Orchestrating Fault Prediction with Live Migration and Checkpointing

Subhendu Behera[1], Lipeng Wan[2], Frank Mueller[1], Matthew Wolf[2], Scott Klasky[2]
{ssbehera,fmuelle}@ncsu.edu,{wanl,wolfmd,klasky}@ornl.gov
[1]North Carolina State University [2]Oak Ridge National Laboratory
Raleigh, North Carolina, USA Oak Ridge, Tennessee, USA

ABSTRACT

Checkpoint/Restart (C/R) is widely used to provide fault tolerance on High-Performance Computing (HPC) systems. However, Parallel File System (PFS) overhead and failure uncertainty cause significant application overhead. This paper develops an adaptive multi-level C/R model that incorporates a failure prediction and analysis model, which orchestrates failure prediction, checkpointing, checkpoint frequency, and proactive live migration along with the additional benefit of Burst Buffers (BB). It effectively reduces the overheads due to failures, checkpointing, and recovery. Simulation results for the Summit supercomputer yield a reduction of ≈20%-86% in application overhead due to BBs, orchestrated failure prediction, and migration. We also observe a ≈29% decrease in checkpoint writes to BBs, which can increase the longevity of the BB storage devices.

KEYWORDS

High Performance Computing, Failure Prediction, Resilience

ACM Reference Format:

Subhendu Behera, Lipeng Wan, Frank Mueller, Matthew Wolf, Scott Klasky. 2020. Orchestrating Fault Prediction with Live Migration and Checkpointing. In *Proceedings of the 29th Int'l Symp' on High-Performance Parallel & Distributed Computing (HPDC '20), June 23–26, 2020, Stockholm, Sweden.* ACM, NY, NY, USA, 5 pages. https://doi.org/10.1145/3369583.3392672

1 INTRODUCTION

Failures [13, 25] and high I/O contention [15, 17, 19] add significant overhead to application execution and become the key challenge for C/R efficiency in the era of exascale computing due to scale, heterogeneity, and high parallelism. This work aims to address these challenges in a novel manner exploiting recent technological advances.

First, we improve the efficiency of writing checkpoints by taking advantage of BBs. The PFS has long been identified as a bottleneck for highly parallel applications [16, 17, 30, 31]. Modern HPC systems address this problem by integrating fast BBs into the I/O subsystem as intermediate storage devices, which provide faster I/O performance via buffering. Our checkpoint model uses BBs to store the checkpoint data faster than writes to PFS would have been. Once the data is in BBs, it asynchronously bleeds it off to PFS, i.e., computation within the application continues while a checkpoint previously committed to a BB is slowly written to the PFS in the background so that it can later be restored on a different node should the current node fail. These fast, node-local writes to the BB reduce the time required to store checkpoints on the critical path of an application.

Second, we leverage failure prediction models to reduce the overhead caused by re-computation due to failures. In past years, there have been significant contributions in the prediction of failures on large-scale systems [6, 7, 10, 11]. We utilize their log-based failure chain characterization technique to perform rigorous failure analysis and prediction on system logs collected from real-world HPC systems to identify instances of different possible failures and distribution of lead times of such failures. Based on this analysis, the reduced failure rate in Young's formula [35] effectively reduces checkpoint frequencies. Further, our checkpoint model integrates Desh's model from [7] to predict failures with known lead times. The novelty is that it considers live migration to reduce the cost of re-computation upon immanent failures and studies its impact.

In summary, we make the following contributions:

- We propose a multi-level checkpoint model that integrates Desh [7] with its ability to predict failures in advance. Based on the predicted lead time to failure, it chooses an appropriate action to avoid or at least reduce re-computation upon failure. It further incorporates BB devices in the model to reduce the latency for storing checkpoints, but also to adaptively consider the additional latency before a checkpoint becomes globally available within the PFS, and to trade off C/R with live migration based on the predicted lead time to failure.
- We use Young's formula [35] and incorporate a rigorous failure analysis model in it, which effectively reduces the frequency of checkpoints.
- Our study shows that the utilization of BBs results in a significant reduction of checkpoint overhead for both small and large size application checkpoints.
- The comparative study also shows that our failure analysis with its prediction model allows for orchestration of C/R plus live migration model with the assistance of BBs results in a significant reduction in application overhead.

