SAND2017-5143C

It's not the heat, it's the humidity: scheduling resilience activity at scale

Patrick M. Widener[1], Kurt B. Ferreira[1], and Scott Levy[1]

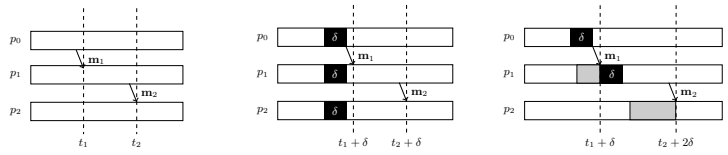
Center for Computing Research, Sandia National Laboratories*
Albuquerque, NM, USA
[pwidene|kbferre|sllevy]@sandia.gov

Abstract. Maintaining the performance of high-performance computing (HPC) applications with the expected increase in failures is a major challenge for next-generation extreme-scale systems. With increasing scale, resilience activities (e.g. checkpointing) are expected to become more diverse, less tightly synchronized, and more computationally intensive. Few existing studies, however, have examined how decisions about scheduling resilience activities impact application performance. In this work, we examine the relationship between the duration and frequency of resilience activities and application performance. Our study reveals several key findings: (i) the aggregate amount of time consumed by resilience activities is not an effective metric for predicting application performance; (ii) the duration of the interruptions due to resilience activities has the greatest influence on application performance; shorter, but more frequent, interruptions are correlated with better application performance; and (iii) the differential impact of resilience activities across applications is related to the applications' inter-collective frequencies; the performance of applications that perform infrequent collective operations scales better in the presence of resilience activities than the performance of applications that perform more frequent collective operations. This initial study demonstrates the importance of considering *how* resilience activities are scheduled. We provide critical analysis and direct guidance on how the resilience challenges of future systems can be met while minimizing the impact on application performance.

1 Introduction

Fault tolerance is a key challenge to building exascale systems. Next-generation systems are projected to have dramatically higher node counts than today's largest systems. The complexity and component count of individual nodes are also projected to grow. These two trends mean that future systems will experience more frequent failures than current systems. Moreover, power optimizations

* Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.



(a) w/o resilience activity

(b) synchronized resilience mechanism

(c) unsynchronized resilience mechcanism

Fig. 1. Example of how delays introduced by unsynchronized resilience mechanisms may propagate along application communication dependencies. The processes p_1, p_2, and p_3 exchange two messages m_1 and m_2 in each of the three scenarios. The black regions marked with a white δ denote the execution of coordinated (subfigure (b)) and uncoordinated (subfigure (c)) resilience activities. The grey regions denote periods in which the execution of a process is stalled due to an unsatisfied communication dependency.

(e.g., decreases in supply voltages) may further increase failure rates. Advances in component technology and system design mean that these systems may fail in new and different ways. In addition to fail-stop faults (e.g., node failure), Byzantine faults [18] due to silent data corruption may also be prevalent [3].

Currently, coordinated checkpoint/restart (cCR) [5] is the most commonly-used method for addressing failures on large-scale HPC systems. However, because the overhead of cCR grows as systems increase in size there is concern that cCR will no longer be a viable option for exascale systems [6]. First, the overhead of coordinating among application processes to determine when to take a checkpoint is expected to be prohibitive. Second, cCR (and checkpoint/restart in general) is only capable of handling fail-stop faults; by itself, it is not able recover from silent errors that may cause the application to produce incorrect results. Finally, as failures become more frequent, resilient operation may require a non-trivial amount of on-node computation to ensure that the application can continue to make meaningful progress. The combination of these factors means that resilience methods on future systems will be more diverse, less tightly synchronized, and more computationally intensive.

