

Asking the Right Questions: Benchmarking Fault-tolerant Extreme-scale Systems

Patrick M. Widener¹, Kurt B. Ferreira¹, Scott Levy², Patrick G. Bridges², Dorian Arnold², and Ron Brightwell¹

¹ Sandia National Laboratories*, Albuquerque, NM
[patrick.widener|kbferre|rbbrigh]@sandia.gov
² University of New Mexico, Albuquerque, NM, USA,
[slevy|bridges|darnold]@cs.unm.edu

Abstract. Much recent research has explored fault-tolerance mechanisms intended for current and future extreme-scale systems. Evaluations of the suitability of checkpoint-based solutions have typically been carried out using relatively uncomplicated computational kernels designed to measure floating point performance. More recent investigations have added scaled-down “proxy” applications to more closely match the composition and behavior of deployed ones. However, the information obtained from these studies (whether floating point performance or application runtime) is not necessarily of the most value in evaluating resilience strategies. We observe that even when using a more sophisticated metric, the information available from evaluating uncoordinated checkpointing using both microbenchmarks and proxy applications does not agree. This implies that not only might researchers be asking the wrong questions, but that the answers to the right ones might be unexpected and potentially misleading. We seek to open a discussion on whether benchmarks designed to provide predictable performance evaluations of HPC hardware and toolchains are providing the right feedback for the evaluation of fault-tolerance in these applications, and more generally on how benchmarking of resilience mechanisms ought to be approached in the exascale design space.

1 Introduction

Issues surrounding reliability in extreme-scale systems are now attracting substantial research attention. Increasing size, component counts, and complexity of modern and projected systems are presenting unpalatable reliability implications for applications. Some estimates point to exascale systems suffering multiple failures per hour, seriously hindering application throughput and thereby squandering considerable hardware and software investment.

The HPC community has arrived at this untenable position largely due to a strong emphasis on peak CPU performance as an indicator of design and deployment success.

* 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.

The growth in popularity of the Top 500 list, for example, has been extremely valuable as a means of increasing the visibility of issues in early extreme-scale system design. While understandable and even beneficial, this emphasis on floating point performance provides little information on how to design applications and system software to deal with the fast-approaching reliability crunch. The disconnect is underscored by the common practice of generating results for the Top 500 list using application runs that avoid the defensive resilience measures (i.e. checkpointing) that are pervasive in front-line HPC applications.

This reliability information deficit will greatly hinder the designers of HPC systems and applications. Similarly to how power and heat constraints are shaping design decisions for exascale systems, sustainable HPC design will need to accommodate fault-tolerance as a first-class objective. Evaluation of systems and their applications must become more sophisticated in order to properly inform the design process. Current evaluations of both systems and software rely strongly on CPU performance-oriented benchmarks such as the NAS Parallel Benchmarks suite. Conceived in an HPC environment with infrequent failures, such benchmarks will have decreasing utility. Another consideration is that such benchmarks do not reflect the behavior of front-line HPC applications. Several well-known applications consist of compositions of computational kernels designed to solve different stages of large problems, and their approaches to resiliency are driven by the nature of those compositions. Their checkpointing strategies change based on application state volatility in each stage of the problem (i.e. the behavior of the computational kernel in question).

Standardized, repeatable benchmarks and problem sets that reflect deployed application behavior and yield a more appropriate metric could alleviate these problems. Recent studies of resilience issues have investigated *efficiency*, defined as the proportion of application runtime spent on the problem rather than on fault-tolerance activities such as checkpoint/restart, as a more informative metric. Efficiency becomes increasingly appropriate as “effective” performance measurements are reduced by increasing system fault rates.

With this paper, we hope to encourage a discussion on useful methods of obtaining information on the application efficiency made possible by architecture, system software, and fault-tolerance design choices. We observe that correlations in efficiency that might be expected between computational kernels from the NAS benchmarks set, “mini-apps” from the Sandia Mantevo project, and front-line scientific codes are not strongly apparent. We believe this is an appropriate departure point on how benchmarking should be considered in a new HPC regime where resilience concerns trump CPU performance. A more complete evaluation of benchmarking in a comprehensive set of fault-tolerance scenarios as well as proposals for evolving existing benchmark sets into a standard fault-tolerance benchmarking suite remain for future work. Our preliminary study indicates, however, that a discussion among the HPC resilience community on how to properly consider the behavior of current and proposed extreme-scale systems and applications is both appropriate and timely.

