Typical values of P may range from 3 (for HDFS style triplication) to 1.2 (for efficient erasure coding). Note that P represents both the capacity overhead as well as the performance overhead since checkpointing is a streaming workload which uses the entire BB tier.

$$CK_{\text{Rel}} = CK_{\text{Unr}} * P \quad (3)$$

Using our assumption that read and write rates are the same, the recovery cost for a reliable BB is the same as its checkpoint cost.

$$R_{\text{Rel}} = CK_{\text{Rel}} \quad (4)$$

We now present the previously deferred definition of R_{Unr} from Equation 1. As shown in Figure 1b, recovery using an unreliable BB will sometimes recover from the BB and sometimes recover from the capacity tier. This is because a recovery from the capacity tier will only be necessary in the event of a simultaneous interruption of both a compute node as well as a BB node. For example, if there is an interruption only on a compute node, then the application can resume using the last checkpoint stored on the BB.

Conversely, if there is a interruption only on the BB tier, then the application does not need to restart at all as it will merely save its next checkpoint to the remaining BB nodes. We define f' as the percentage of interruptions in which both a compute node and a BB node have simultaneously failed. Therefore, the expected workflow on unreliable BBs includes a fraction, f', of recoveries from the capacity tier with the remaining recoveries using the BB tier.

$$R_{\text{Unr}} = (1 - f') * R_{\text{BB}} + f' * R_{\text{CT}} \quad (5)$$

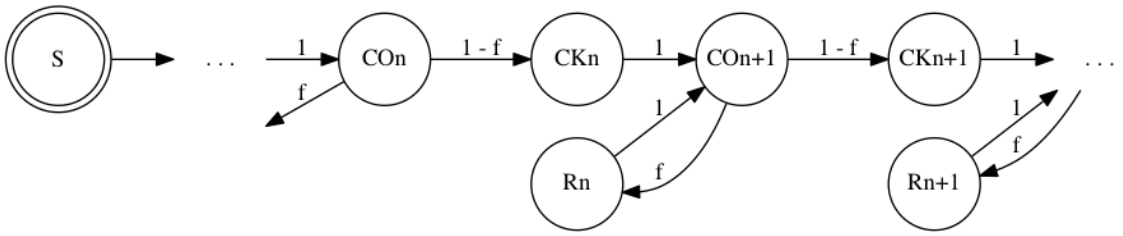
where R_{BB} and R_{CT} are the respective recovery costs from the BB tier and the capacity tier for an unreliable BB.

A more formal description of the time spent recovering from an unreliable BB must include the possibility of the BB tier failing (with probability f_{bb}) during consecutive checkpoint phases, thus requiring the application to recompute multiple compute phases.

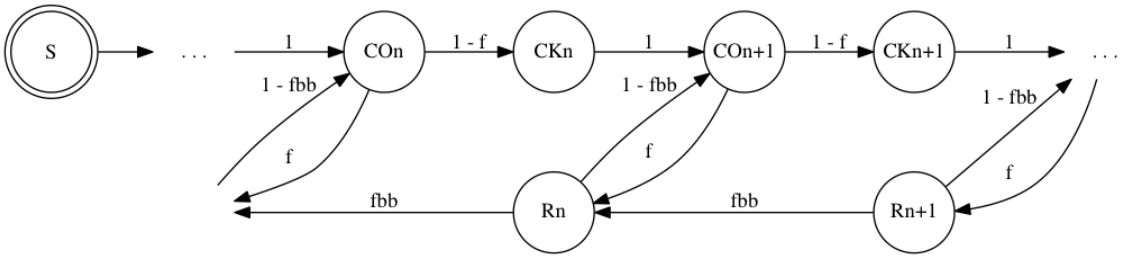
$$R_{\text{Unr}} = (1 - f') * R_{\text{BB}} + f' * (R_{\text{CT}} + (CO + CK_{\text{Unr}}) \sum_{k=1}^{n} {f_{bb}}^{k} (1 - f_{bb}) k) \quad (6)$$

However, geometrically distributed failures are an anomaly that HPC operators will work to prevent, and further the probability of consecutive failures is very small when f' is small. Therefore, for clarity, we use Equation 5 rather than Equation 6 for the remainder of our discussion.