Significant effort has been devoted to developing alternatives to cCR that are able to effectively address failures on next-generation systems [6,8,19]. However, few of these existing studies have examined how decisions about scheduling resilience activities may impact application performance. The mechanism by which independently-scheduled resilience activities affect application performance is analogous to the impact of operating system noise on HPC applications, *see e.g.,* [7]. Drawing on this analogy, Figure 1 illustrates how applications may be affected by the degree to which resilience activities are synchronized across processes. Figure 1a represents the execution of a simple application running without resilience on three processes: p_0, p_1, and p_2. Time progresses from left to right. These processes exchange two messages, m_1 and m_2, at times t_1 and t_2, respectively. For the purposes of this discussion, we assume that these messages

represent tight dependencies: the receiving process will immediately stall if the expected message is delayed. Figure 1b shows how the application's execution is affected when the delays introduced by resilience are perfectly synchronized. Because each process is delayed by the same amount at the same time, the inter-process timings are preserved. In contrast, Figure 1b demonstrates the potential impact of allowing resilience mechanisms to execute in the absence of inter-process synchronization. For example, if the execution of the application on p_0 is delayed by the execution of a fault tolerance mechanism, then it may delay the transmission of message m_1. As a consequence, process p_1 stalls waiting for the arrival of this message. Moreover, this delay may ultimately propagate to process p_2 because of its dependency on communication from process p_1.

In this paper, we investigate how decisions about scheduling resilience activities affect application performance. Specifically, our initial study yields several key findings:

- The aggregate amount of time spent on resilience activities is not an effective metric for predicting application performance at scale (§3.1).
- The duration of interruptions due to resilience activities has the greatest influence on application performance; shorter, but more frequent, interruptions are correlated with better application performance (§3.1).
- The differential impact of resilience activities across applications is related to the applications' inter-collective frequencies; the performance of applications that perform infrequent collective operations scales better in the presence of resilience activities than the performance of applications that perform more frequent collective operations (§3.2).

This study of the importance of considering *how* resilience activities are scheduled has wide-ranging implications for fault-tolerant computing in general. It also provides critical analysis and direct guidance on how the resilience challenges of future systems can be met while ensuring that overheads remain tolerable.

2 Experimental approach

2.1 Modeling Local Checkpoint/Restart

In general, the communication structure of Message Passing Interface (MPI) programs cannot be determined offline because message matches cannot be established statically [2]. This makes modeling application performance analytically challenging even if all parameters of the application (e.g., the complete communication structure and all relative inter-process timings) are known. We therefore use a validated discrete-event simulation framework to evaluate the impact of local checkpointing activities on the performance of real applications.

Our simulation-based approach models checkpointing activities as CPU detours: periods of time during which the CPU is taken from the application and used to compute and commit checkpoint data. This approach allows a level of

fidelity and control not always possible in implementation-based approaches. It also allows us to examine application performance on systems that are much larger than those that are generally available for systems research.

Our simulation framework is based on `LogGOPSim` [13] and the tool chain developed by Levy et al. [20]. `LogGOPSim` uses the LogGOPS model, an extension of the well-known LogP model [4], to account for the temporal cost of communication events. An application's communication events are generated from traces of the application's execution. These traces contain the sequence of MPI operations invoked by each application process. `LogGOPSim` uses these traces to reproduce all communication dependencies, including indirect dependencies between processes which do not communicate directly.

`LogGOPSim` can also extrapolate traces from small application runs; a trace collected by running the application with p processes can be extrapolated to simulate performance of the application running with $k \cdot p$ processes. The extrapolation produces exact communication patterns for MPI collective operations and approximates point-to-point communications [13]. The validation of `LogGOPSim` and its trace extrapolation features have been documented previously [13]. Similarly, its ability to accurately predict local checkpointing overheads has also been documented [8, 20].

2.2 Simulating Different Resilience Schedules

To simulate the impact of depriving the application of CPU cycles in order to perform local resilience operations (like checkpoints), `LogGOPSim` accepts a *resilience activity trace*: an ordered list of events, expressed as the start time and duration of each event. In this paper, we use three different aggregate resilience activity percentages, each scheduled five different ways. The three percentages (1%, 5%, and 10%) represent an aggregate amount of computation time taken away from the application over the course of the entire run. These aggregate amounts are then scheduled five different ways: from a high frequency, low duration schedule; to a low frequency, high duration schedule, in each case the sum total of these schedules equal the aggregate percentage.