The remainder of this paper is organized as follows: Section 2 introduces the system design, develops our checkpoint model and explains how BBs and proactive live migration are utilized. Section 3 discusses results from experiments followed by related work in Section 4 and a summary of the contributions in Section 5.

HPDC '20, June 23–26, 2020, Stockholm, Sweden

2 SYSTEM DESIGN

Our Checkpoint/Restart model is devised for a Summit-like HPC system. On this HPC system, the BB devices are locally attached to compute nodes. As our model uses proactive live migration, a small set of nodes is reserved but yet not allocated to any application. Workload managers such as Slurm [34] and Flux [1] support the allocation of spare nodes. We assume that spare nodes are always available, which is given as long as the rate of failure is lower than the rate at which failed nodes can be recovered. Upon failure, a node can be taken from this reserved set to replace the failed one. Our failure predictor analyzes system logs on a per-node basis and predicts failures with their estimated lead times, i.e., the predictor daemon is placed on each compute node (e.g., in a spare core, which could be shared with other services). Since only system logs are used for analysis, predictors are independent of the application running on the compute nodes. Our model also features a multi-level adaptive checkpoint capability that comprises multiple techniques, including BB bleed-off, failure analysis and prediction, and proactive live migration, which are discussed in the following.

2.1 Checkpoint Model

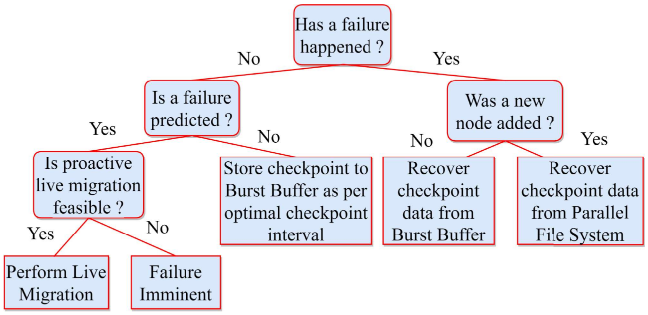
Has a failure happened ?
No
Yes
Is a failure predicted ?
Was a new node added ?
Yes
No
No
Yes
Is proactive live migration feasible ?
Store checkpoint to Burst Buffer as per optimal checkpoint interval
Recover checkpoint data from Burst Buffer
Recover checkpoint data from Parallel File System
Ycs
No
Perform Live Migration
Failure Imminent

Figure 1: Decision Tree of the Checkpoint Model

Our base checkpoint model derives the optimal checkpoint interval from a failure analysis model without considering failure prediction. With the addition of failure prediction, the model is extended to select effective and informed actions based on the decision tree of Figure 1. The criteria based on which decisions are made are as follows:

- *Has a failure happened?* At the top of the decision tree, it is assessed if any failure has occurred. Upon a failure, the application state needs to be recovered from the last checkpoint in a BB or the PFS upon availability of an existing node or addition of a new node from the reserved set, respectively. The recovery strategy is to restart from the last checkpoint. In the absence of a failure, we move to the node where the failure prediction status is checked.
- *Was a new node added?* Upon a failure, all the ranks of an application traditionally recover the checkpoint data from the PFS. This introduces significant contention that increases with checkpoint data size and the number of processes in the application. Since Summit's BBs are local, our checkpoint model mandates only a newly added node, drawn from the reserved set to replace the failed one, to recover the checkpoint data from the PFS. The remaining nodes recover the checkpoint data from local BBs. This considerably reduces failure recovery overhead.
- *Is a failure predicted?* In the absence of a predicted failure, the checkpoint model falls back to the optimal checkpoint interval to save the application state periodically. This provides tolerance against the failures that are neither predictable nor can be handled with a proactive action such as live migration. If a failure can be predicted, then the next action is indicated further down in the decision tree.
- *Is proactive live migration feasible?* If a failure is predicted, then we select an action based on the lead time to failure. With enough lead time to proactively migrate the application, the failure can be avoided by live migration. The application can continue with its computation while being migrated to a new and healthy node selected from the set of reserved nodes. The failed node is recovered and added to the set of reserved nodes for future use. If lead time is insufficient for migration, no action is taken as failure is unavoidable.

