Supporting the Development of Resilient Message Passing Applications using Simulation

Thomas Naughton, Christian Engelmann, Geoffroy Vallée, and Swen Böhm
Oak Ridge National Laboratory
Computer Science and Mathematics Division
Oak Ridge, TN 37831, USA
Email: {naughtont, engelmannc, valleegr, bohms}@ornl.gov

***Abstract*—An emerging aspect of high-performance computing (HPC) hardware/software co-design is investigating performance under failure. The work in this paper extends the Extreme-scale Simulator (xSim), which was designed for evaluating the performance of message passing interface (MPI) applications on future HPC architectures, with fault-tolerant MPI extensions proposed by the MPI Fault Tolerance Working Group. xSim permits running MPI applications with millions of concurrent MPI ranks, while observing application performance in a simulated extreme-scale system using a lightweight parallel discrete event simulation. The newly added features offer user-level failure mitigation (ULFM) extensions at the simulated MPI layer to support algorithm-based fault tolerance (ABFT). The presented solution permits investigating performance under failure and failure handling of ABFT solutions. The newly enhanced xSim is the very first performance tool that supports ULFM and ABFT.**

Keywords*—*High-performance Computing; Message Passing Interface; Algorithm-based Fault Tolerance ; Parallel Discrete Event Simulation; Performance Prediction;

I. Introduction

Today's large-scale computing platforms are comprised of tens-of-thousands of compute nodes. For example, the IBM BlueGene/Q Sequoia supercomputer at Lawrence Livermore National Laboratory that was fully deployed in June 2012 has 98,304 compute nodes, each with a 16-core processor and 16 GB RAM, totaling 1,572,864 processor cores and 1.57 PB RAM. These supercomputing systems are projected to grow to hundreds-of-thousands and even millions of compute nodes during the next decade on the road to exascale computing.

However, as these machines increase in size and complexity, the expected increase in failure rates must be managed [1]–[8]. They will experience more failures due to an increase in the number of components that can fail. Furthermore, as process technology continues to shrink, the number of transistors affected by a single-event multiple-upset incident grows due to increasing transistor density, resulting in a decrease of individual component reliability. Meanwhile, reliance on commercial-off-the-shelf components with consumer-grade reliability is key to build extreme-scale high-performance computing (HPC) systems in an affordable way.

A number of advanced technologies been developed and/or are currently in development dealing with the issue of resilience in extreme-scale systems, including checkpoint/restart-specific file and storage systems, incremental/differential checkpointing, message logging with uncoordinated checkpointing, fault-tolerant message passing interface (MPI) extensions, containment domains, algorithm-based fault tolerance (ABFT), rejuvenation, reliability-aware scheduling, proactive migration, and redundancy [9]–[22].

The work presented in this paper, particularly targets ABFT supported by fault-tolerant MPI extensions. In general, ABFT seeks to balance performance and resilience from an application perspective. Application programmers often know which data structures are read-only and can be recovered after an MPI process failure from an input file. They also often know which data structures require more extensive recovery, such as through recomputing lost or corrupted data. To permit ABFT the system software layer (the MPI runtime) must enable the application (the MPI job) to continue to run in spite of MPI process failures (partial degradation) and provide feedback about failure events to allow the application to respond accordingly. This not only requires a resilient MPI runtime, but also extensions to the MPI for fault tolerance. In the end, this approach permits an application to determine how best to cope with a failure and balance the benefits of continued execution and recovery/repair.

A standard practice in HPC is to use performance evaluation tools to gain insights into the execution behavior of an application on a given platform. The impact of different platform characteristics can have have profound effects on application performance. As HPC resilience methods emerge to cope with failures, the applications must incorporate the effects of fault-tolerance into their study of execution behavior. This not only targets the issue of performance under failure (efficiency), but also the state transitions during failure handling (correctness). Thus, the performance evaluation infrastructure must provide support for understanding the effects of failures on applications. For the purpose of the presented work, the needed support in performance tools targets fault-tolerant MPI extensions, as well as, fault injection to permit efficiency and correctness evaluation of applications that implement ABFT.

An emerging aspect of HPC hardware/software co-design is investigating performance under failure of MPI applications utilizing ABFT at scale on future HPC architectures and the impact of different HPC architecture choices on performance under failure. For example, network interconnect topology and performance characteristics may have a huge impact on ABFT notification and recovery phases, especially when collective communication is involved. Similarly, an application's recovery from an MPI process failure may offer sufficient performance with 100,000 compute nodes, but may be unusable with 1,000,000 nodes due to unknown scaling limitations

of the ABFT approach, such as exponential message storms. Simulations are often used for estimating MPI application performance on future architectures that haven't been build yet. Highly accurate simulations are typically employed to explore compute node architectures, while less accurate and more scalable solutions are used to study systems at scale.

The work presented in this paper extends the Extreme-scale Simulator (xSim) [23]–[27] performance investigation toolkit, which was designed for evaluation of MPI applications at scale by Oak Ridge National Laboratory, with fault-tolerant MPI extensions that have been developed by the MPI Fault Tolerance Working Group (MPI-FTWG). xSim permits running MPI applications in a controlled environment with millions of concurrent MPI ranks, while observing application performance in a simulated extreme-scale system using a lightweight parallel discrete event simulation (PDES). The newly added features extend prior work on MPI process fault injection in xSim [23] with the user-level failure mitigation (ULFM) [28] extensions that are currently considered as a proposed extension of the MPI standard to provide support for ABFT. The presented solution permits investigating performance under failure, as well as, state transitions during failure handling of ABFT solutions on future-generation HPC systems as part of performance/resilience hardware/software HPC system/application co-design. To our knowledge, the newly enhanced xSim is the very first performance evaluation tool that supports ULFM and ABFT.

