SAND2013-9061C

Predicting Coordinated and Uncoordinated Checkpoint/Restart Protocol Performance at Extreme Scales

Kurt B. Ferreira
Scalable System Software
Sandia National Laboratories
Albuquerque, NM
kbferre@sandia.gov

Scott Levy, Bryan Topp and Dorian Arnold
Department of Computer Science
University of New Mexico
Albuquerque, NM
{*slevy,betopp,darnold*}*@cs.unm.edu*

Torsten Hoefler
Computer Science Department
ETH Zürich
Switzerland
htor@inf.ethz.ch

***Abstract*—Fault-tolerance has been identified as a major challenge for future Exascale-class systems. While many studies investigate this issue, few consider systems at the scales expected for future systems. Furthermore, most are performed on platforms with dramatically different architectural features (for example, network and persistent storage devices) than we expect on Exascale-class systems. Applying lessons learned from such studies to potential future systems can be difficult and even misleading. To address the lack good of analytical models for many emerging resilience techniques like hierarchical checkpointing, uncoordinated checkpointing, we have developed a simulation framework to evaluate checkpoint/restart techniques on large-scale systems. In this paper, we present our simulation framework, discuss how it accurately models the performance impacts of faults and fault-tolerance mechanisms on extreme-scale systems, validate its predictive abilities in a failure-prone environment and demonstrate the importance of this framework by gaining insights on the impact an application's computation pattern has on the performance of uncoordinated checkpointing, specifically a slowdown due to `MPI_Allreduce()`, and the performance impact of node failures on uncoordinated checkpointing.**

***Keywords*-Fault Tolerance; Rollback/Recovery; Exascale; Simulation;**

I. Introduction

We observe a steady growth in high-performance computing (HPC) system size over the last 20 years. The number of CPU cores or execution elements continues on an exponential growth trajectory that determines the complexity of the overall system. The vast number of parts (millions to billions) that are used in large-scale systems to solve a single computational problem requires us to consider failure as part of a normal execution. Indeed, once failure is accepted, one can consider trading additional reliability for lower cost systems. The projected mean time between failures of hours or minutes [1] for these large scale systems means that many HPC will be affected by failing components multiple times during each execution. This, in combination with growing scales and fault-tolerance protocol overheads, inevitably leads to exploding costs for fault tolerance.

The study of decreasing fault tolerance overheads is an active research area and a vast and growing collection of literature currently exists. The community has found no single silver bullet solution – the tradeoffs of existing systems depend on detailed characteristics of the application, the execution environment and the system architecture.

Two general fault tolerance schemes exist in the literature: (1) coordinated checkpoint/restart (CR), where processes checkpoint synchronously after quiescing the network and (2) uncoordinated CR, where processes checkpoint at different times to avoid the need to drain the whole network and to decrease filesystem contentions. When a single failure occurs in coordinated CR, all processes have to redo all of the work since their last checkpoints. In uncoordinated CR, processes can continue unless and until they depend on the failed process. When a surviving process encounters a dependency on a failed process it waits until the failed process catches up. Therefore, an uncoordinated checkpoint taken at one process could delay other processes on its critical path, leading to overall performance losses. The tradeoffs between the two protocol families are non-trivial.

Accurate analytic models for coordinated checkpointing exist [2], [3]. However, we lack such tools for predicting the performance impact of many other fault tolerance mechanisms (for example, message-logging [4], communication-induced checkpointing [5] and hierarchical checkpointing [6]). This leads to contradicting studies and results. For example, while several researchers claim that CR protocols may not be scalable in general [7]–[9], there has been a great effort in the research community to optimize such protocols to ensure they remain viable [4], [8], [10]–[13].

One major problem for comparing various studies, protocols and systems is the lack of a common and reproducible testbed. Most fault-tolerance techniques are designed for large-scale systems. Evaluating these techniques or reproducing previous results requires access to the largest systems available. In most cases, access to systems of these scales is difficult or impossible.

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.

In this work, we define a methodology and offer an implementation to serve as a research vehicle for analyzing the overheads of various fault tolerance protocols at large-scale. In general, simulation is a viable tool if interactions become too complex to be captured by an analytic model and direct execution is not feasible. However, cycle-accurate simulators may provide a level of detail that be unnecessary or prohibitively expensive. Thus, a good simulation model is tuned for a specific problem domain and allows researchers to focus specifically on relevant system properties. This allows researchers to strike a balance between accuracy and cost for the problem at hand. We propose a simulation model that is tailored to large-scale resilience research. We explain which parts of applications and systems need to be considered for this simulation to be accurate for modeling resilience while still providing an answer in a tractable amount of time.

Our work is based on the idea that, like system noise [14], checkpoint protocols can be modeled as CPU detours [15]. In this paper, we extend this initial work and design and verify that simulation of failure, recovery and rework in large-scale parallel applications can also be accurately modeled as CPU detours. We begin by describing our simulation infrastructure to present a self-contained holistic picture of our proposed approach for large-scale resiliency research. We then provide evaluate our framework by validating its output and quantifying its performance capabilities. Finally, we present a case study that demonstrates how this infrastructure can be used to gain insights into the impact of scale on common checkpointing protocols and how node failure can impact performance.

Specific contributions of this work include:

- A complete simulation tool-chain for resilience research based on `LogGOPSim` [16]. Our extensions to `LogGOPSim` allow the accurate simulation of CR mechanisms on large-scale systems with several million CPUs and days of application runtime.
- An evaluation of our simulator's ability to predict the impact of node failures on application performance. Using a standard model for coordinated checkpointing [2], we show that our simulator can accurately predict application execution time;
- A case study using this framework to study how asynchronous checkpoint/restart impacts the performance of important HPC applications;
- An examination of the relationship between the performance impact of uncoordinated checkpointing and the target application's communication pattern, specifically showing the impact of `MPI_Allreduce()`; and
- The performance impact of node failure rates with uncoordinated checkpointing.