One point of discussion is the accuracy of prediction. It is assumed here that the accuracy is high [7] and lead times are reliable except for a few outliers [6]. Clearly, if prediction accuracy itself was low, only periodic checkpoints of the base model should be taken, which is not the focus of this work. The following assumptions underlie the model:

(1) Any checkpoint made to save the application state is performed by all the processes of an application.
(2) Failures happen on a single compute node and do not propagate to other nodes.
(3) No distinction is made between soft failures and hard/node failures, i.e., both are handled uniformly.
(4) The data is transferred from BBs to the PFS asynchronously, e.g., via the Spectral library on Summit [22].

2.2 BB Utilization

Concurrent access to PFS by all the compute nodes would introduce high overhead during checkpoints. Instead, BBs are utilized to provide faster access and lower contention depending on the BB architecture. On Summit, a BB device is attached to each compute node locally — in contrast to the cluster BB system on NERSC's Cori [4]. Each BB device has 1.6 TB capacity with up to 2.1 GB/sec write and 5.5 GB/sec read I/O bandwidth on a compute node [29].

There are two scenarios in which concurrent access to PFS leads to performance degradation. First, for checkpoints, all processes need to write checkpoint data concurrently to PFS. We resolve this issue by storing the checkpoint data in BBs and later bleeding it off to the PFS asynchronously, i.e., by limiting the number of nodes with BB to PFS transfer at any time. Second, for failure recovery of the application, all processes need to access PFS concurrently. However, each checkpoint is stored to the local BB first. Hence, only the new node replacing the failed one needs to recover the checkpoint data from PFS. All other nodes recover from their local BB device.

2.3 Proactive Live Migration

Proactive live migration allows us to relocate the processes on a compute node that may fail based on our failure prediction to a new and healthy node. We make several assumptions regarding our proactive live migration technique. First, during most live migration interval, the process continues to execute (until a final frozen state is transferred) [32, 33]. Second, we assume that the costs associated with the process of live migration, e.g., loading the application on one of the reserved nodes and making the changes required in the MPI runtime environment, are minimal and can thus be ignored. Finally, the total amount of data that needs to be transferred to complete live migration is upper bounded by the DRAM size on a compute node. On Summit, DRAM memory is 512 GB per node [29]. We also experimentally determined that the inter-node bandwidth on Summit is ≈12.5 GB/sec. Hence, the maximum amount of time needed to migrate an application is 41 seconds.

2.4 Failure Analysis Model

The optimal checkpoint interval in our checkpoint model further includes a rigorous analysis of failure logs. The study analyzes the system logs collected from three real-world HPC systems from Desh [7] over a period of six months. Using the Desh approach, the most common sequences of phrases in logs that may lead to a failure are considered. Our assumption in this work is that any sequence of phrases, so-called failure chains, results in an actual failure. The time difference between the first phrase and the last phrase in a chain is considered as the lead time. The reduced failure rate is included in the optimal checkpoint interval formula.

3 EVALUATION

3.1 Simulation Framework

Our simulation framework developed using SimPy [27] emulates the system-level characteristics of Summit and the execution based on traces of the real-world scientific applications in Table 1.

