Horseshoes and Hand Grenades: The Case for Approximate Coordination in Local Checkpointing Protocols

Patrick M. Widener(✉), Kurt B. Ferreira, and Scott Levy

Center for Computing Research, Sandia National Laboratories, Albuquerque, NM, USA
{pwidene,kbferre,sllevy}@sandia.gov

Abstract. Fault-tolerance poses a major challenge for future large-scale systems. Active research into coordinated, uncoordinated, and hybrid checkpointing systems has explored how the introduction of asynchrony can address anticipated scalability issues. While fully uncoordinated approaches have been shown to have significant delays, the degree of sychronization required to keep overheads low has not yet been significantly addressed. In this paper, we use a simulation-based approach to show the impact of synchronization on local checkpoint activity. Specifically, we show the degree of synchronization needed to keep the impacts of local checkpointing low is attainable with current technology for a number of key production HPC workloads. Our work provides a critical analysis and comparison of synchronization and local checkpointing. This enables users and system administrators to fine-tune the checkpointing scheme to the application and system characteristics available.

1 Introduction

In response to alarming projections of high failure rates due to the increasing scale and complexity of high-performance computing (HPC) systems [5], researchers have devoted significant effort to the development of methods and techniques that will enable the deployment of resilient extreme-scale HPC systems and applications.

The current *de facto* standard for fault tolerance on HPC systems is coordinated checkpoint/restart (cCR). The overhead of cCR increases with the number of application processes. Current projections indicate that on next-generation systems more than half of an application's execution time may be consumed by the overhead of cCR [15]. Much of this overhead results from contention for storage resources: at the end of each checkpoint interval every application process simultaneously attempts to write out its checkpoint data to persistent storage.

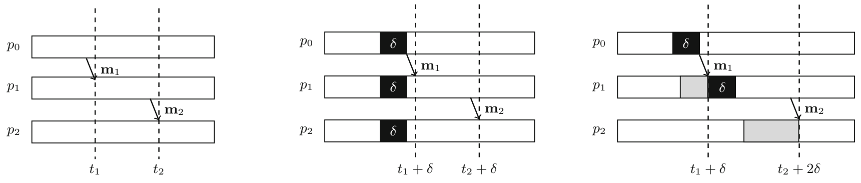
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly-owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND2016-5027C.

F. Desprez et al. (Eds.): Euro-Par 2016 Workshops, LNCS 10104, pp. 623–634, 2017.
DOI: 10.1007/978-3-319-58943-5_50

Uncoordinated checkpoint/restart (uCR) attempts to reduce contention for storage resources by allowing application processes to checkpoint independently. However, the performance impact of eliminating all inter-process coordination of checkpointing activities has been shown to be prohibitive because of the way that checkpointing-induced delays propagate and aggregate along communication dependencies [17].

In this paper, we examine the space between these two checkpoint protocol extremes, and investigate the impact of approximate checkpoint coordination on application performance. Approximate coordination reduces the contention for persistent storage resources in cCR and impedes the propagation of delays in uCR. Specifically, this paper makes the following contributions:

- a discussion of a new method, *approximate coordination*, for reducing the overhead of uCR;
- a description of a simulation-based approach for studying degrees of checkpoint coordination; and
- an initial examination of the impact of the degree of checkpoint coordination on application performance in an idealized scenario where no contention for persistent storage exists (e.g., node-local burst buffers are available for checkpoint storage).



(a) without checkpoint/restart (b) coordinated checkpoint/restart (c) uncoordinated checkpoint/restart

Fig. 1. Propagation of uncoordinated checkpointing delay through 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 δ represent delays due to the taking of checkpoints. The grey regions represent stalls due to unsatisfied message dependencies.

The remainder of this paper is structured as follows: Sect. 2 provides a discussion of checkpoint/restart and motivates our study of approximate coordination. Section 3 describes our experimental approach and Sect. 4 presents the results of these initial experiments. Section 5 discusses related work. Finally, Sect. 6 discusses potential future work and summarizes our initial study of approximate checkpoint coordination.

2 Background

The most common fault tolerance techniques on today's systems are based on checkpoint/restart. In the fundamental operation of checkpoint/restart, an application's processes periodically record their current state onto stable storage (creating a *checkpoint*). When a failure occurs, the application is *restarted* from a saved checkpoint. To ensure that a set of saved checkpoints represents a consistent state, some checkpoint/restart techniques require additional data to be saved (e.g., all sent messages). Several algorithms have been developed to ensure that a set of processes records a consistent state, deriving from seminal work on distributed system snapshots by Chandy and Lamport [8].