II. Our Simulation Framework

In Levy et al. [15], we introduced the key *application*, *hardware* and *resilience mechanism* characteristics that are necessary to accurately simulate resilience on next-generation systems. First, we observed the cross-cutting need for simulation capabilities that can model large *spatial* and *temporal* scales. Large spatial scales allow us to simulate the behavior of future generation systems expected to comprise hundreds of thousands of nodes and millions of processors. Large temporal scales allow us to simulate the behavior of long running applications, particularly those whose behavior varies over time. Table I summarizes the key parameters that are necessary to simulate coordinated and uncoordinated checkpoint/restart mechanisms as well as failures.

A. Simulating Application Characteristics

Our current implementation is based on the freely available discrete-event simulator `LogGOPSim` [16]. `LogGOPSim` is an application simulator based on LogP [17]. `LogGOPSim` consists of three major components: (1) `liballprof`: a trace collector that records the actual MPI communication of the target application; (2) `SchedGen`: a schedule generator that inputs the MPI traces and generates a schedule that captures the application's control and data flow, preserving event *happens-before* relationships; and (3) a discrete-event simulator that reads the generated schedule, performs a full LogGOPS simulation and reports each process' completion time.

`LogGOPSim` can extrapolate traces collected on smaller scale systems to simulate the same communication characteristics at larger scales. This framework has been used to evaluate the performance of collective communications [18] and the impact of OS noise [14] on large-scale applications.

B. Simulating Hardware Characteristics

`LogGOPSim` allows us to model the impact of emerging interconnect technologies. Working within the LogGOPS model, we can simulate the impact of many changes in network behavior on checkpoint/restart techniques by modifying the model's parameters. In addition, as we discuss more fully below, our model of resilience mechanisms allows us to evaluate how improvements to persistent storage systems (for example, the widespread availability of local SSDs) will affect the application performance.

C. Simulating Failures and Resilience

In [15], we observed that resilience mechanisms (like writing checkpoints, restarting after a failure and redoing lost work) can be modeled as CPU *detours*, a number of CPU cycles that are used for something other than the application's computation, similar to OS noise [14], [19]. A key difference between OS noise and resilience detours is that resilience "noise" events may need to be replayed synchronously with

Required to Model	Parameter Name	Parameter Description
All Checkpointing	COORDINATION TIME	time for processes to coordinate the taking of a checkpoint
	CHECKPOINT COMPUTATION	time to compute a checkpoint
	CHECKPOINT COMMIT TIME	time to write a checkpoint to stable storage
	CHECKPOINT INTERVAL	time between consecutive checkpoints
	WORK TIME	time-to-solution without failures or resilience mechanisms
Uncoordinated Checkpointing	COMMUNICATION GRAPH	details of inter-process communication
	COMPUTATION EVENTS	failure-free computation pattern of the application
	DEPENDENCIES	partial ordering of communication and computation events
Failure Occurrences	FAILURE CHARACTERIZATION	rate and distribution of failures
	RESTART TIME	time to read a checkpoint from stable storage after a failure
	RECOVERY MODEL	a model of the time required before forward progress can resume

Table I
SUMMARY OF THE PARAMETERS NEEDED FOR ACCURATE SIMULATION OF HPC APPLICATIONS IN A FAILURE-PRONE SYSTEM.

the application's communication/computation pattern rather than asynchronously, as is typical for OS noise.

We developed a new library, `libsolipsis`, to generate CPU detours based on a specified resilience mechanism and the application's communication pattern. Similar to `liballprof`, the library links to the application using the MPI profiling interface, intercepting all MPI calls. The output of this library is a per-process detour file that can be provided as input to `LogGOPSim`. The detour file contains the timestamp and the duration of each of the resilience detours, T_{detour}, computed as follows:

$$T_{detour} = T_{coord} + T_{ckpt} + T_{commit}$$

where

T_{coord} = time to coordinate the taking of a checkpoint
T_{ckpt} = time to compute a checkpoint
T_{commit} = time to commit the checkpoint to stable storage

Our detours also capture the impact of node failures and optimistic message logging. For failures, T_{detour} includes both restart and rework time: rework time is calculated as the simulated time that has elapsed since the previous checkpoint. For optimistic message logging, `libsolipsis` calculates the time required to write the message to the log given a bandwidth to stable storage.

Coordinated checkpoint/restart: For coordinated checkpoint/restart, we generate a detour file that contains the timestamp and the duration of each checkpoint taken by the application. We assume bulk-synchronous parallel (BSP) applications. Because BSP applications are largely self-synchronizing, we set $T_{coord} = 0$. For simplicity, we currently also assume that $T_{ckpt} = 0$. When the simulation is run, we configure `LogGOPSim` to co-scheduled detour files on all processors, thereby simulating coordinated checkpoint/restart. We also force each process to start at the beginning of the detour file to ensure proper timing of checkpoints.

Uncoordinated checkpoint/restart with message logging: For uncoordinated checkpointing with message logging, we generate detour files that contain the timestamps and the durations of the local checkpoints. Because no coordination is required, $T_{coord} = 0$. Again for simplicity, we assume that $T_{ckpt} = 0$. For pessimistic message logging [20], we modify the CPU overhead parameter (o in the LogGOPS model) for send operations (o_s) to account for the log write to stable storage. `LogGOPSim` uses a single detour file to simulate the local checkpoints in the system; to model the asynchronous nature of these checkpoints, each process starts at a different location in the file.

Failures: To simulate failures, we generate failure times for each node from a random distribution based on a per-node mean time between failure (MTBF). When a failure is generated, the library adds a detour event that includes the the time required to restart from the last checkpoint and the time required for rework (that is, the time since the last checkpoint). `LogGOPSim` ensures that all communication in the trace file that depends on the failed node will be delayed until the node has "recovered".

D. Simulating Large Spatial and Temporal Scales

For a single collective operation, `LogGOPSim` can simulate up to 10,000,000 processes, and for more general workloads, it is capable of simulating more than 64,000 processes [14]. With some relatively minor modifications [15], primarily to optimize memory consumption, `LogGOPSim` is also capable of simulating long time scales. In Section III, we quantify `LogGOPSim`'s capability to execute simulations of large systems for long times in a tractable and efficient way.