2 Related Work

This section briefly discusses the context of our work in checkpointing and benchmarking research.

2.1 Checkpointing

The most prevalent method of defensive fault-tolerance mitigation in modern applications, coordinated checkpointing periodically writes global application or system state to stable storage [1]. Consistent application state snapshots are enforced through global barrier synchronization. When a process fails, all application processes can then be restarted from a known-good, globally consistent state. Algorithmic approaches borrowed from the distributed computing domain [1] allow applications to generate consistent checkpoints without using barriers, avoiding increasingly expensive global synchronization.

Other techniques from distributed computing have given rise to uncoordinated or asynchronous checkpointing [2,3,4,5,6,7,8,9]. In these systems, nodes do not synchronize when they checkpoint, but they also keep a log of their sent messages on stable storage. Nodes restoring from local asynchronous checkpoints can then reconstruct a local state consistent with the application global state by replaying inbound messages from other nodes (using their logs).

We are not strictly concerned here with contributing to the active research comparing and contrasting these two approaches. However, we note that an informed choice between them requires knowledge of the application: how much state is maintained locally, shared with other cooperating nodes, or is easily reconstructed and therefore a candidate for leaving out of a checkpoint. A benchmarking suite that allows such choices by more closely reflecting the composition and behavior of complex applications is more likely to provide guidance on which checkpointing method is appropriate.

2.2 Benchmarks for HPC

Several benchmarking suites are widely used in the high-performance computing community. The Top500 list, itself an evolution of the Mannheim Supercomputer Statistics lists of the late 1980s, introduced the use of the LINPACK [10] linear equation solver as a standard, repeatable benchmark tool. The growing popularity of the Top500 list arguably also led to a reductive emphasis on floating point performance as a design goal, although at the time the available headroom in HPC performance was so great as to make this reasonable.

Finding LINPACK an unsuitable representative for their applications, the designers of the NAS Parallel Benchmarks [11] created first a set of specifications and later a set of reference implementations of a group of small benchmark kernels. Although providing a more nuanced view of supercomputing performance through the aggregation of several types of computational problems, the emphasis on floating point performance persisted.

More recent efforts have addressed the need to represent computational tasks from deployed applications through the creation of proxy applications. Comprising simplified versions of those applications able to execute scaled-down problem instances, benchmark collections from various organizations such as NERSC [12], Sandia [13] and Argonne [14] have refined performance evaluation still further. As these approximations of deployed application behavior become more popular, realistic evaluations of resilience strategies and effects are being pursued through both direct experimentation and simulation [15]. We submit that an emphasis on more sophisticated metrics such as efficiency as well as a more comprehensive consideration of resilience approaches are logical next steps in this evolution.

3 Comparing Benchmarks to Applications

In this section we investigate the correlation between efficiency profiles of a mixed set of microbenchmark/proxy application/full application combinations using uncoordinated asynchronous checkpointing. We made these choices because: 1) in a coordinated scheme, since checkpoint durations and intervals are fixed, the behavior of benchmarks and applications would be similar; 2) alternatives to coordinated checkpointing are necessary for exascale systems; and 3) it seems necessary to examine alternatives to benchmarks (such as NAS) that were designed years ago for coordinated checkpointing environments.

We used a modified message trace-driven simulator [16] to measure application efficiency (a discussion of our modifications and validation of this simulation framework can be found in [15]). Using this framework we found that, using uncoordinated checkpointing, the choice of benchmark can be significant, as codes computing essentially the same problem produced varying performance results. We argue that these results do not necessarily imply the superiority of any particular benchmark; rather, they illustrate that, as different checkpointing approaches are dictated by increasing application/system scale and the associated reliability concerns, evaluating application efficiency may not be as straightforward as evaluating floating point performance.