We additionally make two simplifying assumptions in this work for this investigation. First, we assume no failures. While including failures would not change our overall message and results, we leave them out of this work in order to better understand the measured overheads. Second, we assume no additional interference events occur in the run of the application (e.g. slowdowns due to true operating system noise).

In the remainder of the paper, we present results from simulation experiments based on the behavior of a set of four workloads. These workloads were chosen to be representative of scientific applications that are currently in use and computational kernels thought to be important for future extreme-scale computational science. They include:

- LAMMPS: A scientific application developed by Sandia National Laboratories to perform molecular dynamics simulations. For our experiments, we used the *Lennard-Jones*(LJ) and *2D crack* potentials [24].

– HPCCG: A conjugate gradient solver from the Mantevo suite of mini-applications [12].
– LULESH: An application that represents the behavior of a typical hydrocode [17].

LAMMPS is an important U.S. Department of Energy (DOE) application which runs for long periods of time on production machines and exhibits a range of different communication structures. HPCCG represents an important computational pattern in key HPC applications. LULESH is a proxy for important exascale applications developed by the DOE's Exascale Co-Design Center for Materials in Extreme Environments (ExMatEx).

3 Results and discussion

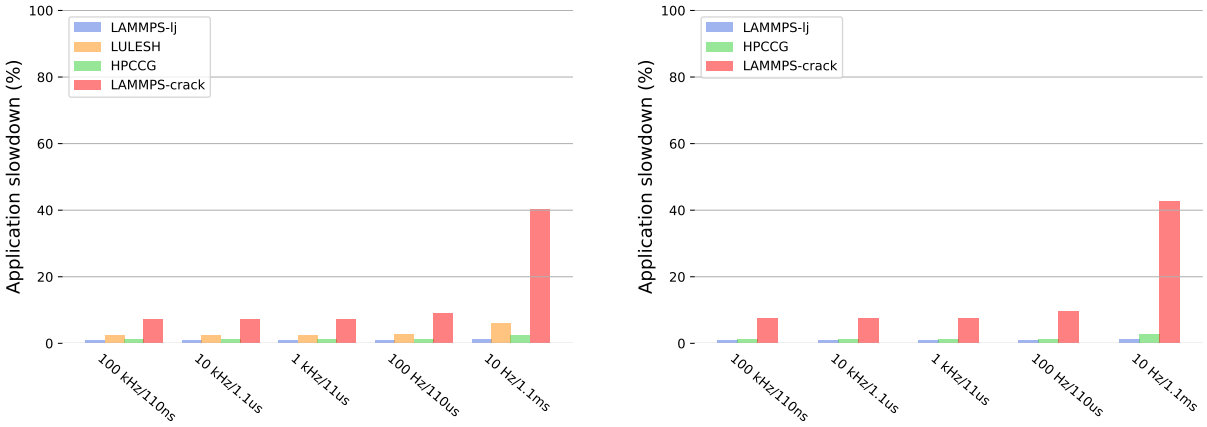
We conducted a set of experiments to explore the effect of different strategies for scheduling resilience activities on the runtime of our chosen workloads. To make our results applicable across a wide range of resilience strategies and application requirements, we explored three general classes of resilience-related activity. These classes are characterized by the percentage of total application runtime taken up by resilience activity; we studied cases where 1%, 5%, and 10% of application time was used.

For each case, we explored different representations of the actual resilience activity. While the total time taken for resilience might sum up to, say, 5% of application runtime, the *frequency* and *duration* of those activities can vary depending on overall resilience strategy, hardware capabilities, contention for storage, and other factors. We have explored the tradeoffs between frequency and duration in uncoordinated checkpointing systems in previous work [8,25]. We focused in these experiments, however, on modeling this tradeoff more abstractly.