This paper is structured as follows. Section II provides background on the MPI-FTWG's proposed enhancements and the xSim toolkit. Section III describes our implementation of the MPI-FTWG's API in the xSim environment, with initial experiments discussed in Section IV. Further related work is discussed in Section V. Section VI offers a summary of the presented work and discusses plans for future work.

II. Background

In this section we provide background information about Algorithm-based Fault Tolerance (ABFT), the related User-Level Failure Mitigation (ULFM) extensions to the MPI standard proposed by the MPI Fault Tolerance Working Group (MPI-FTWG), and the Extreme-scale Simulator (xSim) performance investigation toolkit.

A. Algorithm-based Fault Tolerance

HPC application developers, such as domain scientists, often have very deep insight into the resilience properties of the algorithms they use. There are two separate fault models ABFT is targeting: 1) a fail-stop of an MPI process, and 2) data corruption undetected by other hardware or software layers. This paper targets fail-stop scenarios involving one or more MPI processes. In this failure mode, a MPI process fails to respond for some reason and is eventually declared permanently failed by the MPI runtime. All of its data is lost.

Traditional checkpoint/restart regularly saves data on stable storage to protect it from loss and to recover it after a MPI process failure. With system-level checkpoint/restart, all process data is saved, while in application-level checkpoint/restart the application itself decides which data to save and to restore. In both cases, coordination among all MPI processes is required to maintain consistency in their progress.

ABFT for fail-stop scenarios simply offers more options for the application to react to a failure. It may perform application-level checkpoint/restart, but it also has the option just to recover lost data by maintaining a certain level of data redundancy. It may also be able to simply recompute the lost data or ignore it completely. Examples are a partial differential equation solver that uses recomputation [29], as well as, lossy iterative linear solvers and Eigensolvers [30]. In any case, the application also has the option to continue with less resources when using ABFT, partially degrading its performance but not failing entirely. This property of progress in the presence of failures is the most appealing argument for ABFT, considering the potential overheads for coordinated checkpoint/restart in an extreme-scale HPC system.

The minimum requirements for ABFT are that the programming model (MPI) and its runtime environment support failure detection, notification, and propagation, as well as, failure resilience in the runtime environment. While applications obviously need to be notified about a failure to be able to react to it, there is also a need to optionally propagate this failure notification, i.e., across an entire MPI communicator, to eliminate potential deadlock situations. While recovery is up to the application in ABFT, the MPI runtime environment itself needs to be able to detect and handle MPI process failures to keep the application running with less MPI processes and to notify it of any failures.

B. User-Level Failure Mitigation

ULFM is a set of enhancements to the MPI standard proposed by the MPI-FTWG to provide the basis for ABFT in MPI. An initial prototype showcasing the proposed extensions is available and based on Open MPI (see http://www.fault-tolerance.org). In the following, we briefly describe the ULFM API. A more detailed description is provided in [28].

The default MPI fault behavior forces a job failure upon a process failure. In contrast, ULFM enables application-level handling. It relies on using different MPI communicator error handlers, such as `MPI_ERRORS_RETURN` or user-defined error handlers. The ULFM specification defines six new MPI functions as shown in Listing 1, and two new MPI error types, for process failures and communicator revocation. An MPI communication error due to a failed process returns the error code `MPI_ERR_PROC_FAILED` (or `MPI_ERR_PENDING` for `MPI_ANY_SOURCE` receive requests).

The ULFM API supports failure notification and provides the basis to implement fail-stop fault-tolerance in MPI applications. There are two functions (`MPI_Comm_failure_ack()` and `MPI_Comm_failure_get_acked()`) to acknowledge local failures to quell further notices for known failed ranks in a communicator.

A communicator can be used for point-to-point communications as long as the peer is not a failed rank and the communicator has not been "revoked". Communicator revocation is used to propagate failure information to other members of the group to avoid deadlocks in collective communications. This communicator revocation is initiated through

the MPI_Comm_revoke() function. Once a communicator has been revoked it can no longer be used for communication (collective or point-to-point) and a new handle must be formed using the MPI_Comm_shrink() function, which must be called by all "live" ranks in the communicator. The MPI_Comm_shrink() is analogous to a communicator split with only the "live" ranks included in a newly created communicator. Restated, MPI_Comm_revoke() can be be used to notify other processes about a tainted communicator, while MPI_Comm_shrink() allows to create a new communicator that excludes failed ranks.

Listing 1. The C interface for the ULFM API.

```
1
2  int MPI_Comm_agree (MPI_Comm    comm,
3                      int         *flag);
4
5  int MPI_Comm_iagree (MPI_Comm       comm,
6                       int            *flag,
7                       MPI_Request *request);
8
9  int MPI_Comm_failure_ack (MPI_Comm    comm);
10
11 int MPI_Comm_failure_get_acked (MPI_Comm    comm,
12                                  MPI_Group *failedgrp);
13
14 int MPI_Comm_revoke (MPI_Comm   comm);
15
16 int MPI_Comm_shrink (MPI_Comm    comm,
17                       MPI_Comm   *newcomm);
```

MPI_Comm_agree() facilitates the agreement on a single value at all participating ranks, e.g., to check if an operation was successful on all processes. This distributed consensus mechanism allows for developing application specific failure recovery blocks (failure containment domains) and failure-tolerant collectives (by placing an agree statement at the end of the collective to ensure all return with consistent return codes at additional performance cost [31]). MPI_Comm_iagree() is simply a non-blocking variant of MPI_Comm_agree(), requiring an MPI_wait() for completion.