3.1 Simulation Results & Discussion

Figures 1 through 3 show the uncoordinated checkpoint performance of a number of common HPC microbenchmark applications using the simulator described previously [15]. In each of these figures we assume a two minute checkpoint interval and a one second checkpoint commit time (time to write a checkpoint). For simplicity we assume there are no failures in the each of these runs, but note that failures should not change the shapes of the curves shown. Additionally, we assume a message logging protocol to keep all checkpoints consistent but attribute no runtime overhead to this logging protocol. We believe this is a reasonable assumption for send-deterministic applications given recent work on optimizing message log sizes [17,18,19,20]. The “D” size class of NAS benchmarks were used, and other problem instances were sized accordingly.

Figure 1 illustrates the efficiency of two popular conjugate gradient solvers, NAS-CG and the mini-application HPCCG [21]. From this figure, we see that the overhead of

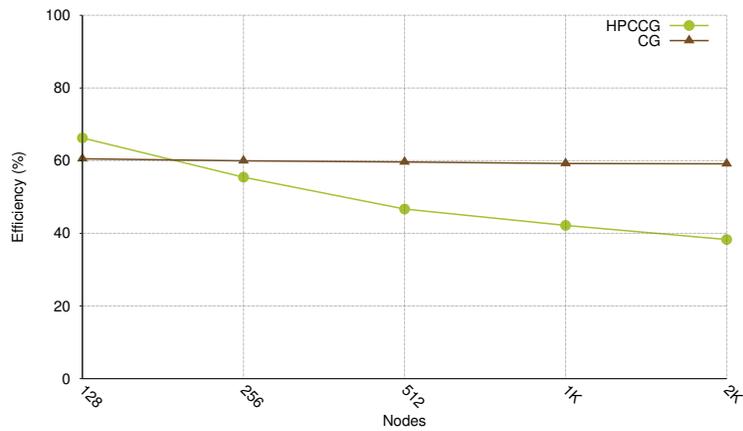


Fig. 1. Application Efficiency of NAS-CG and HPCCG with uncoordinated checkpointing

uncoordinated checkpointing varies greatly between these two workloads even though they are solving similar problems. Most importantly, given that node counts are expected to increase significantly in future systems, the trends suggest that the difference in performance between these two workloads increases with node count. In addition to the efficiency differences, the floating point performance of these two workloads also varies greatly. Using the same compiler, optimization flags, platform, and physical application layout, HPCCG executes 100x more flops/second than does CG.

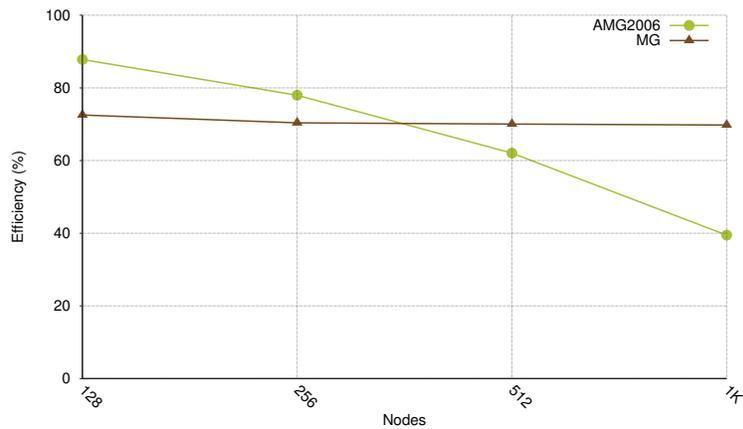


Fig. 2. Application Efficiency of the NAS-MG and AMG with uncoordinated checkpointing.

Similarly, Figure 2 shows the efficiency of NAS-MG and AMG from the ASC Sequoia Benchmark suite [22], two popular multi-grid linear system solvers used in many

DOE simulation workloads. Again, we see each of these workloads behave very differently while solving a similar problem. Also, again the difference in performance increases dramatically as node count increases.