III. Evaluating our Resilience Simulator

In this section, we demonstrate that: (1) our simulator framework produces valid and accurate output for new resilience simulation capabilities, and (2) the framework offers unprecedented performance capabilities for simulating large space and time scales. We now describe the sets of experiments we used to demonstrate these capabilities and the results from these studies. In these studies, we used three principle workloads: LAMMPS, CTH and HPCCG. LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) [21] is a classical molecular dynamics code developed at Sandia National Laboratories. In this paper, we use a Lennard-Jones potential and a SNAP potential as inputs to LAMMPS. CTH is a multi-material, large deformation, strong shock wave, solid mechanics code [22]. CTH is an important U.S. DOE production application. HPCCG is a conjugate gradient benchmark code that is part of the Mantevo [23] suite of mini-apps. In general, our instances of these applications execute in about five to seven minutes of real time.

A. Validating the Simulation Output

One of the first questions that need to be addressed is whether the our trace extrapolation techniques accurately model an application's communication behavior as the system size increases. While we believe our extrapolation likely does not model strong-scaling behavior well, we believe it accurately captures weak-scaling behavior. [16] shows this approach accurately models SWEEP3D. We also verified that for LAMMPS, CTH and HPCCG both the number of MPI operations and the simulated runtime are accurate when compared to *weak* scaling runs on native hardware. In general, our knowledge of how these applications work suggest that our extrapolation method is very appropriate for weak-scaling of our tested workloads.

1) Validating Failure-Free Simulation: In [15], we validated the accuracy of our simulation for failure-free operation using LAMMPS and CTH and compared our framework's output against empirical observations from small-scale (128 nodes) tests using `libhashckpt` [10], a library that supports coordinated and uncoordinated checkpointing on real hardware. We also validated our simulator against a prominent analytical model [2] for larger scales (up to 32,768 nodes). The simulation's predicted execution times were within 1–3% of those from the analytical model and 5–20% of those observed empirically. On the whole, our tests show the simulator is able to accurately model the impact of coordinated and uncoordinated checkpointing in a failure-free environment. More validation details can be found in [15].

2) Validating Simulations with Node Failures: As we described in Section II, we enhanced our framework with the capability to simulate scenarios that include failures and failure recovery. As with failure-free simulations, we validate the simulator's ability to model the impact of failures by comparing against the same validated analytical model [2]. Figure 1 plots the results for simulated and modeled ten hour executions of LAMMPS using a ten minute node restart time, a one second checkpoint commit time and five year node MTBF. We use the optimal checkpoint frequency, 68 seconds, computed using the method in [2]. These results show that our simulator produces highly accurate output compared to the predictions of the analytical model.

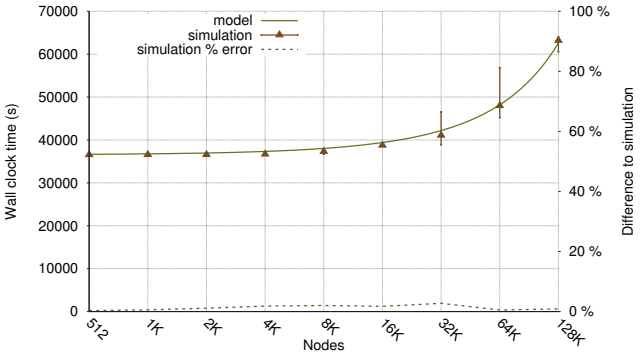


Figure 1. A comparison between efficiency of LAMMPS predicted by the simulator and by a validated analytical model [2] in the presence of failures. The simulator data for each system size represents the range of values produced over 16 runs of the simulator. This simulation models a 10 hour run of LAMMPS using a node restart time of 10 minutes, a checkpoint commit time of 1 second, and a node MTBF of 5 years. Checkpoints are taken every 68 seconds based on the optimal checkpoint interval described in [2].

B. Evaluating Spatial and Temporal Scaling

To evaluate the performance capabilities of our simulation framework, we first demonstrate the simulator's raw performance, then we put this performance in context by showing simulator performance in terms of the speedup or slowdown when comparing the simulator's run time to the simulated execution time of the application at hand.

1) Memory Usage: Figure 2 shows the quantity of memory required to simulate up to 128K (131,072) nodes for each of the three applications. We extrapolate these results to 2M (2,097,152) nodes. These results suggest that for LAMMPS and HPCCG, we could simulate 2M nodes with less than 20 GB of memory. Simulating 2M nodes for CTH would require approximately 100 GB of memory. It is important to note that a system with a relatively modest 10 GB of memory could simulate up to 256K nodes of any of these applications.

As shown in [15], the memory requirements of the simulator are independent of trace length. The simulator requires no additional memory as the temporal scale of the application increases. In other words, for each of the experiments depicted in Figure 2, the same amount of memory would

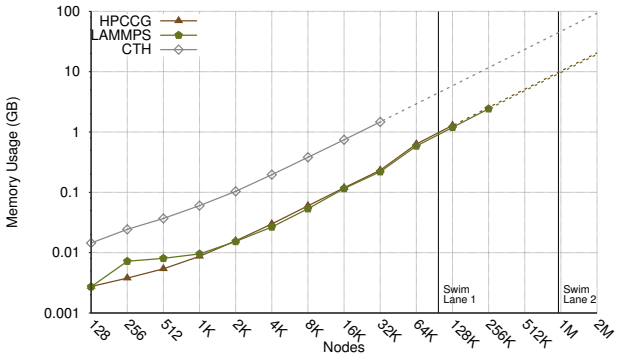


Figure 2. Memory consumption required to simulate a system running one of three applications using our modified version of `LogGOPSim`. This figure is annotated with Swim Lanes 1 and 2 [24] to illustrate the range of system sizes that are currently projected for the first exascale system.