In this paper, we consider the two checkpoint/restart-based techniques introduced in the preceding section: cCR, and uCR. cCR stops the execution of all application processes at the same logical time and records a snapshot of the current state of each process. There are several benefits to cCR. Tight coordination of the timing of checkpoints across application processes ensures that the most recent checkpoint represents a consistent state of the machine [12]. As a result, there is no need to store multiple checkpoints or to record any other execution details (e.g., sent messages). Additionally, inter-process coordination of the timing of checkpoints limits the propagation of checkpointing-induced delays. As shown in Fig. 1b, because every process checkpoints simultaneously, the relative timing of inter-process communication events is preserved. However, because cCR requires that every application process take a checkpoint at the same time, contention for persistent-storage resources may degrade application performance. On next-generation systems, the overheads of coordination and those due to contention for storage resources may be prohibitive. In some cases, an application may spend more of its time on the overhead of cCR than on the computation for which it was designed [15].

To reduce the overhead of contention for storage resources, uCR allows every process to decide when to checkpoint entirely independently from its peers [7,18,22]. However, because of the lack of checkpoint coordination additional information is required in order to guarantee the existence of a set of checkpoints that represent a consistent state of the machine. One common way to resolve this issue is *message logging*. For example, if every process logs every message it sends, then when one process fails it restarts from its last checkpoint. The surviving processes re-send all of the messages that were sent to the failed process in the interval between its failure and its last checkpoint.

If the timing of checkpoints is entirely independent, checkpointing-induced delays can propagate and aggregate, much like OS noise (or *jitter*) [17]. For example, Fig. 1c shows how checkpointing-delays may propagate along communication dependencies. Because process p_0 is delayed because it is taking a checkpoint, process p_1 stalls waiting on the receipt of messages m_1, and the stall of p_1 causes p_2 to stall. Similarly, because process p_1 is subsequently delayed by its own checkpoint, process p_2 continues to stall waiting on the receipt of message m_2.

In this paper, we consider the novel question of whether and to what degree *approximate coordination* of the timing of checkpoints may be able to improve on the performance resulting from the total lack of coordination in uCR without incurring the overheads of resource contention in cCR. In other words, we seek to answer the question of *how uncoordinated* uCR can be while still limiting the propagation of checkpoint-induced delays.

3 Experimental Approach

In this section, we describe the experimental approach used to investigate the influence of approximate checkpoint coordination. First, we describe how we model the impact of checkpoint/restart techniques on application performance. We then discuss how we simulate various degrees of uCR checkpoint coordination.

3.1 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 [6]. 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 discrete-event simulation 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` [21] and the tool chain developed by Levy et al. [24]. `LogGOPSim` uses the LogGOPS model, an extension of the well-known LogP model [9], 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 [21]. The validation of `LogGOPSim` and its trace extrapolation features have been documented previously [20,21]. Similarly, its ability to accurately predict local checkpointing overheads has also been documented [17,24,25].

3.2 Simulating the Role of Coordination

To simulate the impact of depriving the application of CPU cycles in order to perform local checkpoints, `LogGOPSim` accepts a *checkpointing trace*: an ordered list of checkpoints, expressed as the start time and duration of each checkpointing event. In this paper, we use a checkpoint interval of 120 s and a checkpoint commit time of 1 s. Although the optimal checkpoint interval is not known unless checkpoints are totally coordinated, this checkpoint interval would be optimal for cCR on a platform whose system MTBF is approximately 2 h.

`LogGOPSim` can simulate the degree of checkpointing coordination among application processes by adding an initial offset to the replay of the execution trace. Using an initial offset of zero for application processes will simulate a perfectly coordinated checkpointing scheme. At the other extreme, choosing a uniformly distributed random initial offset for each simulated process will simulate a completely uncoordinated approach. Choosing this offset randomly from a normal distribution will simulate different degrees of coordination depending on the standard deviation of the distribution used. Example probability density functions are shown in Fig. 2. The x-axis in this figure is the time offset from the mean and the y-axis is the probability of a node using that offset value. This figure helps illustrate the range of the degree of approximate coordination that we consider. From the figure, as expected, the greater the standard deviation, the greater the likelihood of a large offset value.