The ULFM API is minimalistic and avoids any heavy burden on the implementer of an MPI runtime. More extensive recovery schemes can be built atop these 6 functions, such as within libraries and/or applications.

C. Extreme-scale Simulator

xSim is a performance investigation tool for use with MPI applications [23]–[27]. It is implemented as a PDES, which itself is written as a MPI application. The simulator virtualizes the MPI interface and runs the end-user's MPI application as virtual processes. Different network and processor models can be configured to define the performance characteristics of the simulated HPC platform. The virtual processes (virtual MPI ranks) are implemented as user-level threads in xSim, which are distinct from the native MPI processes used to run the PDES. This approach allows for heavy over-subscription of resources to enable extreme scale-up of virtual MPI ranks on current HPC systems, such as running almost 130,000,000 MPI ranks on a Linux cluster with only close to 1,000 physical cores [24]. Despite this over-subscription, the PDES maintains a virtual process clock for each MPI rank based on the processor and network models, thus permitting performance investigation at scale. The simulator prints out the overall execution time of an MPI application running in the simulated HPC system. It also permits access to the virtual MPI process clock at runtime for application-level timing via virtualized MPI_Wtime() and gettimeofday() calls.

In addition to a fully virtualized MPI layer, the simulator also supports a fully virtualized filesystem layer and a memory management interposition layer. The tool has also been extended to support MPI process and job fault injection for resilience investigation [23]. The simulator provides a generic restart mechanism to allow for restarting virtualized applications, and restoring the xSim state. This permits investigating performance under failure, as well as, state transitions during failure handling of checkpoint/restart approaches.

More details about the technical aspects of xSim, such as the PDES implementation details and the network models, can be obtained from the xSim publications [23]–[27].

III. Implementation

We have extended the xSim performance investigation toolkit with the ULFM API, to simulate the ULFM behavior of an MPI runtime and to support ABFT within xSim for investigating performance under failure and state transitions during failure handling. The newly added features will aid in the co-design of future-generation HPC systems and scalable, resilient HPC applications, considering performance and resilience aspects.

A. Simulated MPI Process Failure Injection and Detection

The following outlines the existing features for simulated MPI process failure injection in xSim [23], the newly added ULFM and ABFT support relies on.

Simulated MPI process failures are injected by scheduling them at the targeted simulated MPI process ahead of or at the time of failure. It is activated when the simulated process clock reaches the scheduled time of failure. In this case, the simulated MPI process' execution ends and all messages directed to it are ignored and deleted. A simulator-internal message is broadcast to notify all other simulated MPI processes of the failure and the time of failure. To simplify the injection of simulated MPI process failures by the application itself, ending a simulated MPI process, such as by returning from main() or calling exit(), without calling MPI_Finalize() will be considered as a simulated MPI process failure.

The simulated MPI process failure detection is implemented as a simulated communication failure detector and relies on the simulator-internal message broadcast of a simulated MPI process failure to fail corresponding message send and receive requests. The simulated network communication time of the waiting process is adjusted for the time of failure and the configurable communication timeout. A simulator-internal synchronization mechanism, which assures a conservative PDES with deadlock detection, is used to fail unmatched MPI_ANY_SOURCE receive requests. It also assures that send requests fail at the correct simulated MPI process time.

The default MPI fault behavior, i.e., the default MPI_ERRORS_ARE_FATAL error handler on MPI communicators, forces an MPI job failure through an MPI abort upon a single MPI process failure. xSim simulates this by default

and prints out a message on the command line about the time and rank of the failed MPI process.

B. User-Level Failure Mitigation

To extend the xSim simulator to support ULFM we had to add the API extensions itself and additional tracking information for the virtual processes. This included enhancements to the communicator registries, adding additional message tags to support "out-of-band" communication that honored ULFM semantics, and leveraging the failure notification capabilities outlined in the previous section.

The simulator maintains internal registries for tracking the simulated MPI job's virtualized communicators. These registries are organized to reduce the overall amount of data required for tracking the simulated MPI. We extended these internal registries to support tracking of the revocation of communicators on a per rank/communicator basis. This required the addition of a unique communicator identifier (commID) for internal use during the creation and revocation/shrink of communicators. The commID is a globally unique monotonically increasing value that is managed by the simulator itself, and is used by the simulated MPI whenever a new communicator is created. A new function was added to check if a given communicator had been revoked, at the current simulated time, in the communicator registry. This check is then used throughout the simulator when processing messages to determine if the simulated MPI should return a `MPI_ERR_REVOKED` (or `MPI_ERR_PENDING` when using `MPI_ANY_SOURCE`). The previously described fault-injection functionality supported the process failure notification (i.e., `MPI_ERR_PROC_FAILED`). This formed the basis for supporting `MPI_Comm_revoke()`.

The counterpart to communicator revocation is the `MPI_Comm_shrink()`. This function is implemented using a two-phase commit protocol to establish the fail group in a consistent manner across the "live" members of the communicator that are involved in the shrink operation. The initial phase is used to exchange the set of failed ranks, with the second phase used to ensure consistency for all members. A new group of "live" ranks is created by excluding the agreed upon "failed" ranks from the current (revoked) communicator. This live group is used to create a new communicator (with a new commID). This is all contained within the shrink function and failed processes encountered during the shrink are excluded from the new communicator that is returned by the collective shrink function. The current implementation is based entirely on point-to-point communication and relies on the lower level failure detection/notification to determine failures during these operations, which includes the advancing of simulated time for the underlying PDES. We use internal message tags to differentiate the messages associated with a shrink or agree operation to allow for internal use of the revoked communicator by the virtual MPI process, i.e., to emulate an "out of band" communication mechanism. The communication is governed by the selected networking model and therefore the supported network types can be tested to see their effects when using the ULFM functions.