Table 1: HPC Workload Characteristics

Application	Number of Nodes	Checkpoint Size (GB)	Computation Time (hour)
CHIMERA	2,272	163,840	360
XGC	1,515	37,926	240
S3D	505	5,120	240
GYRO	126	50	120
POP	126	26	480
VULCAN	64	0.83	720

Previous studies [26, 28] have found that the mean time between failure (MTBF) follows a Weibull distribution. In experiments, the Weibull distribution parameters for OLCF's Titan from Wan et al. [30] are considered to generate failures during the simulation. Since the actual failure statistics of Summit are unavailable, it is assumed that the failure distribution of OLCF's Titan also applies to Summit. Upon a simulated node failure, a node is selected randomly from the set of all nodes, and the failure is simulated on that node. The number of failures for a job depends on the number of nodes an application run uses, i.e., given the MTBF rate, larger jobs (with more nodes) are subject to a high total number of failures. Simulation is repeated 1,000 times, and the measurements are averaged over those 1,000 runs. We also measure the I/O performance on Summit by considering the optimal performance on a single node and then extending it with weak scaling up to 1,000 nodes. This model is utilized in determining I/O latency for storing checkpoints.

3.2 C/R Model Evaluation

The evaluation compares three models to study the impact of the BBs and our failure prediction and analysis model assisted by proactive live migration:

- *Model A:* The base model does not incorporate any of these techniques. It also does not use BBs.
- *Model B:* This model utilizes only BBs without any of the other techniques.
- *Model C:* This model utilizes the BBs, and live migration based on failure analysis and prediction.

Figure 2 depicts the overhead of fault tolerance in percent (y-axis) normalized to the base model A (first bar) for each application (x-axis) compared to models B and C.

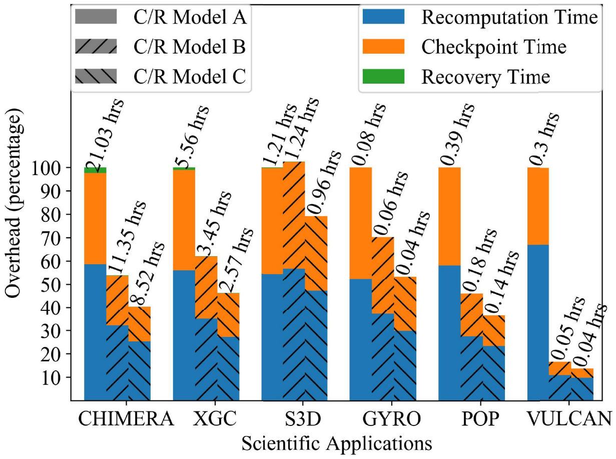

Figure 2: Reduction in Overhead with Failures from Titan

Observation 1: Application overhead is reduced by ≈20%-86% due to (1) the reduction in checkpoint time with the assistance of BBs (model B), (2) the reduction in failure rate given failure analysis and prediction (3) combined with live migration (model C).

The overhead consists of three components depicted as stacked bars. Recovery time is required to restart an application after a node failure. Since only the new node requires to access the PFS to recover the checkpoint without contention while others recover from their local BB, the recovery time is insignificant and hardly visible in the graphs for all the models except for model A. We observe a ≈60%-99% decrease in recovery time for models B and C.

Observation 2: Checkpoint overhead is initially reduced by ≈5%-29% due to the lower residual failure rate after failure prediction. While this prediction provides all the benefits to S3D (with its ≈29% reduction), all other applications further benefit from BBs and asynchronous checkpoint bleed-off, for an overall savings of ≈30%-82% depending on the application.

The checkpoint time is the duration during which application execution is suspended while storing the checkpoints on permanent storage. In model A, the PFS is the permanent storage. BBs assist other models. We observe that due to higher latency to checkpoint the PFS in model A, all applications except for S3D spend more time in storing the checkpoints without BBs. For S3D and its checkpoint size, the PFS I/O bandwidth is almost equal to the I/O bandwidth of BBs as per our I/O performance model. For smaller applications such as GYRO, POP, and VULCAN, the latency to write checkpoints to PFS is higher and results in more checkpoint overhead. Also, for larger applications such as CHIMERA and XGC, the higher latency for PFS writes is due to larger PFS contention.