To determine f', we first assume that the interrupt rate, f, of the compute nodes is the same as the interrupt rate of the BB nodes. Referring back to Table 1, we see that the expected ratio of compute nodes to BB nodes is 100:1, therefore we can



(a) Checkpoint-Restart Workflow with a Reliable Burst Buffer



(b) Checkpoint-Restart Workflow with an Unreliable Burst Buffer

Figure 1: **Typical HPC Workflow.** *These Markov diagrams show an HPC checkpoint-restart workflow for both a reliable and an unreliable burst buffer. CO_n represents a compute phase, CK_n represents writing a checkpoint, and recovery/restart is represented with R_n. f is the failure probability for a compute node and f_{bb} is the probability of a BB node failure. For clarity, we do not show a failure possibility during the checkpoint phase; note this does not affect our analysis.*

extrapolate that there will be a simultaneous interrupt on both a compute node and a BB node at a rate of no more than one percent of the interrupt rate f. We therefore set f' to be 0.01.

Assuming once again equivalent read and write rates,

$$R_{\text{BB}} = CK_{\text{Unr}} \tag{7}$$

R_{CT}, the time to recover from the capacity tier, is:

$$R_{\text{CT}} = F_{\text{CT}} + CO + CK_{\text{Unr}} \tag{8}$$

where F_{CT} is the time to read (fetch) the checkpoint data from the capacity tier and CO and CK_{Unr} represent the time to repeat the lost compute and checkpoint phases (i.e. the application restarts from checkpoint CK_n even though it had previously saved CK_{n+1} to the BB). Assuming equivalent read and write rates one last time, and using D from Table 1:

$$F_{\text{CT}} = D \tag{9}$$

At this point, we see that the general trade-off between reliable and unreliable BB is that the unreliable BB offers a faster checkpoint whereas the reliable BB offers a faster recovery.

Having defined all of our variables, we are now able compare the expected workflows for reliable and unreliable BB architectures. Substituting our defined variables into Equation 2 for for a reliable BB yields

$$W_{\text{Rel}} = 24 * (60 + P * 5) + P * 5 \tag{10}$$

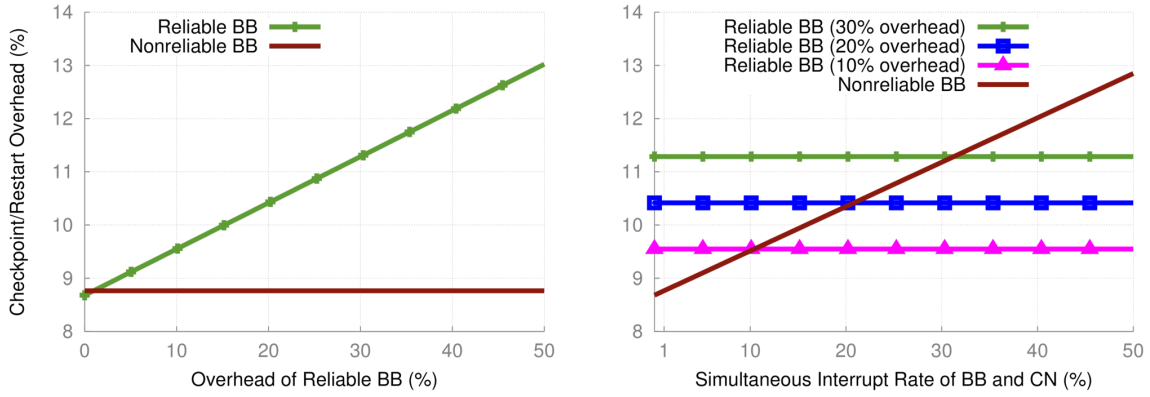
The resulting substitution simplifies to $125P + 1440$ minutes of total run-time for a compute workload of one day (1440 minutes). Whereas the workflow for an unreliable BB architecture is:

$$W_{\text{Unr}} = 24 * (60 + 5) + (0.99 * 5 + 0.01 * (60 + 60 + 5)) \tag{11}$$

The resulting total workflow time of a day's worth of computation for an unreliable BB is 1566.2 minutes, indicating a overhead of less that 9%. Therefore, the total time to completion of a workflow using an unreliable BB is actually faster than using a reliable BB that requires 20% parity overhead to construct an erasure code. In fact, as shown in Figure 2a, a reliable BB only outperforms a unreliable BB when its parity overhead is less than two percent (an unrealistically low value to achieve real reliability). Amdahl's law is the guiding principle here: since the capacity tier recovery is sufficiently infrequent, optimizing it provides little value.

Given the additional software and hardware complexity required to implement a reliable BB and given further that an unreliable BB will outperform a reliable one, the decision to provide an unreliable BB as part of the IOD project is well supported. Only in cases where the performance overhead is negligible, does it make sense to engineer greater reliability into the HPC acceleration tier. In Section 3 we will examine the circumstances in which a storage acceleration tier must be made reliable.