The `MPI_Comm_agree()` function performs a *"logical and"* of the value provided by all members of the communicator. The function ignores any process failures that occur during the execution of the function, and all living members of the communicator obtain a consistent value (i.e., logical *and* of all their input). The underlying implementation does a reduce and broadcast to distribute the flag in a fault-tolerant manner (i.e., the operations are restarted if they fail due to a process failure during the agreement function). This is effectively a `MPI_Allreduce()` with a `MPI_LAND` operation. As with shrink, a simulator internal tag is used to differentiate the messages for agree to ensure proper operation. The current implementation also uses point-to-point to implement the collectives based on the selected networking model.

We also extended the communicator registries to support tracking of ULFM-based failure acknowledgments. These are implemented as bit arrays on a per communicator basis to indicate which rank(s) have acknowledge failures for that group of ranks. This provides the necessary state tracking to support the `MPI_Comm_failure_ack()` and `MPI_Comm_failure_get_acked()` functions with reasonable memory requirements within the simulator.

IV. Evaluation

Our experiments were run on a Linux cluster ("*SAL9000*") at Oak Ridge National Laboratory. The system configuration includes 40 dual-processor compute nodes with 960 cores in total. The nodes have 2Ghz Intel Xeon 5130 processors with 24 cores per node, 64GB memory and a Nvidia Tesla S2050 GPU. The cluster is connected via a dual-bonded 1Gpbs Ethernet interconnect. The operating system is Ubuntu Linux 12.04 LTS and all tests were run using Open MPI version 1.6.5, which was configured with threading support enabled ('–enable-mpi-thread-multiple') and compiled with debug support.

All tests were run on xSim using three different network model configurations that are representative of HPC environments. The first simulates a Gigabit Ethernet (*GigE*) with 20 microsecond latency (10 μ between node and switch) with a rendezvous threshold of 128 KB. The second network roughly simulates a shared memory (*SHMEM*) communication network with 6 μ latency between cores and 3200 Mbps bandwidth. The third roughly simulates a Cray XT5 3D twisted torus (*3dTT*) network with 7μ latency and 8.8 Gbps bandwidth in all three dimensions.

We have developed a set of small test programs to help experiment with the ULFM API and to test our implementation. The first is a Revoke/Shrink synthetic benchmark that is used to measure the overhead of the `MPI_Comm_revoke()` and the collective operation `MPI_Comm_shrink()` that is used to create a working communicator without failed ranks. This *ft-shrink-barrier* test begins by duplicating the `MPI_COMM_WORLD` handle and sets the default error handler for this new handle to `MPI_ERRORS_RETURN`. The test then performs a collective `MPI_Reduce()` to have all members exchange an integer (e.g., communicator size) and the root who prints this value and all ranks exit. To simulate failures, we have a rank die just before calling the `MPI_Reduce()`. Additionally, all members call `MPI_Barrier()` after the collective to ensure they all stay synchronized and to detect any failures that might not have been detected at the rank during their part of the `MPI_Reduce()`, and the failure gets detected in the `MPI_Barrier()`.

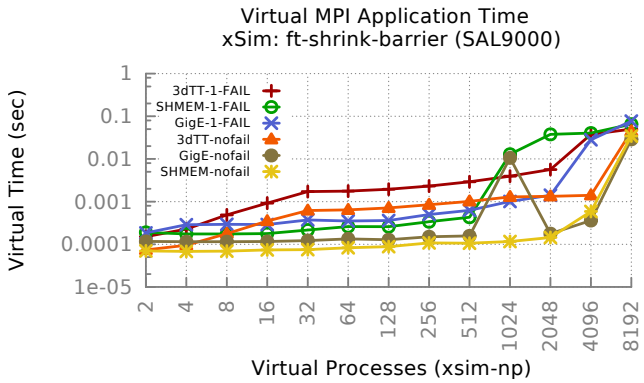

Fig. 1. Shrink/Revoke: Virtual MPI process execution time in seconds for xSim tests running *ft-shrink-barrier* with 3 network models, both with & without a single injected failure.

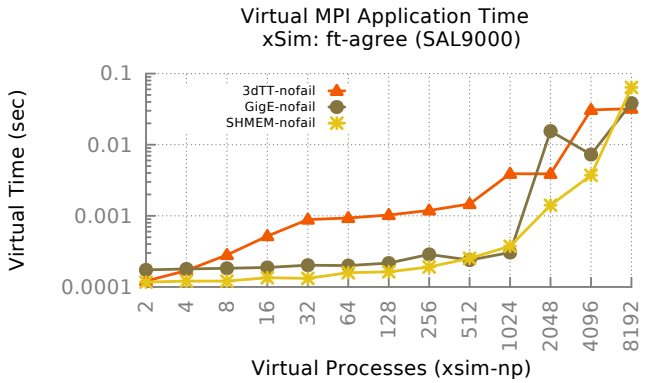

Fig. 2. Agree: Maximum Virtual MPI process execution times in seconds for xSim tests using *ft-agree* with 3 network models without failures.

In Figure 1, we demonstrate the support for using the ULFM functions for Revoke/Shrink on the xSim simulator. The tests show the results of the *ft-shrink-barrier* benchmark at different virtual process (rank) counts with and without an injected process failure. This graph also demonstrates using ULFM with xSim's network models to show the behavior of *ft-shrink-barrier* on the different network configurations.

The maximum virtual MPI process's execution time in seconds is shown in Figure 1. In this graph, only the tests with a simulated failure perform any ULFM functions. Therefore the other lines provide a baseline for comparison to show the overhead for the ULFM Revoke/Shrink and subsequent (successful) Reduce operation.