Observation 3: While re-computation time dominates the base model, it can be reduced by ≈51%-56% for applications with large checkpoint sizes and ≈13-85% for those with smaller checkpoints for model C.

The re-computation time is the duration spent by an application to re-compute the portion of execution to reach the point where execution was interrupted when the node failed. It contributes the largest fraction of overhead for all the applications in the base model. We observe that lack of BBs in the base model A results in a larger checkpoint interval causing more losses due to failures. With BBs, the shorter checkpoint interval provides better fault tolerance from non-predicted failures. This reduces the re-computation overhead by ≈29%-83% for all applications except for S3D as its checkpoint interval is unaffected by BBs.

For CHIMERA and XGC, with larger checkpoints, along with S3D, the additional reduction in re-computation time is ≈13% with live migration. For the remaining applications, the additional decrease is between 2% and 14%. This suggests that live migration may not have a significant impact on smaller size applications.

Observation 4: The reduction in checkpoint time further reduces the amount of checkpoint data written to BBs in an application by ≈ 29%, which increases the lifetime of BBs as they are subject to wear-out.

Figure 3 depicts the aggregate amount of daily write traffic to the BB (y-axis) normalized to model B across all applications (x-axis) for the three models. Since model A excludes BBs, it is not under consideration. It indicated a ≈29% reduction in writes for model C. On Summit, each BB device has a daily write limit of 8 TB given its designation to last for 5 years. Model D effectively increases the longevity of the BBs by ≈41% assuming uniform daily writes across all devices and assuming that BBs are used for checkpointing, only.

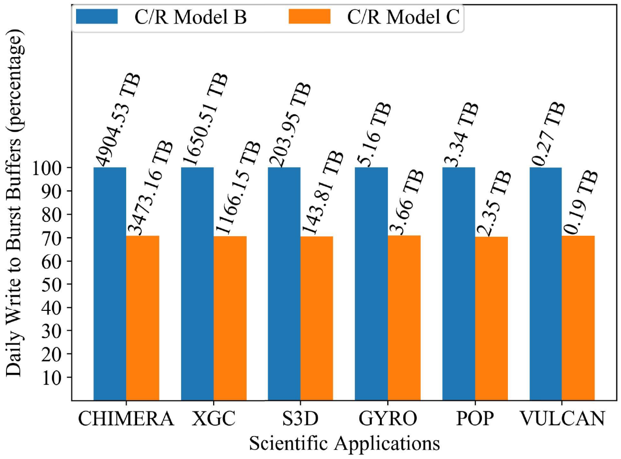

Figure 3: Reduction in Daily Writes to Burst Buffers

4 RELATED WORK

A number of extensive and significant contributions have been made to address fault resilience with C/R techniques over the past years. C/R solutions [3, 8, 20, 23] focus on reducing checkpoint overhead by using multiple storage devices with varying latency to store checkpoints. These works provide a foundation for our work but lack the integration of failure prediction into C/R models.

Other studies [2, 9, 17, 21, 24, 30] have investigated the use of BBs to quantify BB capacity requirement, measure their impact and assess how they can be used in conjunction with the PFS. Our solution, a two-level C/R model that stores its checkpoints efficiently, goes beyond these earlier approaches in its efficient recovery strategy upon failures to mandate only the replacement node to recover checkpoint from the PFS in orchestration with failure prediction, which reduces restart time considerably.

Several failure-aware C/R mechanisms [5, 12, 14, 18, 28, 32] have been devised. Wang et al. [32] perform live migration, Bouguerra et al. [5] use proactive checkpoints with failure prediction. Tiwari et al. [28] checkpoint using a temporal distribution of failures. Our C/R model also takes a similar approach to update the optimal checkpoint rate at a fixed point during C/R handling. However, our C/R model differs significantly from the lazy checkpointing solution based on the prediction techniques underlying our model, which are unique.