We make two simplifying assumptions in our simulation approach:

- The perfect process synchronization we simulate is not achievable in practice. Even using strong coordination protocols such as those derived from Chandy & Lamport, there will still be some time skew between checkpoint commits in a real-world system. Using a simulation approach allows us to apply a global clock to all simulated process checkpoints.
- Our checkpointing simulation assumes no contention for storage resources even when checkpoints are tightly coordinated. In practice, storage resources are typically shared — even node-local ones such as burst buffers. By disregarding contention for these resources, we can observe directly the impact of coordination in local checkpoint propagation.

As a result of these assumptions, the data we present may be optimistic for highly-coordinated checkpointing cases.

3.3 Application Descriptions

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:

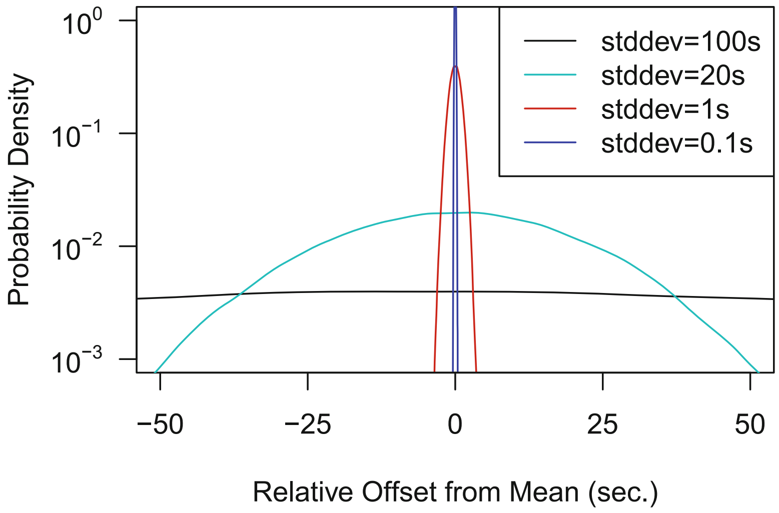


Fig. 2. Normally distributed probability density function of the degree of coordination as a function of standard deviation.

- LAMMPS: A scientific application developed by Sandia National Laboratories to perform molecular dynamics simulations. For our experiments, we used the *Lennard-Jones*(LJ) potential [30].
- CTH: A code developed at Sandia National Laboratories for modeling complex problems that are characterized by large deformations or strong shocks [11].
- HPCCG: A conjugate gradient solver from the Mantevo suite of mini-applications [19,31].
- LULESH: An application that represents the behavior of a typical hydrocode [23].

CTH and LAMMPS are important U.S. Department of Energy (DOE) applications which run for long periods of time on production machines and exhibit a range of different communication structures. HPCCG represents an important computational pattern in key HPC applications. LULESH is an exascale application proxy from the DOE ExMatEx co-design center [14].

4 Results

We conducted a set of experiments to quantify the effects of checkpointing synchronization for uCR in our chosen workloads. As described in the previous section, we staggered the starting offset of simulated checkpointing activity for each simulated process to produce different degrees of synchronization. Completely uncoordinated checkpointing is simulated by choosing a uniformly distributed random starting offset for each process, and completely coordinated checkpointing by using the same offset for each process. Producing offsets representing varying degrees of synchronization is done by drawing values from a normal distribution with mean 0 and a given standard deviation; changing the standard deviation of the distribution changes the degree of synchronization. We chose the following standard deviations for our trials: 1 μs, 100 μs, 75 ms, 100 ms, and 1, 20, 40, 60, 80, and 100 s. In our discussion below we refer to each different distribution of offsets by the value of its associated standard deviation.

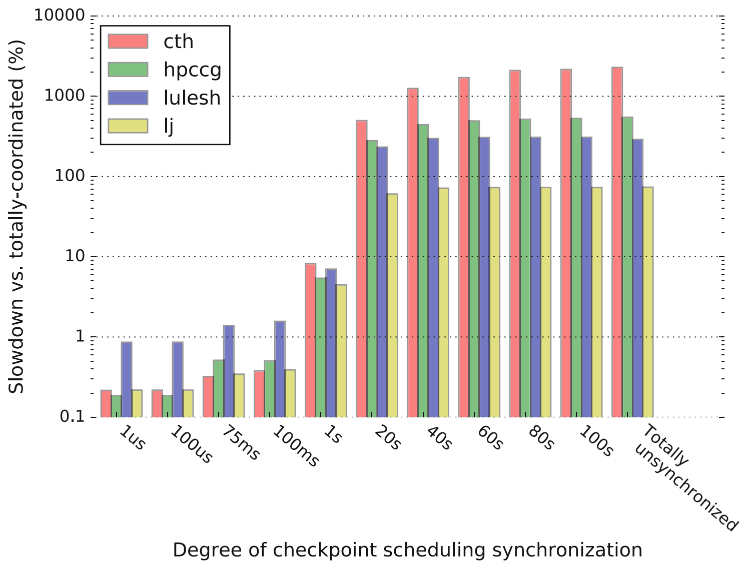
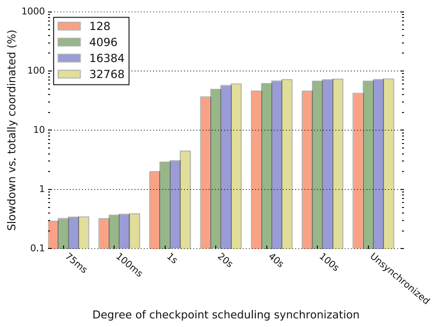