Note that this evaluation does not consider *semi-reliable* BB architectures, such as one in which the parity is added asynchronously after the checkpoint completes, or an architecture in which only the metadata is reliable to enable recovery of partial data. Since the need for reliability in the BB is only once every 100 days, this complexity seems unnecessary. We also do not consider novel programming models and job scheduling algorithms that are able to immediately restart subsets of processes following a failure. Since we are not aware of any such algorithms existing in practice, we cannot yet evaluate their actual workflow costs.



(a) Variable Reliability Overhead

(b) Variable Interrupt Rates

Figure 2: **Comparison of Reliable and Nonreliable Burst Buffers.** *These graphs show the equations developed within this paper to compare the checkpoint/restart overhead imposed on applications when using a reliable or an unreliable burst buffer architectures. Graph 2a shows that a reliable burst buffer outperforms an unreliable burst buffer only when the performance costs of parity overhead are less than 2%. This graph uses the projected exascale interrupt rates to fix the frequency of correlated compute node and burst buffer interruptions at one percent. Graph 2b explores the impact of further increasing the number of simulataneous failures in the unreliable burst buffer and the compute partition along the x-axis. Three reference lines are shown for reliable burst buffers with 10, 20, and 30 percent parity overheads. As long as the rate of simulataneous failures of both the burst buffer and compute partition are less than 10%, an unreliable burst buffer provides better compute throughout than even low-overhead reliable burst buffers.*

2.1 Varying the Interrupt Rate

In the previous section, we relied on several assumptions for our analysis (namely, that our assumed values were correct, durable persistence of every checkpoint is not required, bandwidths are well matched within the system, nodes are repaired or replaced during the recovery time, and that failures in the BB do not impact a running application).

We carefully use the term *interrupt* rate and not *failure* rate because the projections for MTBI shown in Table 1 assume that not every failure in the supercomputer results in an interruption for the application. The application may rely on as-yet-undeveloped resilience techniques that guard against some classes of failure. Whether these same techniques can similarly improve the interrupt rate for BB tiers is not yet known. If they cannot, the interrupt rates for BB nodes and running applications will be different thereby invalidating our previous expected value of 0.01 for f' (the rate of simultaneous interruptions in the BB tier and the running application).

As shown above, when the simultaneous interrupt rate is one percent, the unreliable BB architecture is the correct design. However, if the BB nodes are not as resistant to interrupt as the compute nodes, then the rate of simultaneous interrupt may be much higher than one percent. Since simultaneous interrupts are very expensive for an unreliable BB but do not affect a reliable BB, higher simultaneous interrupt rates will favor reliable BB designs.

Figure 2b compares the overhead of a unreliable BB architecture under varying rates of simultaneous interruptions and shows that a unreliable BB is preferable to even a very low overhead reliable BB for rates of simultaneous interruptions of less than ten percent.

These results extend earlier research, such as BAD-FS [2] and Tachyon [9], which similarly show the value of reproducing/recomputing lost data rather than protecting data.

3 Acceleration Tier Reliability

In our first analysis, we determined that a reliable burst buffer is unnecessary for the HPC computing use cases. In this section, we examine the conditions in which a storage acceleration tier must be reliable. Effectively, if either of the following conditions is present, then the storage acceleration tier must be reliable: the data to store is non-deterministic, or an application cannot coherently detect acceleration tier failures.

3.1 Non-deterministic Data

The first scenario in which a storage acceleration tier must be reliable is the case where data is *non-deterministic* (as defined in [9]), that is the data has a temporal component, such that it cannot be reconstructed later. For example, if we consider a remote sensing array that is ingesting data from a collection of security cameras, assuming the security system data is important, we cannot safely discard data during a failure within the acceleration tier because it is impossible to reconstruct the data at a later time. That is, due to the non-deterministic nature of the data, it is impossible to reconstruct this data at a later date, and so while a storage acceleration tier that compresses the video data may easily meet the performance requirements to store the data, it must also be engineered with sufficient reliability to ensure that no data is lost during an acceleration tier failure.

In the case of deterministic data that is difficult or expensive to recompute, an analysis similar to the one in Section 2

is needed to determine if the expense required to make the storage acceleration tier reliable is justified.

3.2 Incoherent Failure Detection

The second condition that mandates a reliable storage acceleration tier is when the application is unable to coherently detect a failure within the storage acceleration tier. A simple example of the failure detection problem exists for RAID controller pairs. The flash storage located on modern RAID controllers is responsible for both accelerating performance and ensuring that, in the face of a power failure that causes incoherent states across the array of disks, the acceleration tier contains sufficient metadata to determine the status of the writes successfully queued onto the controller. Thus, RAID controllers require the non-volatile property of the flash storage when recovering from an interruption.