In the tests, the portion associated with the `MPI_Comm_revoke()` is very small as this is an entirely local operation, and most of the time shown is spent performing the two collective operations: `MPI_Comm_shrink()` and the restarted `MPI_Reduce()`. The overhead for these functions is very small for the simulated time, with shared memory being slightly faster for the lower counts due to lower latency for intra-node communications.

To demonstrate the support for the ULFM agreement function with xSim we wrote a simple unit test called *ft-agree*. This test calls the `MPI_Comm_Agree()` function for a specified number of iterations. Then the root rank collects the times to perform the agreement and reports the average/min/max for the collective ULFM operation. The tests in Figure 2 show the results for running *ft-agree* with an iteration count of 1 (i.e., one call to collective function `MPI_Comm_agree()` for each run of test).

The ULFM agreement function can be used to ensure all callers have a consistent value for an operation. For example, agreement can be used to determine if all callers received a successful return value for a function and act according if they are not consistent. Therefore, the tests with *ft-agree* in Figure 2 show the overhead for the collective agreement function. Since there is no recovery in this unit test there are no failures for this case and the graph shows the overhead for using the consensus function. As we increase the the number of virtual processes the network models begin to show the difference in communication costs for the point-to-point based two-phase commit protocol used in the current implementation of `MPI_Comm_agree()`.

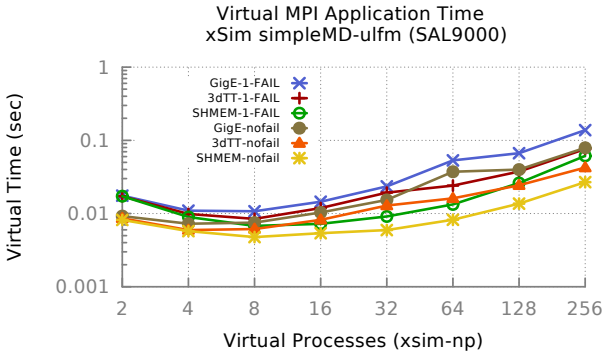

Fig. 3. SimpleMD: Virtual MPI process execution time in seconds for xSim tests using *simpleMD-uflm* with 3 network models, both with & without a single injected failure.

To show the use of the simulator with a full application, we extended a basic demonstration code called *simpleMD* to use ULFM. The application simulates classical molecular dynamics (MD) for a system of N atoms that interact through a Lennard-Jones potential in a closed box with elastic collisions. The application support an optional visualization of the simulation using the Visual Molecular Dynamics (VMD) viewer [32]. The MD application has a parameters file that allows the user to set details about the simulation, which we used to control the number of "steps" (iterations) to perform for the run. The code supports application-level checkpoint/restart with the checkpoint generated in a synchronous manner at defined intervals when the physical properties of the simulation are evaluated. These checkpoints are generated at the start of each new "step", when the intermediate values for the MD properties are displayed to the user and written to the checkpoint file.

The ULFM enhancements to the application included some addition error checking for MPI calls and a forced roll-back to a previous time step when a process failure occurs. The roll-back procedure include the ULFM communicator revocation and shrinkage to obtain a fully function communicator. Then

the application falls back to a previous successful step in the MD simulation (reading from the checkpoint file) and continues the execution. As with the previous examples, we simulated a failure by modifying the *simpleMD-ulfm* code to force a "victim" rank to prematurely terminate, creating an MPI process failure. All failed communications in the simulation will just return the error, and the main simulation loop recognizes ULFM's process failures error types (`MPI_ERR_PROC_FAILED` & `MPI_ERR_REVOKED`). This triggers the revoke, shrink and roll-back to prior checkpointed (saved) timestep. We tested the enhanced version of the simpleMD application with the MPI-FTWG's reference prototype to validate our test before using it under the xSim implementation.

The tests with the ULFM enhanced MD application (*simpleMD-ulfm*) are shown in Figure 3. The graph shows the maximum virtual time (v-tmax) in seconds for the virtual MPI processes. We limit the MD simulation to just four timesteps to reduce the real wallclock time to perform the experiments with xSim. The virtual process times includes the time for the ULFM related communication (revoke and shrink) and the communication related times spent rolling back to the prior checkpoint. The tests were run with the same three network models as those used in the unit tests with a baseline (no failures) and with a single injected failure at the second timestep for each of the simulated network topologies. The pattern for each network type is roughly consistent between the failure and non-failure cases, showing an increase as the virtual process counts increase. The initial dip in the graph is due to parallelism for the application's small problem size, however, the small problem size used for this demonstration preclude problem size scaling studies. The does provide an indication about the general scaling of the application and the effects of the network model. The application maintains a tight synchronization for each timestep of the MD simulation, which is in part due to the way it performs the synchronous application-level checkpointing. These synchronizations are reflected in this graph by the increase in virtual time as process count increases.

This demonstrates xSim's support for a real application scenario with application-level checkpoint restart that is triggered upon process-failure in conjunction with the ULFM support to revoke and shrink the communicator. The graph in Figure 3 provides a rough idea of how the application will behave when failures occur and the application-level fault tolerance is applied as the network topology changes.

V. Related Work

The following discusses further related work in the areas of HPC resilience modeling and simulation, HPC fault injection tools and studies, and HPC modeling and simulation toolkits for performance evaluation/estimation.