Fig. 3. Application slowdown with varying degrees of synchronization at 32Ki processes, measured relative to each totally coordinated case.

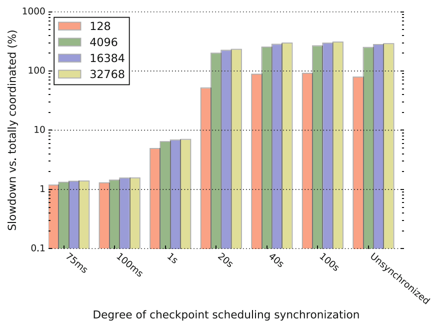
The results of these experiments are presented in Figs. 3 and 4. Figure 3 shows the application slowdown caused by using uCR with varying degrees of synchronization for each of our representative workloads at a fixed application size of 32 Ki processes[1]. We then examined each application in detail for slowdowns at varying process counts (Fig. 4). In each of these figures, we present the slowdown as a percentage of the runtime for a cCR (totally-coordinated checkpointing) execution of the simulation.

For each of the workloads we studied, a significant and increasing performance slowdown occurs as checkpoint synchronization among processes is relaxed beyond 100 ms. Previous work in this area has demonstrated that completely unsynchronized checkpointing will result in severe slowdowns [17]; as this figure also makes clear, some synchronization between checkpointing processes is necessary.

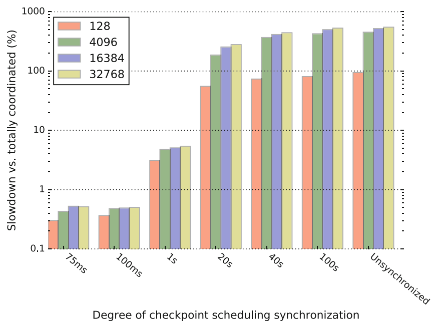
[1] Throughout this paper, we use the binary prefixes defined by the International Electrotechnical Commission (IEC). For example, 1 Ki processes is equivalent to 1024 processes.



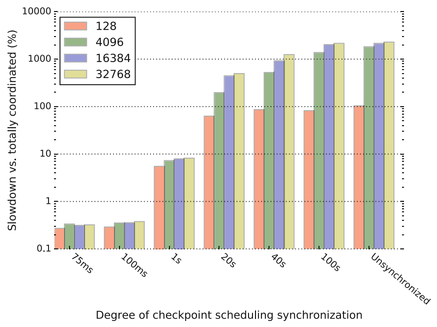
(a) LAMMPS-LJ



(b) LULESH



(c) HPCCG



(d) CTH

Fig. 4. Slowdown in applications at different process counts with varying degrees of checkpoint coordination, measured relative to each totally coordinated case.

Determining the degree of synchronization required in order to maintain performance then becomes the issue, and it is here that our results imply an important insight. Relatively loose synchronization is sufficient to keep the slowdown induced by checkpointing activity to a level much lower than that produced by completely uncoordinated checkpointing. The results presented in Fig. 4 show that if the pattern of checkpoints for all processes follows a normal distribution with standard deviation 100 ms (i.e., on average, 95% of process checkpoints will occur in a 200 ms time window), application runtime is increased by less than 5%. Synchronizing processes to this degree is well within the capabilities of systems with hardware support (such as a dedicated global interconnect or specialized equipment such as a GPS card), which have achieved clock skews on the order of 1μs [1,2]. Even software-based solutions such as NTP are able to achieve synchronization well within the 100 ms case we discuss here [27,29].