We generated a *detour list* for a set of discrete frequency/duration combinations in each of the 1%, 5%, and 10% cases. A detour list consists of a set of pairs $(timestamp, duration)$ indicating when each detour begins and how long it lasts, representing the particular frequency/duration tradeoff for a particular scenario. We then conducted simulations using execution traces of our chosen workloads and each detour list, simulating the execution of the workload in the presence of the indicated resilience activity pattern and (implicitly) amount. For each case, we simulated the the effects of the following combinations of detour frequency and duration: 100KHz/110ns, 10KHz/1.1μs, 1KHz/11μs, 100Hz/110μs, and 10Hz/1.1ms.

We present results for all of our chosen workloads with 32Ki simulated processes, and due to technical constraints, for 3 workloads with 64Ki simulated processes[1].

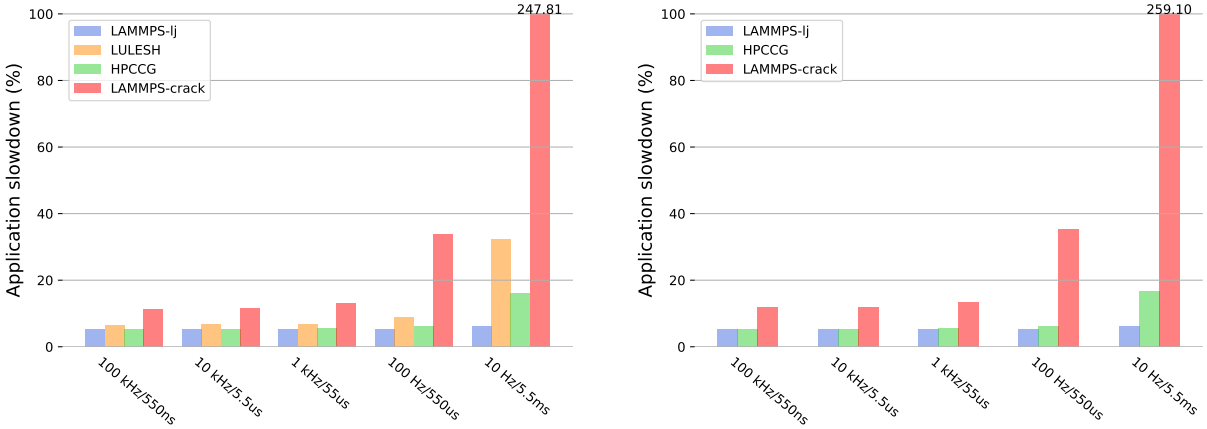
[1] We use the binary prefixes defined by the International Electrotechnical Commission (IEC). For example, 1Ki processes denotes $2^{10} = 1024$ processes.



(a) 1% resilience activity, 32Ki processes.

(b) 1% resilience activity, 64Ki processes.

Fig. 2. 1% resilience activity with varying frequency/duration compositions.



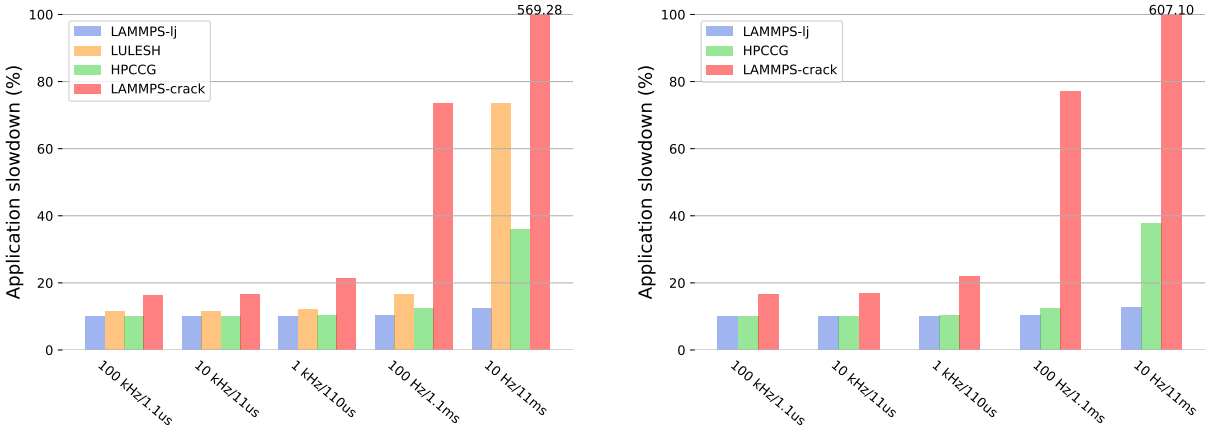
(a) 5% resilience activity, 32Ki processes.