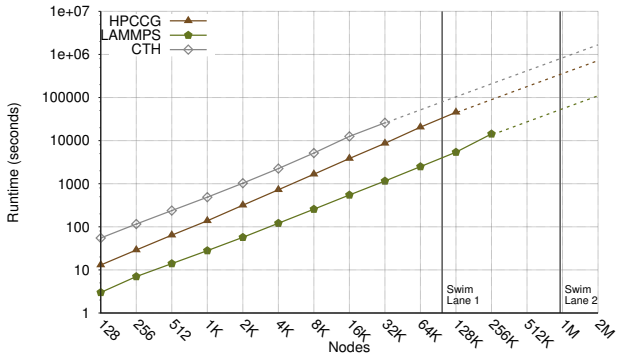


Figure 4. Simulator runtime, measured in seconds (lower is better), for simulating systems running three different applications. This figure is annotated with Swim Lanes 1 and 2 [24] to illustrate the range of system sizes that are currently projected for the first Exascale system.

be required to simulate a minute, hour, day, week or month of application execution time!

2) Simulation Performance: Figure 3 shows the performance of the simulator in terms of the number of events processed per second. We see that for CTH the simulator processes between 200 and 350 thousand events per second. For LAMMPS, the simulator can process between 500 and 800 events per second. Lastly for HPCCG, the simulator can process between 600,000 and 1.3 million events per second. The exact event rate depends on the detailed workload (e.g., mix of collective operations and point-to-point messages as well as the types of collective operations. We examine each applications communication pattern in Section IV). This achieved performance lies well in the range of other highly-tuned simulators like Mambo [25] and gem5 [26].

Figure 4 shows the wall clock time required to simulate three different application instances as a function of system size. In each case, there is a linear relationship between the simulator runtime and the size of the system being simulated. To put this figure in context, consider the largest simulator run time shown: simulating HPCCG on 128K nodes. Simulating HPCCG at this scale required approximately 30,000 seconds. In this simulation, the simulator modeled 7 minutes of application time on 128K nodes. As a result, the simulator required approximately 30,000 seconds to model approximately 55 million node-seconds of simulated execution time.

Finally, Figure 5 shows the performance of the simulator in term of speedup: the ratio of simulated application time to simulator time. This figure demonstrates that the simulator can achieve speedups as large as nearly five orders of magnitude. Moreover, even in the worse case, the simulator achieves speedups of two orders of magnitude. These results corroborate our initial position [15] that coarse-grained simulation can be an effective method for accurately simulating applications on extreme-scale systems.

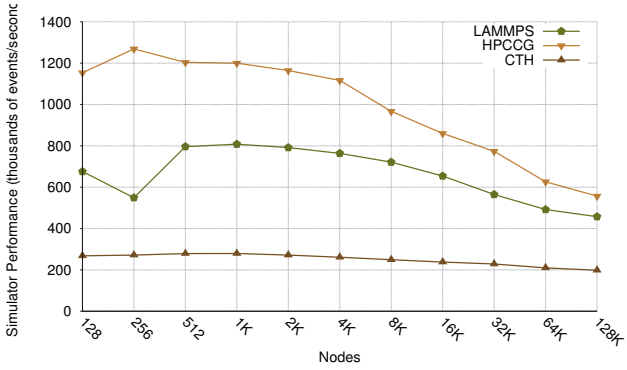


Figure 3. Simulator performance, measured in events/second, when simulating a system running LAMMPS, CTH, and HPCCG.

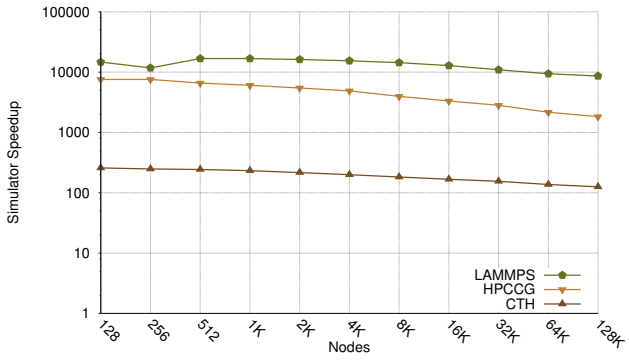
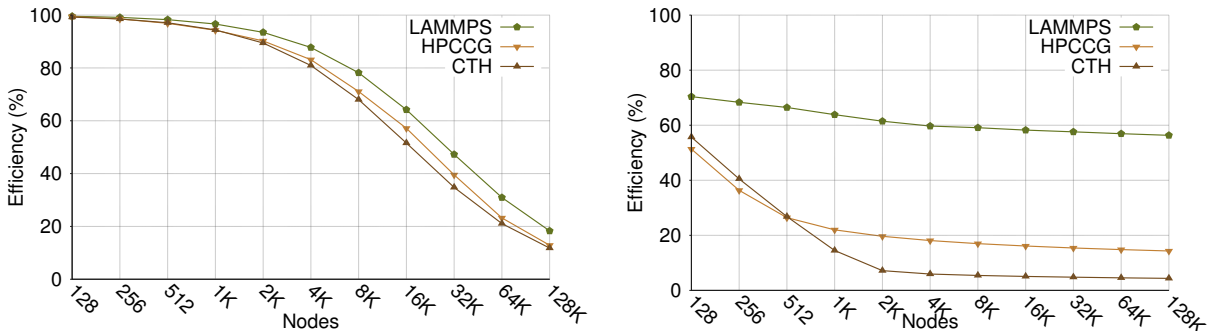


Figure 5. Simulator speedup defined as native application run time in node hours divided by simulation run time for that workload in node hours.



(a) Coordinated

(b) Uncoordinated

Figure 6. Coordinated and Uncoordinated checkpointing efficiency (the percent of time spent performing work for the application and and not the resilience mechanism) using the simulator for CTH and LAMMPS. For both we assume a 120 second checkpoint interval. For uncoordinated checkpointing, we assume each node has a checkpoint commit duration of 1 second, checkpoint times generated independently on each node and the message-logging protocol used to keep all checkpoints consistent is assumed to impose no overhead [13]. For Coordinated checkpointing we assume 2GB of checkpoint data per node and a 256GB/sec aggregate checkpoint bandwidth

IV. Case Study: Performance Impact of Checkpoint/Restart at Scale

We now demonstrate the usefulness of our simulation framework by using it to study the impact of failures and resilience on large scale application performance.