Burst buffers experience a similar, but not identical, failure condition in the case where multiple clients attempt to asynchronously determine if the data written into the burst buffer as a distributed checkpoint is complete, and can be used for recovery. Rather than provide a reliable storage tier, the IOD prototype implemented the required synchronization to achieve coherent failure within the burst buffer [5, 6]. IOD achieves this requirement by carefully ordering and naming the interactions with the burst buffer such that an application can easily detect if a checkpoint is valid. Note that the implementation does not rely on the non-volatile nature of the underlying storage media, but it does require that a fence operation exists within the burst buffer to ensure that the most recently written data is reliably returned from the storage acceleration tier (thus constructing a coherent failure detector [3]). MPI, a message passing middleware common to the HPC community is used to create this fencing operation.

4 Conclusions

In our first analysis, we determined that a reliable burst buffer is unnecessary for the HPC computing use cases. Our analysis demonstrated that unless data protection capacity overheads are less than 2%, an unreliable burst buffer will reduce the time spent performing checkpoint storage, thus improving the overall time spent performing useful work rather than defensive I/O operations. We also examined the impact of more frequent failures within the burst buffer and observed that only in the case where greater than 10% of the job failures are unable to use the unreliable burst buffer for recovery are even highly-efficient erasure-code based reliable burst buffers preferable to an unreliable implementation.

In fact, it follows from our analysis that the only reason flash memory is used in the construction of burst buffers (or any unreliable storage acceleration tier) is that it is available at a price point that offers superior bandwidth and sufficient capacity to construct an acceleration tier at least twice as large as the largest system's main memory size. Flash memory's performance is characterized by fast reads (for both streaming workloads and random access), slightly less fast streaming writes, and much slower random writes. An ideal acceleration tier storage media could be volatile, and may have faster write performance than read performance. Further, NAND flash dissipates much more power while writing data as compared to the power dissipated while reading data, and has a limited number of write-erase cycles before the flash is no longer able to accept new data. Because burst buffers are not required to relay every checkpoint to the durable backing store, we expect the burst buffer write workload to be far greater than the read workload, which is contrary to both the energy use characteristics and wear capabilities of the media. Rather, the HPC community would receive greater benefit from a less expensive volatile media, that sacrificed read bandwidth for better write performance. Burst buffer designers would also prefer lower power writes and improved write wear characteristics that reduced the need to replace the media as it wears out due to the intensive write workload.

Finally, we examined the conditions that require a storage acceleration tier to leverage non-volatile storage: the presence of non-deterministic data or the lack of coherent failure detection. Even in the cases of non-deterministic data, and incoherent failure semantics, it is not clear that the performance and energy use properties of flash memory best suit these workloads. However, as greater performance and non-volatility are required in these scenarios, systems software designers are at least able to take advantage of the durable nature of flash-based storage.

References

[1] J. Bent, S. Faibish, J. Ahrens, G. Grider, J. Patchett, P. Tzelnic, and J. Woodring. Jitter-free co-processing on a prototype exascale storage stack. In *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, pages 1–5, April 2012.

[2] J. Bent, D. Thain, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and M. Livny. Explicit Control in a Batch-Aware Distributed File System. In *First Symposium on Networked Systems Design and Implementation (NSDI '04)*, pages 365–378, San Francisco, California, Mar. 2004.

[3] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, Mar. 1996.

[4] D. E. I. S. Committee. A decadal DOE plan for providing exascale applications and technologies for DOE mission needs. Technical report, 2009.

[5] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *J. ACM*, 34(1):77–97, Jan. 1987.

[6] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, Apr. 1985.

[7] Intel, The HDF Group, Cray, and EMC. Extreme-scale computing research and development Fast Forward storage and I/O. Technical report, 2014.

[8] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

[9] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica. Tachyon: Reliable, memory speed storage for cluster computing frameworks. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–15. ACM, 2014.

[10] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn. On the role of burst buffers in leadership-class storage systems. In *In Proceedings of the 2012 IEEE Conference on Massive Data Storage*, 2012.

[11] M. Sanvido, F. Chu, A. Kulkarni, and R. Selinger. NAND flash memory and its role in storage architectures. *Proceedings of the IEEE*, 96(11):1864–1874, Nov 2008.

[12] T. Wang, S. Oral, Y. Wang, B. Settlemyer, S. Atchley, and W. Yu. Burstmem: A high-performance burst buffer system for scientific applications. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 71–79, Oct 2014.