(b) 5% resilience activity, 64Ki processes.

Fig. 3. 5% resilience activity with varying frequency/duration compositions. The bars for LAMMPS-crack at 10Hz/5.5ms in each plot have been truncated; the magnitude is displayed as an annotation in the plot.

3.1 Discussion

The results of our experiments are presented in Figures 2 to 4. These figures plot the total application time-to-solution slowdown for each scenario. The most general result of note from these figures is that each application behaves dif-



(a) 10% resilience activity, 32Ki processes.

(b) 10% resilience activity, 64Ki processes.

Fig. 4. 10% resilience activity with varying frequency/duration compositions. The bars for LAMMPS-crack at 10Hz/11ms in each plot have been truncated; the magnitude is displayed as an annotation in the plot.

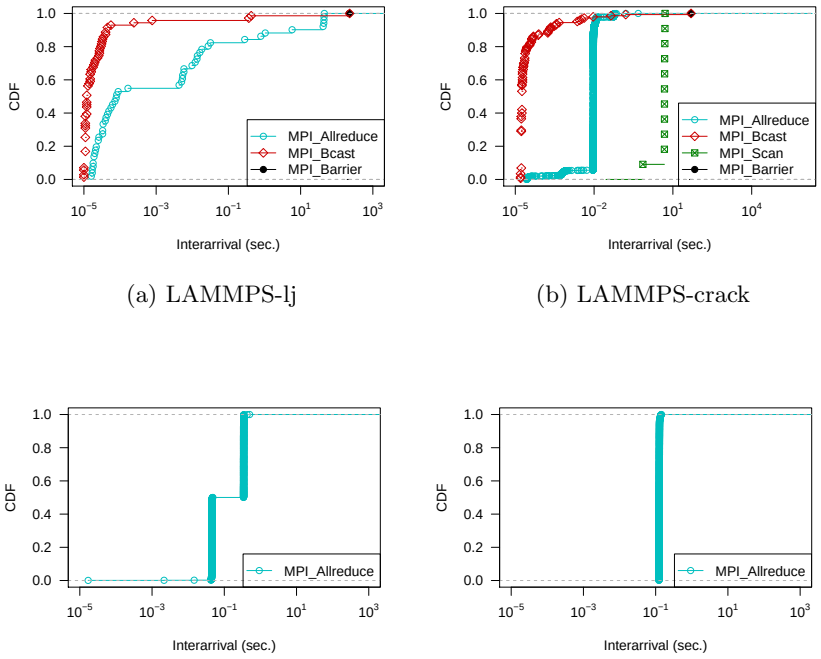
ferently under each resilience activity schedule, with LAMMPS-crack showing the greatest impacts and LAMMPS-lj showing the least. Also of significance is that the composition of a resilience activity (the frequency and duration) has a greater effect on application runtime than does the aggregate amount of that resilience activity. This is easily visible in all of the cases (Figures 2 to 4), where increasing the duration of detours eventually results in significant slowdowns for all our tested workloads, even as the total time taken in detours remains the same. Similar results were observed at the two different simulated process counts we studied, implying that the effect of increasing duration appears to be generally insensitive to application size.

Our results also raise the possibility that there may not be a strictly linear relationship between application slowdown and the proportion of runtime spent servicing each detour event. In other words, a factor of 5 increase in duration between two cases, does not imply a factor 5 overall slowdown in application performance. In fact, in most cases it is strictly less. Lastly, it is important to note that for each of the aggregate noise cases (1%, 5% and 10%), there exists a fine-grained schedule that significantly reduces overall impact and therefore can possibly be exploited by future applications.