A. Coordinated versus Uncoordinated Checkpointing

Although coordinated checkpointing is currently a most widely used resilience approach, several studies have shown that this approach may not be viable for the next generation of extreme-scale systems [7]–[9]. As a result, researchers have devoted significant effort to identifying alternative approaches. One prominent approach being considered is uncoordinated checkpointing [4], [13]. By eliminating coordination between the processes of an application, this approach reduces contention for persistent storage and reduces the amount of time required to commit a checkpoint.

To demonstrate the capabilities of the modified simulator, we examine the performance of two applications, CTH and LAMMPS, using both of these checkpoint/restart techniques. Figure 6(a) shows the performance impact of coordinated checkpointing. In this figure, we express application performance in terms of efficiency: the percent of total execution time that is used for computation by the application. These results are consistent with results published elsewhere; as the scale of the system increases, more and more time is consumed by writing checkpoints and less and less time is available for the application's computation. By the time the system has grown to 32,768 (32k) nodes, more than half of the execution time is consumed by checkpointing activities.

Figure 6(b) shows results of the same simulations for uncoordinated checkpointing. The first observation is that in one case (LAMMPS) uncoordinated checkpointing generally performs better and in the other cases, coordinated checkpointing does. However, the most striking feature of this figure is the disparity in the performance impact on the two applications. For LAMMPS, its efficiency decreases slowly with scale. For systems larger than 32,768 (32k) nodes, uncoordinated checkpointing outperforms coordinated checkpointing. The response of CTH is significantly different. The fraction of the total run time used for the application's computation decreases rapidly plateauing below 10%. Even at modest scales, uncoordinated checkpointing activities consume a majority of the total execution time for CTH.

B. Application Analysis

Given the starkly different performance impacts of uncoordinated checkpointing on LAMMPS and on CTH and HPCCG, we try to rationalize the disparity. We first consider the time each application spends in communication – as with OS noise, communication events propagate perturbations caused by checkpointing activity disruptions. Figure 7 shows the ratio of each applications communication time computation time as scale increases. LAMMPS spends less than 3% of its time in communication resulting in fewer opportunities for perturbations due to checkpointing activities to propagate. This result is consistent with the trend shown in Figure 6(b); the performance impact of uncoordinated checkpointing is relatively constant even as the system grows significantly in size. In contrast, even at relatively small scales, CTH spends more than 40% of its total execution time communicating. This behavior leads to increased propagation of checkpointing activity disruptions. HPCCG is more complicated. Given its ratio of computation to communication, we would expect it to behave like LAMMPS. However, as Figure 6(b) shows, the impact of uncoordinated checkpointing on its performance is much more similar to CTH.

Although differences in the communication-to-computation ratio help explain the differences between the impact of uncoordinated checkpointing on CTH and

LAMMPS, it does not entirely explain the decrease in application efficiency for CTH shown in Figure 6(b). Nor does it explain the behavior of HPCCG. This suggest that, in addition to the total execution time consumed by communication events, the nature of the communication also matters. For example, checkpointing activities that disrupt a point-to-point communication event may have a smaller total impact on application performance than activities that disrupt collective communications: generally, point-to-point communication events affect a smaller fraction of the processes in the system than collective communication events do. In Figure 8, we examine the communication time for CTH, LAMMPS and HPCCG broken down by communication operation. For LAMMPS, the vast majority of the communication time is consumed by two point-to-point operations: `MPI_Send()` and `MPI_Wait()`. In contrast, CTH devotes more than 30% of its communication time to two collectives: `MPI_Allreduce()` and `MPI_Bcast()`. For HPCCG, a significant majority of its communication is devoted to `MPI_Allreduce()`. Based on this data, our hypothesis is that large numbers of calls to `MPI_Allreduce()` account for the significant negative impact of uncoordinated checkpointing on the efficiency of LAMMPS and HPCCG.

In Figure 9 we test this hypothesis that the slowdown is due to `MPI_Allreduce()`. To do so we use a microbenchmark to measure the performance impact of the various collective operations found in these applications as well as a 3-D stencil communication pattern similar to what is found in CTH. In the figure, we use the same uncoordinated checkpointing pattern as in Figure 6. From the figure we see that `MPI_Allreduce()` and `MPI_Barrier()` see the greatest slowdowns. This is due to the tree-based dependencies that are created in these two collectives. This result combined with the prevalence of `MPI_Allreduce()` in CTH and HPCCG shown previously, indeed point to Allreduce as the source of slowdown.

These results show the power of our simulator. In Figure 6, we simulated the performance impact of a simple uncoordinated checkpointing scheme on three important applications running on very large systems. By examining the communication characteristics of these applications, we were able to develop a hypothesis about the source of the performance impact disparity between LAMMPS and the other two applications: CTH and HPCCG. With a microbenchmark, we then used the simulator to confirm the performance impact of `MPI_Allreduce()` on uncoordinated checkpointing performance.

C. Impact of Failures on Uncoordinated Checkpointing

Thus far, we have examined application performance with two rollback/recovery techniques in a failure free environment. Now we consider the impact with failures. Note, in this section we will only be considering the performance impact with uncoordinated checkpointing. Recall from Figure 1 that an analytical model exists for coordinated and so we will not be considering that here. Also recall from Figure 6, that CTH and HPCCG showed significant slowdowns with uncoordinated checkpointing without failures. Adding failures into this scenario will only further slowdown performance. Therefore, for this study will only be considering LAMMPS uncoordinated performance with failures.