Even extremely loose synchronization with standard deviation on the order of 1 s produces approximately 10% application slowdown for our studied workloads. This is a value easily realizable in modern HPC systems and is also possible with acceptable reliability in wide-area or cloud-computing contexts. We also note that much tighter synchronization of uCR checkpointing does not improve performance markedly over that provided at the 100 ms standard-deviation value. Finally, this result is observed for a range of process counts, indicating that this effect is relatively insensitive to scale for this degree of synchronization.

5 Related Work

In this paper, we study the impact of approximate checkpoint coordination on application performance. In this section, we provide an overview of related publications.

The xSim simulator [13] has been used to study the effects of interference amplification and absorption on MPI collectives. Its authors propose its use as a tool for future HPC hardware/software co-design. Pradipta De et al. [10] proposed an emulation approach for studying similar performance impacts. Our simulation framework differs from these approaches in its ability to simulate interference overheads for systems of tens or hundreds of thousands of processes with modest hardware requirements. Our simulation framework also makes a different tradeoff between the level of detail produced and simulation time than these tools do, allowing it to be used for rapid evaluation of different application configurations.

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 [4,22,26]. The complete lack of checkpoint coordination in uCR has been frequently relied upon as an important feature [7,18,22].

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 [3]. ciCR uses the application's communication patterns to avoid checkpoints that cannot be used to recover a consistent global state. Hierarchical checkpointing attempts group application processes into clusters that communicate frequently with each other [18,28]. 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.

Our study has origins in published research that characterizes application behavior in the presence of OS noise [16,20]. 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 perfect coordination of OS noise events can significantly reduce performance impact.

Ferreira et al. [17] used an analogy to OS noise to show that when the timing of checkpoints is completely uncoordinated, checkpoint-induced delays have the potential to propagate and significantly degrade application performance.

In this paper, we extend the results of these studies of OS noise to examine how approximate coordination impacts application performance. Specifically, we show that perfect coordination is unnecessary to mitigate the performance cost of uCR; it may be sufficient to approximately coordinate OS noise events (e.g., local checkpoints).

6 Conclusions and Future Work

Developers of resilient HPC applications face design decisions about how best to implement fault-tolerance measures for extreme-scale computing. As coordinated checkpointing reaches a predicted scalability ceiling, a better understanding of the performance implications of introducing uncoordinated checkpointing is necessary. This paper contributes in several ways. We have: introduced the concept of *approximate coordination* for reducing the overhead of uncoordinated checkpointing; described a validated simulation-based technique for studying the coordination of processes using uncoordinated checkpointing; and presented an examination, carried out using our simulation framework, of the impact that varying degrees of checkpoint coordination has on application performance. Our results show that, while a degree of coordination between processes is necessary in order to avoid severe performance penalties, this degree can be quite modest. Dedicated hardware support for process synchronization is not necessary, and software-based coordination provides a degree of synchronization sufficient to keep checkpointing-related performance slowdowns below 10%.

We are pursuing several directions of future work based on this research. The projections we have presented here do not consider contention between processes for I/O bandwidth; we plan to refine our simulation approach to account for this. We also are working to determine which application and checkpointing features contribute to slowdowns in performance, and to characterize their interactions. Finally, we will use our extended simulation framework to provide more detailed information about how applications can leverage the synchronization of their processes to avoid performance issues.

References

1. Adiga, N., et al.: An overview of the BlueGene/L supercomputer. In: ACM/IEEE 2002 Conference Supercomputing, p. 60, November 2002
2. Almási, G., Heidelberger, P., Archer, C.J., Martorell, X., Erway, C.C., Moreira, J.E., Steinmacher-Burow, B., Zheng, Y.: Optimization of MPI collective communication on BlueGene/L systems. In: Proceedings of the 19th Annual International Conference on Supercomputing, ICS 2005, NY, USA, pp. 253–262 (2005). http://doi.acm.org/10.1145/1088149.1088183
3. Alvisi, L., Elnozahy, E., Rao, S., Husain, S., de Mel, A.: An analysis of communication induced checkpointing. In: Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing, 1999, Digest of Papers, pp. 242–249 (1999)
4. Alvisi, L., Marzullo, K.: Message logging: pessimistic, optimistic, causal, and optimal. IEEE Trans. Softw. Eng. **24**(2), 149–159 (1998)