3.2 Application Inter-collective Times

In this section, we examine the reasons behind the differential performance impact across applications described in the previous section. Specifically, we ex-

amine the relationship between application performance and the application's inter-collective period.



(a) LAMMPS-lj (b) LAMMPS-crack

(c) HPCCG (d) LULESH

Fig. 5. Discrete cumulative distribution function (CDF) of the MPI collective inter-arrival time for each application. NOTE: the CDF for `MPI_Barrier()` is represented by a single point for both LAMMPS-crack and LAMMPS-lj because only two such operations occur during their execution.

Figure 5 shows the discrete cumulative distribution functions (CDF) of the inter-collective periods for the MPI collective operations performed by each of our workloads. In this figure, a point at (x, y) indicates that, for a given application, at least $(x * 100)\%$ of the inter-collective times are smaller than y seconds. For example, Figure 5d shows that for LULESH 100% of the inter-collective times for `MPI_Allreduce()` are less than 150 milliseconds.

Our first observation is that `MPI_Allreduce()` is the most common collective operation for all four workloads. In fact, for LULESH and HPCCG, `MPI_-`

`Allreduce()` is the only collective operation.[2] The next observation is that the frequency of `MPI_Allreduce()` varies significantly between applications:

- in HPCCG, the inter-collective times for `MPI_Allreduce()` are bimodal: approximately half are between 40 and 50 milliseconds, and approximately half are between 300 and 500 milliseconds;
- in LAMMPS-crack, 80% of the `MPI_Allreduce()` inter-collective times are between 9 and 10 milliseconds, but there are also a small number inter-collective periods that exceed 150 milliseconds;
- in LAMMPS-lj, half of the `MPI_Allreduce()` inter-collective times are between 10 and 100 microseconds, but more than 10% are in excess of 5 seconds; and
- in LULESH, all of the `MPI_Allreduce()` inter-collective times are approximately 100 milliseconds.

We also observe from these CDFs that for the total aggregate noise cases, applications which perform more frequent collective operations are slowed down more by resilience activities that are longer in duration but occur less frequently. The exact interplay between inter-collective periods and resilience activity durations is beyond the scope of this paper, but is fertile ground for future investigation.

4 Related Work

In this paper, we study how the schedule of a general resilience mechanism can influence HPC application performance. To the best of our knowledge, no published works explicitly examine the influence of schedules. In this section we attempt to provide an overview of more loosely-related work.

Our study has origins in published research that characterizes application behavior in the presence of OS noise [7]. Collectively, this research shows that the pattern of OS noise events determines the impact on application performance and the benefits of coordination. Moreover, it shows that the duration of an OS noise event can significantly slowdown application performance.

Closely related, Ferreira et al. [8] studied the effects of communication on uncoordinated checkpointing at scale. This previous work makes a number of contributions that relate directly to the present paper. First, the authors show, contrary to previous work in the area, that a completely uncoordinated local checkpointing protocol can lead to significant application slowdown at scale. These local checkpoints can lead to process delays that can propagate through messaging relations (typically MPI collectives) to other processes causing a cascading series of delays. To ameliorate these slowdowns, the authors demonstrate how a hierarchical (or clustered) checkpointing approach [11] typically used to reduce message log volumes also can be effective at reducing impacts from local

[2] Although `MPI_Allreduce()` is the only collective operation that we observed in our experiments, the occurrence of MPI collective operations may depend on the inputs provided to the application.

checkpoints. While our work has antecedents in this previous work, we investigate the role of fine-grained scheduling in reducing overheads for local resilience approaches.