The current state-of-practice in HPC to evaluate the performance impact of a resilience solution is modeling and stochastic evaluation. It mostly focuses on 1) assuring hardware reliability and 2) optimizing checkpoint/restart. Assuring hardware reliability is performed by component manufacturers and HPC system integrators using modeling and simulation to assure a certain upper bound on the failure in time (FIT, the number of failures that can be expected in 10^9 hours of operation) rate. This method is tightly integrated in the design process and focuses on cost trade-offs. As explained before, checkpoint/restart is the standard practice for resilience in HPC today and prior efforts focused on optimizing the application of this particular mitigation technique, such as by finding the optimal checkpoint interval [33]. There has been recent work in incremental checkpointing and in redundancy for HPC, which used modeling and simulation to compare these mitigation techniques with checkpoint/restart [9], [20].

Fault injection is an experimental method to investigate the impact of a fault and of a matching mitigation strategy. Fault injection can be used to identify fault propagation, fault detection delay, proper fault handling, and impact in case of unmitigated faults. The field of fault injection in HPC is still not explored well. An initial framework [34] utilized the capabilities of the `ptrace(2)` system call to inject bit flips in the core image and registers of a victim process, and of the fault injection feature in Linux [35] to inject slab, page allocation, and disk I/O errors. Other recent work leveraged process-level redundancy [36] to track propagation of injected data corruption when partially isolating the redundant copies of the execution. One replica was the victim, while the other was the live control. The impact of data corruption in MPI-based applications has also been studied using instrumentation tools, like Pin [37]. One study found that data corruption had noticeable effects on MPI applications 28-71 % of the time [38], including hangs and incorrect output.

There are a few simulation toolkits for estimating the performance of HPC applications on future-generation HPC systems. To our knowledge, none of them support ABFT or ULFM. The Structural Simulation Toolkit (SST) is probably the most advanced solution. SST/micro [39] offers simulation of novel compute-node architectures, including processor, memory, and network. It is a modular PDES framework that utilizes external modeling and simulation tools. SST/-macro [40] is a complementary toolkit that processes output from the MPI tracing library DUMPI for performance evaluation. SST/micro can be used to generate DUMPI traces at smaller-scale, which then can be used as input for SST/macro to extrapolate performance at larger scale. DIMEMAS [41] is a toolkit similar to SST/macro. It processes traces from MPIDTrace obtained from an HPC application run on an existing HPC system and generates output for the performance tools, PARAVER [42] and Vampir [43]. SimGrid [44] and OMNeT++ [45] are multi-purpose simulation toolkits that have also been used to investigate the runtime capabilities of future system architectures.

VI. Summary and Future Work

With this paper, we presented the very first implementation of a simulation-based performance evaluation tool for MPI applications that supports the ULFM extensions. The xSim performance investigation toolkit was extended with ULFM simulation support, such that fault tolerant MPI applications relying on ULFM can be investigated with respect to performance under failure and failure handling correctness. The result is a simulation capability that offers the most important ULFM MPI calls with their defined functionality and their

simulated timing based on xSim's architectural model for simulating a HPC system.

Planned future work includes providing a complete reference implementation of the proposed ULFM MPI extensions, e.g., adding `MPI_Comm_iagree()`. As the ULFM MPI specification has not been ratified by the MPI Forum yet to be part of the MPI standard, any changes to it until ratification will be integrated as well. There are a number of other opportunities in future work, such as the implementation of different algorithms for the ULFM core functions, `MPI_Comm_revoke()`, `MPI_Comm_shrink()`, and `MPI_Comm_agree()`, including a general implementation of fault-tolerant MPI collectives. Future work will also look at investigating the failure handling characteristics of fault-tolerant MPI applications, e.g., the mentioned ABFT solutions for partial differential equation solvers, iterative linear solvers, and Eigensolvers.

ACKNOWLEDGMENTS

Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. De-AC05-00OR22725. This manuscript has been authored by UT-Battelle, LLC, under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

REFERENCES

[1] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A. A. Chien, P. Coteus, N. A. DeBardeleben, P. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, S. Krishnamoorthy, S. Leyffer, D. Liberty, S. Mitra, T. Munson, R. Schreiber, J. Stearley, and E. V. Hensbergen, "Workshop report: Addressing failures in exascale computing," Apr. 2013. [Online]. Available: http://www.christian-engelmann.info/publications/snir13addressing.pdf

[2] J. Daly, B. Harrod, T. Hoang, L. Nowell, B. Adolf, S. Borkar, N. DeBardeleben, M. Elnozahy, M. Heroux, D. Rogers, R. Ross, V. Sarkar, M. Schulz, M. Snir, P. Woodward, R. Aulwes, M. Bancroft, G. Bronevetsky, B. Carlson, A. Geist, M. Hall, J. Hollingsworth, B. Lucas, A. Lumsdaine, T. Macaluso, D. Quinlan, S. Sachs, J. Shalf, T. Smith, J. Stearley, B. Still, and J. Wu, "Inter-Agency Workshop on HPC Resilience at Extreme Scale," Feb. 2012. [Online]. Available: http://institutes.lanl.gov/resilience/docs/InterAgencyResilienceReport.pdf

[3] F. Cappello, A. Geist, B. Gropp, L. V. Kale, W. Kramer, and M. Snir, "Toward exascale resilience," University of Illinois at Urbana-Champaign (UIUC) - Institut National de Recherche en Informatique et en Automatique (INRIA) Joint Laboratory on PetaScale Computing, Tech. Rep. TR-JLPC-09-01, Jun. 2009.

[4] M. Elnozahy, R. Bianchini, T. El-Ghazawi, A. Fox, F. Godfrey, A. Hoisie, K. McKinley, R. Melhem, J. Plank, P. Ranganathan, and J. Simons, "System resilience at extreme scale," Defense Advanced Research Project Agency (DARPA), Tech. Rep., 2008.