5 CONCLUSION

This work contributes a multi-level C/R model instantiated to resemble a Summit-like large-scale HPC system. (1) It integrates failure analysis (by analyzing system logs from three real-world HPC systems) and a prediction model, which predicts failure types and lead time, that are subsequently used in Young's formula. (2) It issues live migration of an application from a faulty node to a healthy, reserved node in face of failures. (3) It utilizes BBs to reduce PFS contention during application execution through asynchronous bleed off and for failure recovery by mandating only the replacement node to recover from a failure. The failure analysis model indicates that ≈ 44% of failures can be avoided using live migration and yields a ≈ 29% reduction in checkpoint time, which increases the lifetime of BBs. Multilevel asynchronous checkpoints supported by BBs and proactive live migration result in ≈20%-86% reduced application overhead.

Past work has studied failure/reliability-awareness to improve application efficiency. However, our C/R model's suitability for contemporary and future large-scale HPC systems, its applicability to wide range of applications, and its fault tolerance using advanced failure analysis and prediction model along with live migration are unprecedented, which gives it a significant advantage over prior solutions.

ACKNOWLEDGMENT

This research was supported in part by NSF grants 1525609, 1813004, and an appointment to the Oak Ridge National Laboratory ASTRO Program, sponsored by the U.S. Department of Energy and administered by the Oak Ridge Institute for Science and Education. This research was also supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

REFERENCES

[1] Dong Ahn, Ned Bass, Albert Chu, Jim Garlick, Mark Grondona, Stephen Herbein, Joseph Koning, Patki Tapasya, Thomas Scogland, Becky Springmeyer, and Michela Taufer. 2018. Flux: Overcoming Scheduling Challenges for Exascale Workflows. 10–19. https://doi.org/10.1109/WORKS.2018.00007

[2] Leonardo Bautista-Gomez, Seiji Tsuboi, Dimitri Komatitsch, Franck Cappello, Naoya Maruyama, and Satoshi Matsuoka. 2011. FTI: High Performance Fault Tolerance Interface for Hybrid Systems *(SC '11)*. Association for Computing Machinery, New York, NY, USA, Article Article 32, 32 pages. https://doi.org/10.1145/2063384.2063427

[3] Anne Benoit, Aurélien Cavelan, Valentin Fevre, Yves Robert, and Hongyang Sun. 2017. Towards Optimal Multi-Level Checkpointing. *IEEE Trans. Comput.* 66, 7 (July 2017), 1212–1226. https://doi.org/10.1109/TC.2016.2643660

[4] Wahid Bhimji, Deborah Bard, Melissa Romanus, David Paul, Andrey Ovsyannikov, Brian Friesen, and Matt and Bryson. 2016. Accelerating Science with the NERSC Burst Buffer Early User Program. https://www.nersc.gov/assets/Uploads/Nersc-BB-EUP-CUG.pdf

[5] Mohamed Bouguerra, Ana Gainaru, Leonardo Bautista-Gomez, Franck Cappello, Satoshi Matsuoka, and Naoya Maruyama. 2013. Improving the Computing Efficiency of HPC Systems Using a Combination of Proactive and Preventive Checkpointing. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing.* 501–512. https://doi.org/10.1109/IPDPS.2013.74

[6] Anwesha Das, Frank Mueller, Paul Hargrove, Eric Roman, and Scott Baden. 2018. Doomsday: Predicting Which Node Will Fail When on Supercomputers. In *Supercomputing.* 9:1–9:14. https://doi.org/10.1109/SC.2018.00012

[7] Anwesha Das, Frank Mueller, Charles Siegel, and Abhinav Vishnu. 2018. Desh: Deep Learning for System Health Prediction of Lead Times to Failure in HPC. In *Symposium on High Performance Distributed Computing.* 40–51. https://doi.org/10.1145/3208040.3208051