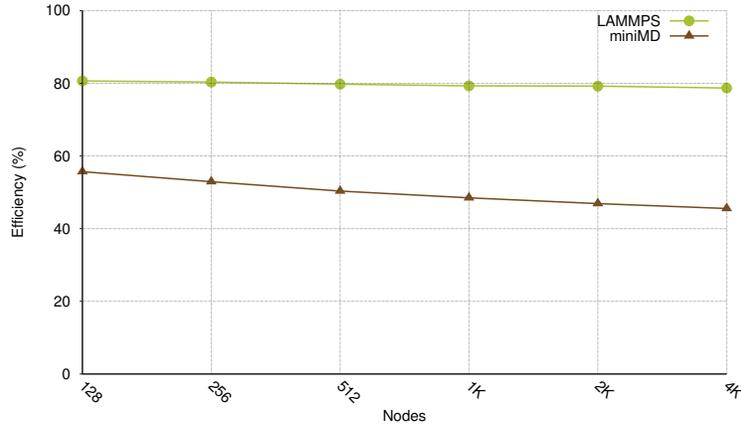


Fig. 3. Application Efficiency of miniMD and LAMMPS with uncoordinated checkpointing.

Finally, Figure 3 contains the efficiency results for the application LAMMPS [23,24] and the Mantevo mini-application molecular dynamics code miniMD [21]. Of the comparisons shown in this paper, miniMD and LAMMPS behave most similarly with uncoordinated checkpointing. This fact is not a surprise given these two workloads share a significant amount of source code and an overlap in the development teams. However, even with these workloads sharing considerable portions of source code, there exists a significant difference in performance with uncoordinated checkpointing.

As we see from this section, applications computing very similar problems and even using very similar algorithms can respond to a resilience mechanism (in this case uncoordinated checkpointing) in very different ways. In an effort to better understand the differences, we next examine each application’s communication pattern. In Figure 4 we show a comparison of the most frequently called MPI functions for each of the workloads. In this figure, we only include MPI functions which contribute to a significant portion of the total function count. From the figure we see that, as could be expected, for NAS-CG and HPCCG as well NAS-MG and AMG these benchmarks compute their respective problems in a very different manner, leading to their respective differences in efficiency. In contrast, the miniMD and LAMMPS MPI function profiles look very similar.

We believe these results point to two important issues when evaluating resilience for extreme-scale systems. First, as opposed to coordinated checkpointing in which checkpoints can be taken at known quiesced synchronization points, under uncoordinated checkpointing application communication patterns can significantly affect application efficiency. Second, understanding how communication patterns and other non-local op-

NAS-CG		HPCCG	
MPI_Send ()	33.33%	MPI_Send ()	19.97%
MPI_Irecv ()	33.33%	MPI_Irecv ()	19.97%
MPI_Wait ()	33.33%	MPI_Wait ()	19.97%
		MPI_Allreduce ()	19.97%

(a) NAS-CG and HPCCG

AMG		NAS-MG	
MPI_Iprobe ()	49.63%	MPI_Irecv ()	33.33%
MPI_Testall ()	49.63%	MPI_Wait ()	33.33%
MPI_Wait ()	0.36%	MPI_Send ()	32.88%

(b) AMG and NAS-MG

miniMD		LAMMPS	
MPI_Send ()	33.81%	MPI_Wait ()	30.56%
MPI_Irecv ()	32.23%	MPI_Send ()	30.56%
MPI_Wait ()	32.23%	MPI_Irecv ()	30.56%

(c) miniMD and LAMMPS

Fig. 4. Pairwise comparison of the most frequently-called MPI routines in each pair of programs as percentages of the total number of MPI calls.

erations impact performance is important in designing algorithms for faulty environments. These results point to a need to consider resilience a first-class concern in designing algorithms for future systems where failures will be common, as opposed to the resilience-as-an-afterthought approach that has been used successfully in the past with coordinated checkpointing.

4 Application vs. System Resilience

The choice between application-directed and system-directed resilience action has been explored in some detail. Our results in the previous section imply that appropriate resilience strategy choices are likely to be highly application dependent upon information not sufficiently exposed by microbenchmarks. Similarly, delegating responsibility for resilience to the operating system also disregards knowledge and context and can greatly affect performance measurements. This can take several forms:

- System-directed checkpointing ignores significant work that real applications do for their own resilience. Furthermore, system checkpoints are typically larger and (since they are conducted without input from the application) may contain redundant or stale data.
- Real applications have evolved downstream ecosystems based on consumption of checkpoint products such as telemetry, analysis/visualization and steering processes. System-directed checkpointing that assumes complete control of resilience policy decisions can interfere with these downstream processes and reduce their utility.