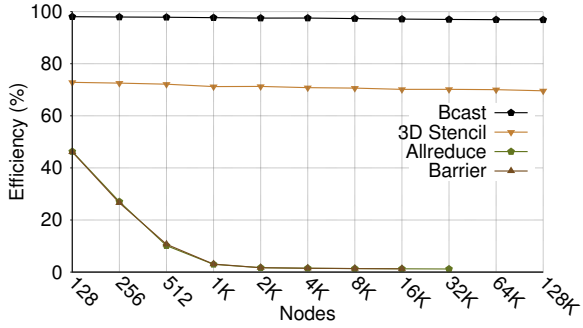
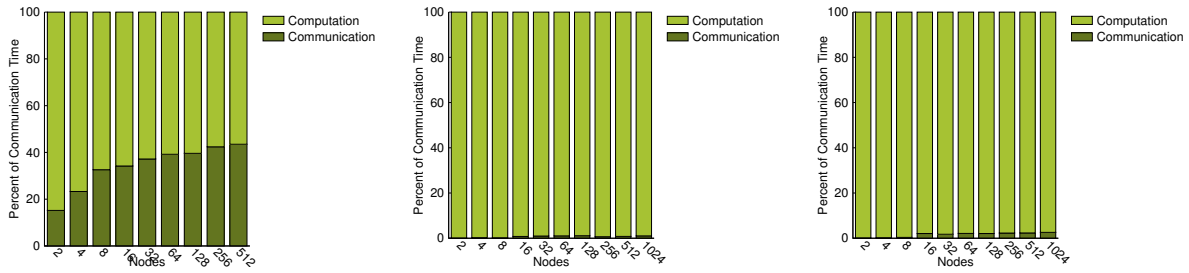


Figure 9. Microbenchmark performance with uncoordinated checkpointing. Similar to Figure 6, a checkpoint is taken every 120 seconds and each checkpoint takes 1 second to complete. Each checkpoint is taken independently and the message logging protocol used to keep checkpoints consistent is assumed to be free. Collective functions see significantly higher slowdowns than the 3-D point-to-point stencil pattern. Note, the Allreduce and Barrier lines are sitting on top of each other in the figure

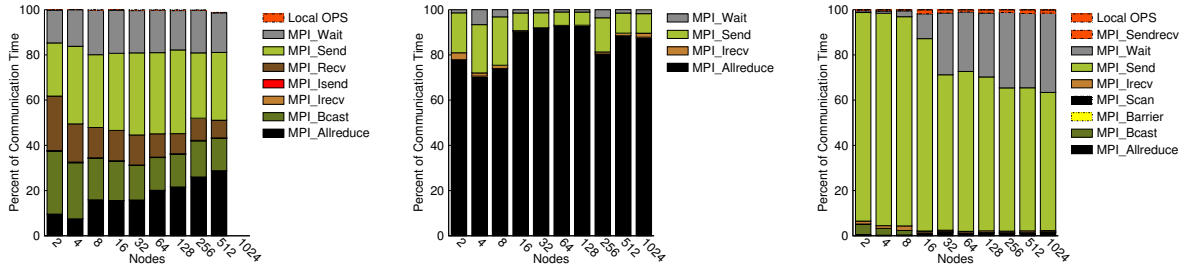
Figure 6 demonstrates that the impact of uncoordinated checkpointing on the performance of LAMMPS is relatively low due to the fact that a small fraction of its execution time is spent in communication. Given this result, our next inquiry was to determine the impact of node failure on its performance. Answering this question required an application trace of LAMMPS that simulated a significant amount of time such that, for a reasonable probability of node failure, failures were likely to occur. Therefore, we used a different LAMMPS potential (SNAP) then we used before in this paper (Lennard-Jones). The SNAP potential is computationally intensive and uses the same kernel as the GAP potential [28]. For the data presented here, we used a configuration that ran for 10.1 hours naively. Finally, because this potential is significantly more computationally intensive than the LJ potential, it has significantly higher efficiency than LJ with uncoordinated checkpointing without failures. This increase is due to the fact that it spends very little of its execution time communicating between nodes.

Figure 10 shows the efficiency of the SNAP potential on 65,536 nodes for a range of node failures rates between one and ten years [29], [30]. Similar to Figure 1 failure times are generated from an exponential distribution and the restart and rework times are modeled as OS noise by the simulator. The restart time for a failed node is again set to 10 minutes and rework is the time since the last local checkpoint. Because we are modeling uncoordinated checkpointing in



(a) CTH (b) HPCCG (c) LAMMPS

Figure 7. Communication/Computation ratios for CTH, HPCCG, and LAMMPS. Data gathered using a Cray XE6 and the mpiP profiling library [27].



(a) CTH (b) HPCCG (c) LAMMPS

Figure 8. Percentage of communication time spent in each MPI function for CTH, HPCCG, and LAMMPS. Data gathered using a Cray XE6 and the mpiP profiling library [27].

this case, a failure on one node only causes the failed node to restart. The surviving nodes continue making progress unless and until they encounter a dependency on communication with the failed node. The simulator properly handles this communication dependency and will delay nodes until the failed node has met the dependency.

The top (green) line in the figure is the application efficiency while taking checkpoint but with no failures. The lower (brown) line is with failures with the specified node MTBF. From this figure we note a number of important points. First, even for very high failure rates (e.g., a one year node MTBF, much higher than has been observed on most current systems) LAMMPS sees relatively high efficiencies on average. As expected, the average efficiency increases as the system's failure rate decreases. Most importantly, we observe that, independent of failure rate, when failures occur they significantly decrease the efficiency of LAMMPS, as noted by the error bars. This suggests that even in the presence of failures, the average efficiency of LAMMPS is quite good. Nonetheless, if high average efficiency is not sufficient, additional mechanisms will be required to avoid occasional low-efficiency runs.

V. Related Work

Though HPC fault tolerance research is a very active area, few tools exist that allow us to project behavior beyond small-scale systems. Such tools need an appropriate level of detail about the communication of the target application.