[5] A. Geist and R. F. Lucas, "Major computer science challenges at exascale," International Exascale Software Project, Tech. Rep., Feb. 2009, whitepaper. [Online]. Available: http://www.exascale.org/mediawiki/images/8/87/ExascaleSWChallenges-Geist_Lucas.pdf

[6] P. Kogge et al., "ExaScale computing study: Technology challenges in achieving exascale systems," Defense Advanced Research Project Agency (DARPA) Information Processing Techniques Office (IPTO), Tech. Rep., 2008.

[7] B. Schroeder and G. A. Gibson, "Understanding failures in petascale computers," in *Journal of Physics: Proceedings of the Scientific Discovery through Advanced Computing Program (SciDAC) Conference 2007*, vol. 78. Boston, MA, USA: Institute of Physics Publishing, Bristol, UK, Jun. 24-28, 2007, pp. 2022–2032.

[8] N. DeBardeleben, J. Laros, J. T. Daly, S. L. Scott, C. Engelmann, and B. Harrod, "High-end computing resilience: Analysis of issues facing the HEC community and path-forward for research and development," Whitepaper, Dec. 2009.

[9] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann, "Combining partial redundancy and checkpointing for HPC," in *Proceedings of the 32^{nd} International Conference on Distributed Computing Systems (ICDCS) 2012*. Macau, China: IEEE Computer Society, Los Alamitos, CA, USA, Jun. 18-21, 2012.

[10] C. Engelmann and S. Böhm, "Redundant execution of HPC applications with MR-MPI," in *Proceedings of the 10^{th} IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN) 2011*. Innsbruck, Austria: ACTA Press, Calgary, AB, Canada, Feb. 15-17, 2011, pp. 31–38.

[11] C. Engelmann, G. Vallée, T. Naughton, and S. L. Scott, "Proactive fault tolerance using preemptive migration," in *Proceedings of the 17^{th} Euromicro International Conference on Parallel, Distributed, and network-based Processing (PDP) 2009*. Weimar, Germany: IEEE Computer Society, Feb. 18-20, 2009, pp. 252–257.

[12] G. Fagg, E. Gabriel, G. Bosilca, T. Angskun, Z. Chen, J. Pjesivac-grbovic, K. London, and J. Dongarra, "Extending the mpi specification for process fault tolerance on high performance computing systems," in *In Proceeding of International Supercomputer Conference (ICS)*, 2003.

[13] N. Gottumukkala, B. Leangsuksun, N. Taerat, R. Nassar, and S. L. Scott, "Reliability-aware resource allocation in hpc systems," in *Proceedings of the 2007 IEEE International Conference on Cluster Computing*, ser. CLUSTER '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 312–321.

[14] P. H. Hargrove and J. C. Duell, "Berkeley Lab Checkpoint/Restart (BLCR) for Linux clusters," in *Journal of Physics: Proceedings of the Scientific Discovery through Advanced Computing Program (SciDAC) Conference 2006*, vol. 46. Denver, CO, USA: Institute of Physics Publishing, Bristol, UK, Jun. 25-29, 2006, pp. 494–499.

[15] M. Li, S. Vazhkudai, A. Butt, F. Meng, X. Ma, Y. Kim, C. Engelmann, and G. Shipman, "Functional partitioning to optimize end-to-end performance on many-core architectures," in *Proceedings of the 23^{rd} IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC) 2010*. New Orleans, LA, USA: ACM Press, New York, NY, USA, Nov. 13-19, 2010, pp. 1–12.

[16] P. Lemarinier, A. Bouteiller, T. Herault, and G. Krawezik, "Improved message logging versus improved coordinated checkpointing for fault tolerant mpi," in *in IEEE International Conference on Cluster Computing (Cluster 2004). IEEE CS*. Press, 2004.

[17] N. Naksinehaboon, N. Taerat, C. Leangsuksun, C. F. Chandler, and S. L. Scott, "Benefits of software rejuvenation on HPC systems," in *Proceedings of the International Symposium on Parallel and Distributed Processing with Applications*, ser. ISPA '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 499–506.

[18] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott, "A job pause service under LAM/MPI+BLCR for transparent fault tolerance," in *Proceedings of the 21^{st} IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2007*. Long Beach, CA, USA: ACM Press, New York, NY, USA, Mar. 26-30, 2007.

[19] ——, "Proactive process-level live migration in HPC environments," in *Proceedings of the IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC) 2008*. Austin, TX, USA: ACM Press, New York, NY, USA, Nov. 15-21, 2008.

[20] ——, "Hybrid checkpointing for MPI jobs in HPC environments," in *Proceedings of the 16^{th} IEEE International Conference on Parallel and Distributed Systems (ICPADS) 2010*. Shanghai, China: IEEE Computer Society, Los Alamitos, CA, USA, Dec. 8-10, 2010, pp. 524–533.

[21] ——, "Proactive process-level live migration and back migration in

HPC environments," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 72, no. 2, pp. 254–267, Feb. 2012.

[22] J. Dongarra, P. Beckman, and et al., "The international exascale software roadmap," *International Journal of High Performance Computer Applications*, vol. 25, no. 1, 2011. [Online]. Available: http://www.exascale.org/mediawiki/images/2/20/IESP-roadmap.pdf

[23] C. Engelmann and T. Naughton, "Toward a performance/resilience tool for hardware/software co-design of high-performance computing systems," in *(To appear) Proceedings of the 4th International Workshop on Parallel Software Tools and Tool Infrastructures (PSTI)*. IEEE, Oct. 2013.

[24] C. Engelmann, "Scaling to a million cores and beyond: Using light-weight simulation to understand the challenges ahead on the road to exascale," *Future Generation Computer Systems (FGCS)*, 2013, to appear.