Checkpoint/restart protocols in HPC systems have been extensively studied. There are many descriptions of the foundations of both coordinated and uncoordinated CR protocols available in the literature [16, 21]. Beyond uCR and cCR, many other checkpoint/restart protocols have been proposed. Alvisi et al. examined the performance impact of coarse-grained communication patterns on the performance of three communication-induced checkpoint/restart (ciCR) algorithms [1]. ciCR uses the application's communication patterns to avoid checkpoints that cannot be used to recover a consistent global state. Hierarchical checkpointing attempts to group application processes into clusters that communicate frequently with each other [11, 22]. cCR is used within a cluster and uCR plus message logging is used between clusters. Because the number of processes in a cluster is smaller than the total application, contention for filesystem resources is reduced. Also, because most of the communication is within a cluster, the volume of message log data is also reduced.

Significant research has been conducted on how to reduce checkpoint commit time. The approaches that have arisen out of this research include: compression [14], exploiting faster storage media [23], excluding unchanged memory contents from checkpoints [9, 10], and de-duplication [15].

In this paper, we extend the results of these studies of checkpointing and general resilience mechanisms to examine how best to schedule these activities to reduce application performance. Specifically, we show that, as a whole, lowering the duration of each resilience event is more important to performance than decreasing the frequency.

5 Conclusion

Near-future HPC application developers will need to understand the performance implications of their design choices. This is especially true for applications implementing fault-tolerance strategies, as predicted scalability ceilings force exploration of alternate approaches. The work we describe in this paper contributes in several ways. We have presented a simulation-based approach for examining the tradeoffs between resilience activity duration and frequency, without regard to a particular resilience strategy. Our results reinforce earlier performance characterizations of uncoordinated checkpointing which suggested that detour duration has greater impact than detour frequency. This paper adds to that understanding by confirming this result for a range of frequency/duration compositions of a particular detour profile.

We intend to pursue several directions of future work based on this research. One area of interest is characterizing the relationship between the overall amount of resilience activity and the duration of detours for particular applications. *Can an application's communication pattern suggest, a priori, how resilience activities should be scheduled to minimize the impact on application performance?* We also

plan to extend our study to additional workloads and a wider range of application sizes.

References

1. Alvisi, L., Elnozahy, E., Rao, S., Husain, S., de Mel, A.: An analysis of communication induced checkpointing. In: Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on. pp. 242–249 (1999)
2. Bronevetsky, G.: Communication-sensitive static dataflow for parallel message passing applications. In: Proceedings of the 7th annual IEEE/ACM International Symposium on Code Generation and Optimization. pp. 1–12. IEEE Computer Society (2009)
3. Cappello, F., Geist, A., Gropp, W., Kale, S., Kramer, B., Snir, M.: Toward exascale resilience: 2014 update. Supercomputing Frontiers and Innovations 1(1) (2014), http://superfri.org/superfri/article/view/14
4. Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R., von Eicken, T.: LogP: Towards a realistic model of parallel computation. In: Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pp. 1–12. PPOPP '93, ACM, New York, NY, USA (1993), http://doi.acm.org/10.1145/155332.155333
5. Elnozahy, E.N., Alvisi, L., Wang, Y.M., Johnson, D.B.: A survey of rollback-recovery protocols in message-passing systems. ACM Computing Surveys 34(3), 375–408 (2002)
6. Ferreira, K., Riesen, R., Stearley, J., III, J.H.L., Oldfield, R., Pedretti, K., Bridges, P., Arnold, D., Brightwell, R.: Evaluating the viability of process replication reliability for exascale systems. In: Proceedings of the ACM/IEEE International Conference on High Performance Computing, Networking, Storage, and Analysis, (SC11) (Nov 2011)
7. Ferreira, K.B., Bridges, P., Brightwell, R.: Characterizing application sensitivity to os interference using kernel-level noise injection. In: Proceedings of the 2008 ACM/IEEE conference on Supercomputing. p. 19. IEEE Press (2008)
8. Ferreira, K.B., Levy, S., Widener, P., Arnold, D., Hoefler, T.: Understanding the effects of communication and coordination on checkpointing at scale. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC14). pp. 883–894. IEEE Press (2014)
9. Ferreira, K.B., Riesen, R., Brighwell, R., Bridges, P.G., Arnold, D.: libhashckpt: hash-based incremental checkpointing using GPUs. Recent Advances in the Message Passing Interface pp. 272–281 (2011)
10. Gioiosa, R., Sancho, J.C., Jiang, S., Petrini, F., Davis, K.: Transparent, incremental checkpointing at kernel level: a foundation for fault tolerance for parallel computers. In: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing. p. 9. IEEE Computer Society (2005)
11. Guermouche, A., Ropars, T., Brunet, E., Snir, M., Cappello, F.: Uncoordinated checkpointing without domino effect for send-deterministic MPI applications. In: International Parallel Distributed Processing Symposium (IPDPS). pp. 989–1000 (may 2011)
12. Heroux, M.A., Doerfler, D.W., Crozier, P.S., Willenbring, J.M., Edwards, H.C., Williams, A., Rajan, M., Keiter, E.R., Thornquist, H.K., Numrich, R.W.: Improving performance via mini-applications. Sandia National Laboratories, Tech. Rep (2009)