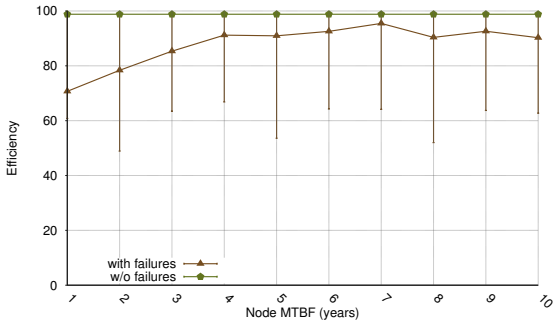


Figure 10. The impact of failures on uncoordinated checkpointing for LAMMPS as a function of Node MTBF. The simulator data for each system size represents the range of values produced over 16 runs of the simulator. This simulation models a 10 hour run of LAMMPS using the SNAP potential, a node restart time of 10 minutes and a checkpoint commit time of 1 second. Checkpoints are taken every 68 seconds based on the optimal checkpoint interval described in [2].

Without sufficient representation of application communication, we cannot accurately simulate some fault tolerance approaches, like uncoordinated checkpoint/restart. Too much details reduce simulator performance with no benefit. Existing application simulators tend to fall to either extreme; either they are not communication-accurate or they simulate communication in greater detail than appears to be necessary and are thus limited in their simulation capabilities.

In [31], Riesen et al. present a simulator that can model the impact of node failure on application performance in the context of traditional coordinated checkpoint/restart.

This simulator can also account for process replication. Tikotekar et al. take a similar approach in [32]. They present a simulator that models coordinated checkpoint and can also simulate fault prediction and process migration. While these tools have been shown to be effective for their stated purposes, they are not communication-accurate. As a result, they cannot account for fault tolerance techniques for which communication patterns influence performance.

At the other extreme is xSim [33]. xSim uses the MPI profiling interface and interposes itself between the application and the MPI library and run unmodified applications. Scaling is achieved by oversubscribing the nodes of the system used for validation. While this provides us with a lot of application performance detail, it imposes a significant cost. Limits on degree of oversubscription necessitate large-scale systems to simulate systems that approach extreme-scale. Moreover, as the size of the simulated system grows and the degree of oversubscription therefore increases, the time required to simulate the system grows dramatically. Lastly, this oversubscription could place significant limits simulated problem sizes as the memory for each simulated node must exist in the memory of one physical node. In contrast, our approach allows us to simulate fault tolerance mechanisms for systems comprised of tens or hundreds of thousands of nodes on very modest hardware (for example, a single node). In some cases, this simulation completes in less time than it would take to run the application itself, but with the less computation details.

Boteanu et al. present a fault tolerance extension to an existing simulator in [34]. However, they target a datacenter environment where each job is a discrete unit that is assigned to a single processing element. As a result, their simulator does not map well to HPC workloads.

Finally, SST/macro [35], [36] is a coarse-grained, lightweight simulator designed to simulate the performance of existing and future large-scale systems. By collecting traces of application execution, SST/macro is able to simulate the application's computation and computation patterns at scales and on hardware that does not yet exist. However, SST/macro does not currently account for the impact CPU detours (OS noise). As a result, because the foundation of our approach is based on the observation that resilience can be modeled as CPU detours, we concluded that SST/macro was not a suitable starting point for our investigation.

VI. Conclusion & Future Work

In this paper, we presented three primary contributions in the area of large scale application simulation in the presence of failures: (1) we validated the functionality of a simulator that allows us to study coordinated and uncoordinated checkpoint/restart protocols with very high accuracy; (2) we demonstrated that our simulation framework allows us unprecedented capabilities for simulating large time scales (10 hours of simulated time) and large space scales (millions of processes) in a very tractable manner; and (3) we used some case studies to demonstrate the usefulness of our simulation framework and the types of insights it enables.

The design space for evaluating resilience methods in large-scale HPC applications is young and still evolving. While our simulation framework has expanded that space in new and useful ways, several areas for future work remain. Among these, we intend to investigate mechanisms to integrate both coarse- and fine-grained simulation for failures. This will allow us to use coarse-grained simulation in areas where failures do not occur, and fine-grained simulation when failures or other interesting events do occur. We also plan to address support for additional resilience mechanisms such as hierarchical checkpointing, replication-based approaches, process migration and cloning, as well as integration with ongoing standards efforts like the current fault tolerance proposal put forth in the MPI forum [37]. Finally, we plan to investigate additional improvements to our framework, including the benefits of parallelization.

References

[1] K. Bergman *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep., Sep. 2008.

[2] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Gener. Comput. Syst.*, vol. 22, no. 3, pp. 303–312, 2006.

[3] M.-S. Bouguerra, T. Gautier, D. Trystram, and J.-M. Vincent, "A flexible checkpoint/restart model in distributed systems," in *Parallel Processing and Applied Mathematics*. Springer, 2010, pp. 206–215.

[4] A. Guermouche, T. Ropars, E. Brunet, M. Snir, and F. Cappello, "Uncoordinated checkpointing without domino effect for send-deterministic MPI applications," in *International Parallel Distributed Processing Symposium (IPDPS)*, May 2011, pp. 989–1000.

[5] L. Alvisi, E. Elnozahy, S. Rao, S. Husain, and A. de Mel, "An analysis of communication induced checkpointing," in *Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on*, 1999, pp. 242–249.

[6] S. Monnet, C. Morin, and R. Badrinath, "A hierarchical checkpointing protocol for parallel applications in cluster federations," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE, 2004, p. 211.

[7] R. A. Oldfield, S. Arunagiri, P. J. Teller, S. Seelam, M. R. Varela, R. Riesen, and P. C. Roth, "Modeling the impact of checkpoints on next-generation systems," in *24th IEEE Conference on Mass Storage Systems and Technologies*, Sep. 2007, pp. 30–46.

[8] K. Ferreira, R. Riesen, P. Bridges, D. Arnold, J. Stearley, J. H. L. III, R. Oldfield, K. Pedretti, and R. Brightwell, "Evaluating the viability of process replication reliability for exascale systems," in *Conference on High Performance Computing Networking, Storage and Analysis, SC 2011, Seattle, WA, USA, November 12-18, 2011*, Nov. 2011.

[9] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," in *Inter-*

national Conference on Dependable Systems and Networks (DSN), Jun. 2006.