Ideally, the distinction between application- and system-provided resilience would become much less stark. System resilience utilities, as opposed to exercising complete control over policy decisions, should instead act more as resource monitors (as other operating system constructs such as schedulers and memory managers do). In such an arrangement, system checkpointing libraries would expose hardware resilience information and allow collaboration between applications and the system. This would allow applications to continue to satisfy downstream consumers of resilience data as well as provide a measure of fine-grain control to future benchmark suites.

5 Concluding Remarks

This work was motivated by our growing awareness of a disconnect between the increasing emphasis on resilience issues in exascale systems research and the nature of the tools being used to evaluate the responses to those issues. Classical benchmarking tools, while entirely appropriate for the era in which they were designed, will be of decreasing utility in the face of current exascale design trends. As we have discussed, several “next-generation” benchmarking approaches are under development which promise a more nuanced picture of the performance of real applications under the new constraints implied by those trends. We believe, however, that resilience will be a fundamental issue for exascale, and that a community-wide discussion of what kinds of benchmarks and metrics are most valuable is both necessary and timely.

The establishment and popularity of the Top500 list clearly had beneficial results, driving advances in hardware, system software and application design and implementation which in turn have had numerous trickle-down effects for HPC research. Other “500” lists have emerged, each lending visibility to specific areas: energy consumption and sustainability are highlighted by the Green500 (www.green500.org), and the different HPC design decisions necessary to support efficient graph-based analytics algorithms by the Graph500 (www.graph500.org). Resilience is becoming widely accepted as a core design goal for HPC systems, roughly paralleling the path that energy efficiency concerns followed to prominence. While it may strike some as facetious, a “Resilient500” list could have similar beneficial effects: addressing a core emerging design concern (as does the Green500) by spurring an interested community to establish commonly accepted information-rich metrics and benchmarks (as the Graph500 organizers are doing).

We also note that important vehicles for this type of discussion are already underway. The DOE ASCR Co-Design Centers [25] (ExaCT [26], ExMatEx [27], CESAR [28]) are each charged with an application area important for the future of exascale system design. In particular, these efforts will focus on the fusion of scientific problem requirements (which we have shown can have notable impact in performance evaluation) with architectural and system software concerns (providing visibility to hardware resilience issues as well as propagation of resilience information throughout the software stack). Their unique charter makes the Co-Design Centers an ideal venue for efforts to further develop and standardize resilience-aware benchmarking.