5. Bergman, K., Borkar, S., Campbell, D., Carlson, W., Dally, W., Denneau, M., Franzon, P., Harrod, W., Hill, K., Hiller, J., Karp, S., Keckler, S., Klein, D., Kogge, P., Lucas, R., Richards, M., Scarpelli, A., Scott, S., Snavely, A., Sterling, T., Williams, R.S., Yelick, K.: Exascale computing study: technology challenges in achieving exascale systems, September 2008. http://www.science.energy.gov/ascr/Research/CS/DARPA/exascale-hardware(2008).pdf
6. 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)
7. Cappello, F.: Fault tolerance in petascale/exascale systems: current knowledge, challenges and research opportunities. IJHPCA **23**(3), 212–226 (2009)
8. Chandy, K.M., Lamport, L.: Distributed snapshots: determining global states of distributed systems. ACM Trans. Comp. Syst. **3**(1), 63–75 (1985)
9. 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. SIGPLAN Not. **28**(7), 1–12 (1993)
10. De, P., Kothari, R., Mann, V.: A trace-driven emulation framework to predict scalability of large clusters in presence of OS jitter. In: 2008 IEEE International Conference on Cluster Computing, pp. 232–241. IEEE (2008)
11. Hertel Jr., E.S., Bell, R.L., Elrick, M.G., Farnsworth, A.V., Kerley, G.I., McGlaun, J.M., Petney, S.V., Silling, S.A., Taylor, P.A., Yarrington, L.: CTH: a software family for multi-dimensional shock physics analysis. In: Proceedings of the 19th International Symposium on Shock Waves, pp. 377–382 July 1993
12. Elnozahy, E.N., Alvisi, L., Wang, Y.M., Johnson, D.B.: A survey of rollback-recovery protocols in message-passing systems. ACM Comput. Surv. **34**(3), 375–408 (2002)
13. Engelmann, C.: Investigating operating system noise in extreme-scale high-performance computing systems using simulation. In: Proceedings of the 11th IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN 2013), pp. 11–13 (2013)
14. Exascale Co-Design Center for Materials in Extreme Environments (ExMatEx). http://exmatex.lanl.gov/. Accessed 10 June 2013
15. Ferreira, K., Riesen, R., Stearley, J., Laros, J.H., 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, (SC 2011), November 2011
16. Ferreira, K.B., Brightwell, R., Bridges, P.G.: Characterizing application sensitivity to OS interference using kernel-level noise injection. In: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (SC 2008), November 2008
17. Ferreira, K.B., Widener, P., Levy, S., Arnold, D., Hoefler, T.: Understanding the effects of communication and coordination on checkpointing at scale. In: Proceedings of the 2014 International Conference for High Performance Computing, Networking, Storage and Analysis (Supercomputing) (2014)
18. 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

19. 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. Technical report, SAND2009-5574, Sandia National Laboratories (2009)
20. Hoefler, T., Schneider, T., Lumsdaine, A.: Characterizing the influence of system noise on large-scale applications by simulation. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–11. IEEE Computer Society (2010)
21. 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 (2010)
22. 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)
23. 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. Technical report LLNL-TR-608824, Lawrence Livermore National Laboratory, December 2012
24. 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: 2013 SC Companion: High Performance Computing, Networking, Storage and Analysis (SCC). IEEE (2013)
25. Levy, S., Topp, B., Ferreira, K.B., Arnold, D., Widener, P., Hoefler, T.: Using simulation to evaluate the performance of resilience strategies and process failures. Technical report SAND2014-0688, Sandia National Laboratories (2014)
26. Maloney, A., Goscinski, A.: A survey and review of the current state of rollback-recovery for cluster systems. Concurr. Comput. Pract. Exp. **21**(12), 1632–1666 (2009). doi:10.1002/cpe.1413. ISSN 1532-0634
27. Mills, D.L.: Internet time synchronization: the network time protocol. IEEE Trans. Commun. **39**(10), 1482–1493 (1991)
28. Monnet, S., Morin, C., Badrinath, R.: A hierarchical checkpointing protocol for parallel applications in cluster federations. In: 2004 Proceedings of 18th International Parallel and Distributed Processing Symposium, p. 211. IEEE (2004)
29. Murta, C.D., Torres Jr., P.R.T., Mohapatra, P.: Characterizing quality of time and topology in a time synchronization network. In: IEEE Globecom 2006, pp. 1–5, November 2006
30. Plimpton, S.: Fast parallel algorithms for short-range molecular dynamics. J. Comput. Phys. **117**(1), 1–19 (1995)
31. Sandia National Laboratory: Mantevo project home page, January 2014. http://mantevo.org