13. Hoefler, T., Schneider, T., Lumsdaine, A.: LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. pp. 597–604. ACM (Jun 2010)
14. Ibtesham, D., Arnold, D., Bridges, P.G., Ferreira, K.B., Brightwell, R.: On the viability of compression for reducing the overheads of checkpoint/restart-based fault tolerance. In: Parallel Processing (ICPP), 2012 41st International Conference on. pp. 148–157. IEEE (2012)
15. Islam, T.Z., Mohror, K., Bagchi, S., Moody, A., De Supinski, B.R., Eigenmann, R.: McrEngine: A scalable checkpointing system using data-aware aggregation and compression. In: High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for. pp. 1–11. IEEE (2012)
16. Johnson, D.B., Zwaenepoel, W.: Recovery in distributed systems using asynchronous message logging and checkpointing. In: Proceedings of the seventh annual ACM Symposium on Principles of distributed computing. pp. 171–181 (1988)
17. Karlin, I., Bhatele, A., Chamberlain, B.L., Cohen, J., Devito, Z., Gokhale, M., Haque, R., Hornung, R., Keasler, J., Laney, D., Luke, E., Lloyd, S., McGraw, J., Neely, R., Richards, D., Schulz, M., Still, C.H., Wang, F., Wong, D.: LULESH programming model and performance ports overview. Tech. Rep. LLNL-TR-608824, Lawrence Livermore National Laboratory (December 2012)
18. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems (TOPLAS) 4(3), 382–401 (1982)
19. Levy, S., Ferreira, K.B., Bridges, P.G.: Improving application resilience to memory errors with lightweight compression. In: High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for. pp. 323–334. IEEE (2016)
20. Levy, S., Topp, B., Ferreira, K.B., Arnold, D., Hoefler, T., Widener, P.: Using simulation to evaluate the performance of resilience strategies at scale. In: High Performance Computing, Networking, Storage and Analysis (SCC), 2013 SC Companion:. IEEE (2013)
21. Maloney, A., Goscinski, A.: A survey and review of the current state of rollback-recovery for cluster systems. Concurrency and Computation: Practice and Experience (Apr 2009)
22. Monnet, S., Morin, C., Badrinath, R.: A hierarchical checkpointing protocol for parallel applications in cluster federations. In: Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International. p. 211. IEEE (2004)
23. Moody, A., Bronevetsky, G., Mohror, K., Supinski, B.R.d.: 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 (SC10). pp. 1–11. SC '10, IEEE Computer Society, Washington, DC, USA (2010), http://dx.doi.org/10.1109/SC.2010.18
24. Plimpton, S.: Fast parallel algorithms for short-range molecular-dynamics. Journal of Computational Physics 117(1), 1–19 (1995)
25. Widener, P., Ferreira, K., Levy, S.: Horseshoes and hand grenades: The case for approximate coordination in local checkpointing protocols. In: Lecture Notes in Computer Science: Proceedings of the European Conference on Parallel and Distributed Computing (Euro-Par) 2016: Workshop on Resiliency in High Performance Computing (Resilience) in Clusters, Clouds, and Grids. Springer Verlag, Berlin, Germany, Grenoble, France (Aug 2016)