References

1. K. Mani Chandy and Leslie Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, 1985.
2. Jinho Ahn. 2-step algorithm for enhancing effectiveness of sender-based message logging. In *SpringSim '07: Proceedings of the 2007 spring simulation multiconference*, pages 429–434, 2007.
3. Q. Jiang and D. Manivannan. An optimistic checkpointing and selective message logging approach for consistent global checkpoint collection in distributed systems. In *Proceedings of the 2007 IEEE International Parallel and Distributed Processing Symposium*, March 2007.
4. David B. Johnson and Willy Zwaenepoel. Recovery in distributed systems using asynchronous message logging and checkpointing. In *Proceedings of the seventh annual ACM Symposium on Principles of distributed computing*, pages 171–181, 1988.
5. Lorenzo Alvisi and Keith Marzullo. Message logging: Pessimistic, optimistic, causal, and optimal. *IEEE Trans. Softw. Eng.*, 24(2):149–159, February 1998.
6. E. N. (Mootaz) Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, 2002.
7. George Bosilca, Aurelien Bouteiller, Franck Cappello, Samir Djilali, Gilles Fedak, Cecile Germain, Thomas Herault, Pierre Lemarinier, Oleg Lodygensky, Frederic Magniette, Vincent Neri, and Anton Selikhov. MPICH-V: Toward a scalable fault tolerant mpi for volatile nodes. In *Conference on High Performance Networking and Computing (SC2002)*, pages 1–18, Baltimore, MD, november 2002.
8. P. Lemarinier, A. Bouteiller, T. Herault, G. Krawezik, and F. Cappello. Improved message logging versus improved coordinated checkpointing for fault tolerant MPI. In *IEEE International Conference on Cluster Computing*, pages 115–124, 2004.
9. Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, 2005.
10. Jack J Dongarra, Piotr Luszczek, and Antoine Petit. The linpack benchmark: past, present and future. *Concurrency and Computation: practice and experience*, 15(9):803–820, 2003.
11. Rob F Van der Wijngaart and Parkson Wong. NAS parallel benchmarks version 2.4. Technical report, NAS technical report, NAS-02-007, 2002.
12. Katie Antypas, John Shalf, and Harvey Wasserman. NERSC-6 workload analysis and benchmark selection process. Technical Report LBLNL-1014E, Lawrence Berkeley National Laboratories, 2000.
13. Sandia National Laboratories. Mantevo. <http://software.sandia.gov/mantevo>.
14. CESAR, Argonne National Laboratory. The CESAR Proxy-apps. <https://cesar.mcs.anl.gov/content/software>. Retrieved 10 June 2013.
15. Kurt B. Ferreira, Scott Levy, and Patrick G. Bridges. A simulation infrastructure for examining the performance of resilience strategies at scale. Technical Report SAND 2013-3180, Sandia National Laboratories, April 2013.
16. Torsten Hoeftler, Timo Schneider, and Andrew Lumsdaine. Loggopsim: simulating large-scale applications in the LogGOPS model. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 597–604. ACM, 2010.
17. Rolf Riesen, Kurt Ferreira, Dilma Da Silva, Pierre Lemarinier, Dorian Arnold, and Patrick G. Bridges. Alleviating scalability issues of checkpointing protocols. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '12*, pages 18:1–18:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.
18. Amina Guermouche, Thomas Ropars, Marc Snir, and Franck Cappello. HyDEE: Failure containment without event logging for large scale send-deterministic mpi applications. In *IPDPS*, pages 1216–1227. IEEE Computer Society, 2012.

19. Thomas Ropars, Amina Guermouche, Bora Uçar, Esteban Meneses, Laxmikant V. Kalé, and Franck Cappello. On the use of cluster-based partial message logging to improve fault tolerance for mpi hpc applications. In Emmanuel Jeannot, Raymond Namyst, and Jean Roman, editors, *Euro-Par (1)*, volume 6852 of *Lecture Notes in Computer Science*, pages 567–578. Springer, 2011.
20. Amina Guermouche, Thomas Ropars, Elisabeth Brunet, Marc Snir, and Franck Cappello. Uncoordinated checkpointing without domino effect for send-deterministic MPI applications. In *International Parallel Distributed Processing Symposium (IPDPS)*, pages 989–1000, may 2011.
21. Sandia National Laboratory. Mantevo project home page. <https://software.sandia.gov/mantevo>, Apr. 10 2010.
22. Lawrence Livermore National Laboratory. Asc sequoia benchmark codes. <https://asc.llnl.gov/sequoia/benchmarks/#amg>, Apr. 6 2013.
23. Steven J. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal Computation Physics*, 117:1–19, 1995.
24. Sandia National Laboratory. LAMMPS molecular dynamics simulator. <http://lammps.sandia.gov>, Apr. 10 2010.
25. Advanced Scientific Computing Research program, Office of Science, US Department of Energy. SciDAC Co-Design. <http://science.energy.gov/ascr/research/scidac/co-design/>. Retrieved 10 June 2013.
26. Center for Exascale Simulation of Combustion in Turbulence (ExaCT). <http://exactcodesign.org/>. Retrieved 10 June 2013.
27. Exascale Co-Design Center for Materials in Extreme Environments (ExMatEx). <http://exmatex.lanl.gov/>. Retrieved 10 June 2013.
28. Center for Exascale Simulation of Advanced Reactors (CESAR). <https://cesar.mcs.anl.gov/>. Retrieved 10 June 2013.