[8] Sheng Di, Yves Robert, Frédéric Vivien, and Franck Cappello. 2017. Toward an Optimal Online Checkpoint Solution under a Two-Level HPC Checkpoint Model. *IEEE Transactions on Parallel and Distributed Systems* 28, 1 (Jan 2017), 244–259. https://doi.org/10.1109/TPDS.2016.2546248

[9] Aiman Fang and Andrew A. Chien. 2015. How Much SSD Is Useful for Resilience in Supercomputers *(FTXS '15)*. ACM, New York, NY, USA, 47–54. https://doi.org/10.1145/2751504.2751509

[10] Ana Gainaru, Franck Cappello, and William Kramer. 2012. Taming of the Shrew: Modeling the Normal and Faulty Behaviour of Large-scale HPC Systems. *2012 IEEE 26th International Parallel and Distributed Processing Symposium* (2012), 1168–1179.

[11] Ana Gainaru, Franck Cappello, Marc Snir, and William Kramer. 2012. Fault Prediction under the Microscope: A Closer Look into HPC Systems *(SC '12)*. IEEE Computer Society Press, Washington, DC, USA, Article Article 77, 11 pages.

[12] Rohan Garg, Tirthak Patel, Gene Cooperman, and Devesh Tiwari. 2018. Shiraz: Exploiting System Reliability and Application Resilience Characteristics to Improve Large Scale System Throughput. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN).* 83–94. https://doi.org/10.1109/DSN.2018.00021

[13] Al Geist and Christian Engelmann. 2003. Development of Naturally Fault Tolerant Algorithms for Computing on 100,000 Processors. (01 2003). https://www.csm.ornl.gov/~geist/Lyon2002-geist.pdf

[14] Cijo George and Sathish Vadhiyar. 2015. Fault Tolerance on Large Scale Systems using Adaptive Process Replication. *IEEE Trans. Comput.* 64, 8 (Aug 2015), 2213–2225. https://doi.org/10.1109/TC.2014.2360536

[15] Kamil Iskra, John W. Romein, Kazutomo Yoshii, and Pete Beckman. 2008. ZOID: I/O-forwarding Infrastructure for Petascale Architectures *(PPoPP '08)*. ACM, New York, NY, USA, 153–162. https://doi.org/10.1145/1345206.1345230

[16] Harsh Khetawat, Christopher Zimmer, Frank Mueller, Scott Atchley, Sudharshan Vazhkudai, and Misbah Mubarak. 2019. Evaluating Burst Buffer Placement in HPC Systems. In *IEEE Cluster.*

[17] Ning Liu, Jason Cope, Philip Carns, Christopher Carothers, Robert Ross, Gary Grider, Adam Crume, and Carlos Maltzahn. 2012. On the role of burst buffers in leadership-class storage systems. In *2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST).* 1–11. https://doi.org/10.1109/MSST.2012.6232369

[18] Yudan Liu, Raja Nassar, Chokchai Leangsuksun, Nichamon Naksinehaboon, Mihaela Paun, and Stephen L. Scott. 2008. An optimal checkpoint/restart model for a large scale high performance computing system. In *2008 IEEE International Symposium on Parallel and Distributed Processing.* 1–9. https://doi.org/10.1109/IPDPS.2008.4536279

[19] Robert Lucas, James Ang, Keren Bergman, Shekhar Borkar, William Carlson, Laura Carrington, and George Chiu. 2014. DOE Advanced Scientific Computing Advisory Subcommittee (ASCAC) Report: Top Ten Exascale Research Challenges. (2 2014). https://doi.org/10.2172/1222713

[20] Adam Moody, Greg Bronevetsky, Kathryn Mohror, and Bronis R. de Supinski. 2010. 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 (SC '10).* IEEE Computer Society, USA, 1–11. https://doi.org/10.1109/SC.2010.18