[25] S. Böhm and C. Engelmann, "xSim: The extreme-scale simulator," in *Proceedings of the International Conference on High Performance Computing and Simulation (HPCS) 2011*. Istanbul, Turkey: IEEE Computer Society, Los Alamitos, CA, USA, Jul. 4-8, 2011, pp. 280–286.

[26] C. Engelmann and F. Lauer, ""Facilitating co-design for extreme-scale systems through lightweight simulation"," in *Proceedings of the* 12^{th} *IEEE International Conference on Cluster Computing (Cluster) 2010:* 1^{st} *Workshop on Application/Architecture Co-design for Extreme-scale Computing (AACEC)*. Hersonissos, Crete, Greece: IEEE Computer Society, Sep. 20-24, 2010, pp. 1–8.

[27] I. S. Jones and C. Engelmann, "Simulation of large-scale HPC architectures," in *Proceedings of the* 40^{th} *International Conference on Parallel Processing (ICPP) 2011:* 2^{nd} *International Workshop on Parallel Software Tools and Tool Infrastructures (PSTI)*. Taipei, Taiwan: IEEE Computer Society, Los Alamitos, CA, USA, Sep. 13-19, 2011, pp. 447–456.

[28] 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 *Proceedings of the 19th European conference on Recent Advances in the Message Passing Interface*, ser. EuroMPI'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 193–203.

[29] H. Ltaief, E. Gabriel, and M. Garbey, ""Fault tolerant algorithms for heat transfer problems"," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 68, no. 5, pp. 663–677, 2008.

[30] G. Bosilca, Z. Chen, J. Dongarra, and J. Langou, ""Recovery patterns for iterative methods in a parallel unstable environment"," *SIAM Journal on Scientific Computing (SISC)*, vol. 30, no. 1, pp. 102–116, 2007.

[31] J. Hursey and R. Graham, "Preserving collective performance across process failure for a fault tolerant MPI," in *16th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS) held in conjunction with the 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Anchorage, Alaska, May 2011.

[32] "Visual Molecular Dynamics (VMD) project website," (Last viewed: 23jul2013). [Online]. Available: http://www.ks.uiuc.edu/Research/vmd

[33] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Generation Computing Systems (FGCS)*, vol. 22, no. 3, pp. 303–312, 2006.

[34] T. Naughton, W. Bland, G. Vallée, C. Engelmann, and S. L. Scott, "Fault injection framework for system resilience evaluation – Fake faults for finding future failures," in *Proceedings of the* 18^{th} *International Symposium on High Performance Distributed Computing (HPDC) 2009:* 2^{nd} *Workshop on Resiliency in High Performance Computing (Resilience) 2009*. Munich, Germany: ACM Press, New York, NY, USA, Jun. 9, 2009, pp. 23–28.

[35] "Linux fault injection capabilities infrastructure," documentation available at: http://lxr.linux.no/linux/Documentation/fault-injection/.

[36] D. Fiala, F. Mueller, C. Engelmann, K. Ferreira, R. Brightwell, and R. Riesen, "Detection and correction of silent data corruption for large-scale high-performance computing," in *Proceedings of the* 25^{th} *IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC) 2012*. Salt Lake City, UT, USA: ACM Press, New York, NY, USA, Nov. 10-16, 2012, pp. 78:1–78:12.

[37] P. P. Bungale and C.-K. Luk, "PinOS: A programmable framework for whole-system dynamic instrumentation," in *Proceedings of the 3rd International Conference on Virtual Execution Environments (VEE'07)*. New York, NY, USA: ACM, 2007, pp. 137–147.

[38] C. da Lu and D. A. Reed, "Assessing fault sensitivity in mpi applications," in *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2004, p. 37.

[39] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob, "The structural simulation toolkit," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 4, pp. 37–42, Mar. 2011. [Online]. Available: http://doi.acm.org/10.1145/1964218.1964225

[40] 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 Parallel and Distributed System Technology*, vol. 1, no. 2, pp. 57–73, Apr. 2010.

[41] S. Girona, J. Labarta, and R. M. Badia, "Validation of dimemas communication model for MPI collective operations," in *Lecture Notes in Computer Science: Proceedings of the* 7^{th} *European PVM/MPI Users' Group Meeting (EuroPVM/MPI) 2000*, vol. 1908. Balatonfüred, Hungary: Springer Verlag, Berlin, Germany, Sep. 10-13 2000, pp. 39–46.

[42] V. Pillet, J. Labarta, T. Cortes, and S. Girona, "PARAVER: A Tool to Visualize and Analyze Parallel Code," in *Proceedings of WoTUG-18: Transputer and occam Developments*, Mar. 1995, pp. 17–31.

[43] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. S. Müller, and W. E. Nagel, "The vampir performance analysis toolset," in *Tools for High Performance Computing*, M. Resch, R. Keller, V. Himmler, B. Krammer, and A. Schulz, Eds. Springer Berlin Heidelberg, 2008, pp. 139–155.

[44] P.-N. Clauss, M. Stillwell, S. Genaud, F. Suter, H. Casanova, and M. Quinson, "Single Node On-Line Simulation of MPI Applications with SMPI," in *International Parallel & Distributed Processing Symposium*. Anchorange (AK), États-Unis: IEEE, May 2011, rR-7426 RR-7426.

[45] C. Minkenberg and G. R. Herrera, "Trace-driven co-simulation of high-performance computing systems using omnet++," in *OMNeT++ 2009: Proceedings of the 2nd International Workshop on OMNeT++ (hosted by SIMUTools 2009)*. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.