[10] K. B. Ferreira, R. Riesen, R. Brightwell, P. G. Bridges, and D. Arnold, "Libhashckpt: Hash-based incremental checkpointing using GPUs," in *Proceedings of the 18th EuroMPI Conference*, Santorini, Greece, September 2011 [to appear].

[11] A. Moody, G. Bronevetsky, K. Mohror, and B. R. de Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '10)*, 2010, pp. 1–11.

[12] D. Ibtesham, D. Arnold, P. G. Bridges, K. B. Ferreira, and R. Brightwell, "On the viability of compression for reducing the overheads of checkpoint/restart-based fault tolerance," *2012 41st International Conference on Parallel Processing*, vol. 0, pp. 148–157, 2012.

[13] A. Guermouche, T. Ropars, M. Snir, and F. Cappello, "HydEE: Failure containment without event logging for large scale send-deterministic mpi applications," in *IPDPS*. IEEE Computer Society, 2012, pp. 1216–1227.

[14] T. Hoefler, T. Schneider, and A. Lumsdaine, "Characterizing the Influence of System Noise on Large-Scale Applications by Simulation," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*, Nov. 2010.

[15] S. Levy, B. Topp, K. B. Ferreira, D. Arnold, T. Hoefler, and P. Widener, "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, to appear. Available at www.cs.unm.edu/˜darnold/Levy2013UsingSimulation.pdf.

[16] T. Hoefler, T. Schneider, and A. Lumsdaine, "LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, Jun. 2010, pp. 597–604.

[17] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "Logp: towards a realistic model of parallel computation," *SIGPLAN Not.*, vol. 28, no. 7, pp. 1–12, Jul. 1993.

[18] T. Hoefler, C. Siebert, and A. Lumsdaine, "Group Operation Assembly Language - A Flexible Way to Express Collective Communication," in *ICPP-2009 - The 38th International Conference on Parallel Processing*. IEEE, Sep. 2009.

[19] K. B. Ferreira, P. Bridges, and R. Brightwell, "Characterizing application sensitivity to os interference using kernel-level noise injection," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, 2008, p. 19.

[20] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Comput. Surv.*, vol. 34, no. 3, p. 375, 2002.

[21] S. J. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal Computation Physics*, vol. 117, pp. 1–19, 1995.

[22] J. E. S. Hertel, R. L. Bell, M. G. Elrick, A. V. Farnsworth, G. I. Kerley, J. M. McGlaun, S. V. PetneY, S. A. Silling, P. A. Taylor, and L. Yarrington, "CTH: A software family for multi-dimensional shock physics analysis," in *Proceedings of the 19th Intl. Symp. on Shock Waves*, Jul. 1993, pp. 377–382.

[23] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, "Improving performance via mini-applications," Sandia National Laboratory, Tech. Rep. SAND2009-5574, 2009.

[24] S. Ahern, A. Shoshani, K.-L. Ma, A. Choudhary, T. Critchlow, S. Klasky, V. Pascucci, J. Ahrens, W. Bethel, H. Childs *et al.*, "Scientific discovery at the exascale: report from the DOE ASCR 2011 workshop on exascale data management, analysis, and visualization," 2011.

[25] P. Bohrer, J. Peterson, M. Elnozahy, R. Rajamony, A. Gheith, R. Rockhold, C. Lefurgy, H. Shafi, T. Nakra, R. Simpson, E. Speight, K. Sudeep, E. Van Hensbergen, and L. Zhang, "Mambo: a full system simulator for the powerpc architecture," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 4, pp. 8–12, Mar. 2004.

[26] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[27] J. Vetter and C. Chambreau, "mpip: Lightweight, scalable mpi profiling," *URL: http://www. llnl. gov/CASC/mpiP*, 2005.

[28] A. Bartók, M. Payne, R. Kondor, and G. Csányi, "Gaussian approximation potentials: the accuracy of quantum mechanics, without the electrons," *Physical review letters*, vol. 104, no. 13, p. 136403, 2010.

[29] B. Schroeder and G. A. Gibson, "Understanding failures in petascale computers," *Journal of Physics: Conference Series*, vol. 78, no. 1, p. 012022, 2007.

[30] G. Gibson, B. Schroeder, and J. Digney, "Failure tolerance in petascale computers," *CTWatch Quarterly*, vol. 3, 2007.

[31] R. Riesen, K. Ferreira, J. Stearley, R. Oldfield, J. H. Laros III, K. Pedretti, R. Brightwell *et al.*, "Redundant computing for exascale systems," Technical report SAND2010-8709, Sandia National Laboratories, Tech. Rep., 2010.

[32] A. Tikotekar, G. Vallée, T. Naughton, S. L. Scott, and C. Leangsuksun, "Evaluation of fault-tolerant policies using simulation," in *Cluster Computing, 2007 IEEE International Conference on*. IEEE, 2007, pp. 303–311.

[33] S. Bohm and C. Engelmann, "xSim: The extreme-scale simulator," in *High Performance Computing and Simulation (HPCS), 2011 International Conference on*. IEEE, 2011, pp. 280–286.

[34] A. Boteanu, C. Dobre, F. Pop, and V. Cristea, "Simulator for fault tolerance in large scale distributed systems," in *Intelligent Computer Communication and Processing (ICCP)*. IEEE, 2010, pp. 443–450.

[35] C. L. Janssen, H. Adalsteinsson, S. Cranford, J. P. Kenny, A. Pinar, D. A. Evensky, and J. Mayo, "A simulator for large-scale parallel computer architectures," *International Journal of Distributed Systems and Technologies (IJDST)*, vol. 1, no. 2, pp. 57–73, 2010.

[36] "Sst: The structural simulation toolkit," http://sst.sandia.gov/about_sstmacro.html, 2011.

[37] W. Bland, A. Bouteiller, T. Herault, J. Hursey, G. Bosilca, and J. J. Dongarra, "An evaluation of user-level failure mitigation support in mpi," in *Recent Advances in the Message Passing Interface*, ser. Lecture Notes in Computer Science, J. L. Träff, S. Benkner, and J. J. Dongarra, Eds. Springer Berlin Heidelberg, 2012, vol. 7490, pp. 193–203.