[21] Bogdan Nicolae, Adam Moody, Elsa Gonsiorowski, Kathryn Mohror, and Franck Cappello. 2019. VeloC: Towards High Performance Adaptive Asynchronous Checkpointing at Large Scale. In *IPDPS'19: The 2019 IEEE International Parallel and Distributed Processing Symposium.* Rio de Janeiro, Brazil, 911–920. https://hal.archives-ouvertes.fr/hal-02184203

[22] ORNL. 2020. *Spectral Library.* https://www.olcf.ornl.gov/spectral-library/

[23] Kento Sato, Naoya Maruyama, Kathryn Mohror, Adam Moody, Todd Gamblin, Bronis R. de Supinski, and Satoshi Matsuoka. 2012. Design and modeling of a non-blocking checkpointing system. In *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis.* 1–10. https://doi.org/10.1109/SC.2012.46

[24] Kento Sato, Kathryn Mohror, Adam Moody, Todd Gamblin, Bronis R. de Supinski, Naoya Maruyama, and Satoshi Matsuoka. 2014. A User-Level InfiniBand-Based File System and Checkpoint Strategy for Burst Buffers. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing.* 21–30. https://doi.org/10.1109/CCGrid.2014.24

[25] Bianca Schroeder and Garth A Gibson. 2007. Understanding failures in petascale computers. *Journal of Physics: Conference Series* 78 (jul 2007), 012022. https://doi.org/10.1088/1742-6596/78/1/012022

[26] Bianca Schroeder and Garth A. Gibson. 2010. A Large-Scale Study of Failures in High-Performance Computing Systems. *IEEE Transactions on Dependable and Secure Computing* 7, 4 (Oct 2010), 337–350. https://doi.org/10.1109/TDSC.2009.4

[27] SimPy Team. 2020. *SimPy: Discrete-Event Simulation for Python.* https://pypi.org/project/simpy/

[28] Devesh Tiwari, Saurabh Gupta, and Sudharshan S. Vazhkudai. 2014. Lazy Checkpointing: Exploiting Temporal Locality in Failures to Mitigate Checkpointing Overheads on Extreme-Scale Systems. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks.* 25–36. https://doi.org/10.1109/DSN.2014.101

[29] Sudharshan S. Vazhkudai, Bronis R. de Supinski, Arthur S. Bland, Al Geist, James Sexton, Jim Kahle, Christopher J. Zimmer, Scott Atchley, Sarp Oral, Don E. Maxwell, and et al. 2018. The Design, Deployment, and Evaluation of the CORAL Pre-Exascale Systems *(Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC '18)*. IEEE Press, 12.

[30] Lipeng Wan, Qing Cao, Feiyi Wang, and Sarp Oral. 2017. Optimizing checkpoint data placement with guaranteed burst buffer endurance in large-scale hierarchical storage systems. *J. Parallel and Distrib. Comput.* 100 (2017), 16 – 29. https://doi.org/10.1016/j.jpdc.2016.10.002

[31] Lipeng Wan, Matthew Wolf, Feiyi Wang, Jong Youl Choi, George Ostrouchov, and Scott Klasky. 2017. Comprehensive Measurement and Analysis of the User-Perceived I/O Performance in a Production Leadership-Class Storage System. In *IEEE 37th International Conference on Distributed Computing Systems (ICDCS '17).* 1022–1031.

[32] Chao Wang, Frank Mueller, Christian Engelmann, and Stephen L. Scott. 2008. Proactive Process-Level Live Migration in HPC Environments. In *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing.* https://doi.org/10.1145/1413370.1413414

[33] Chao Wang, Frank Mueller, Christian Engelmann, and Stephen L. Scott. 2012. Proactive Process-Level Live Migration and Back Migration in HPC Environments. *Journal of Parallel Distributed Computing* 72, 2 (Feb. 2012), 254–267. https://doi.org/10.1016/j.jpdc.2011.10.009

[34] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing*, Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 44–60.

[35] John W. Young. 1974. A first order approximation to the optimum checkpoint interval. *Commun. ACM* 17, 9 (1974), 530–531. https://doi.org/10.1145/361147.361115