U.S. Department of Energy
Fault Management Workshop

BWI Airport Marriott, Maryland
June 6, 2012

Participants

Al Geist (ORNL)
Marc Snir (ANL)
Eric Roman (LBNL)
Bert Still (LLNL)
Robert Clay (SNL)
Christian Engelmann (ORNL)
Rob Ross (ANL)
Martin Schulz (LLNL)
Sriram Krishnamoorthy (PNNL)
Abhinav Vishnu (PNNL)
Bob Lucas (ISI)
Shekhar Borkar (Intel)
Mootaz Elnozahy (IBM)
Andrew Chien (ANL)
John Wu (LBNL)
Nathan DeBardeleben (LANL)
Larry Kaplan (Cray)
Mike Heroux (SNL)
Lucy Nowell (DOE)
Lee-Ann Talley

Executive Summary

A Department of Energy (DOE) Fault Management Workshop was held on June 6, 2012 at the BWI Airport Marriot hotel in Maryland. The goals of this workshop were to:

1. Describe the required HPC resilience for critical DOE mission needs
2. Detail what HPC resilience research is already being done at the DOE national laboratories and is expected to be done by industry or other groups
3. Determine what fault management research is a priority for DOE's Office of Science and National Nuclear Security Administration (NNSA) over the next five years
4. Develop a roadmap for getting the necessary research accomplished in the timeframe when it will be needed by the large computing facilities across DOE

The workshop was attended by representatives from the DOE national laboratories: Oak Ridge National Laboratory (ORNL), Argonne National Laboratory (ANL), Los Alamos National Laboratory (LANL), Lawrence Berkley National Laboratory (LBNL), Lawrence Livermore National Laboratory (LLNL), Pacific Northwest National Laboratory (PNNL), and Sandia National Laboratories (SNL). It also included vendor representation from IBM, Intel, and Cray.

The resilience requirements for national nuclear security applications, advanced reactor simulations, simulations of materials in extreme conditions, and climate simulations were discussed and it was found that they require different types of resilience. Some require run times of weeks to months. Some require extreme accuracy and bit-wise reproducibility; and most require resilience that works at extreme scale.

Unique at this workshop was a discussion of the many fault management research projects that are already being done at the DOE national laboratories. These efforts include making checkpoint/restart more efficient, fault prediction and avoidance, incorporating resilience into programming models, and algorithm-based fault tolerance. This discussion identified an existing deficiency in understanding the types and rates of faults in current and future HPC systems.

The workshop's recommendations for high priority DOE activities and investments are:
(1) the formation of a Resilience Technical Council with representation from each DOE laboratory and responsible for: identifying research gaps and scope of efforts required, and coordinating the research to ensure integration and compatibility of the solutions.
(2) research and development into fast global and local checkpointing to address DOE's immediate resilience needs and provide an evolutionary path for legacy applications;
(3) research to characterize and quantifying the resilience problem, which includes identifying the types, rates, and causes of faults in future HPC systems and developing a fault model that specifies detection, notification, and recovery options for HW/SW solutions.

Once the problem is characterized, research is needed to eliminate the barriers to resilience solutions including: integrating fault detection and containment throughout the software stack, providing resiliency features and support in programming environments, failure avoidance, and developing resilient algorithms that enable applications to adapt and recover from faults and even silent errors.

1.0 Introduction

Fault management has been identified as a critical need for future HPC systems [Kogge08]. The 1000-fold increase in computational capabilities expected over the next decade, along with incorporation of techniques for reducing energy consumption, are predicted to increase the error rate of the largest systems to a point where present checkpoint/restart methods will no longer be viable. Without research into new fault management techniques and the development of supporting resilience technologies, DOE's mission critical applications may not be able to run to completion, or worse, will complete but get the wrong result due to undetected errors. To determine the DOE mission needs in resilience, and the gaps in present research, a DOE Fault Management Workshop was held June 6, 2012 at the BWI Airport Marriott in Maryland. It was attended by representatives from the DOE National Laboratories: ORNL, ANL, LANL, LBNL, LLNL, PNNL, and SNL. The workshop also included vendor representation from IBM, Intel, and Cray. All attendees are experienced researchers in fault tolerance and resilience, and most of the attendees had also attended a Multi-Agency Resilience Workshop held in Catonsville, Maryland in February 2012.

The intention of this one day follow-up workshop was to build on the work done at the Multi-Agency workshop [Daly12] (which covered the broad needs of multiple agencies) and focus specifically on what are DOE's critical mission needs that will be impacted by resilience, what is the resilience research already conducted by other groups, and what research will need to be done by DOE. The goals of the Fault Management Workshop were to:

1. Describe the required HPC resilience for critical DOE mission needs
2. Detail what HPC resilience research is already being done at the DOE national laboratories and is expected to be done by industry or other groups
3. Determine what fault management research is a priority for DOE's Office of Science and NNSA over the next five years
4. Develop a roadmap for getting the necessary research accomplished in the timeframe when it will be needed by the large computing facilities across DOE

The next two sections detail the DOE mission need for resilience and the present resilience research across the DOE complex. Section 4 discusses the risks, uncertainties, and barriers to more effective fault management. Section 5 describes research opportunities and priorities to address immediate needs, to quantify the resilience problem, and to eliminate barriers to resilience solutions. Section 6 presents a roadmap for resilience over the next decade.

2.0 DOE Mission Need for Resilience

DOE's Office of Science and NNSA have several critical mission deliverables, including annual stockpile certification and safety assurance for NNSA and future energy generation technologies for Office of Science. Computer simulations are key to meeting these deliverables and must be resilient enough to complete in time and correctly to meet the respective critical mission need. A number of representative mission needs and their resilience requirements are given in the following examples.

Resilience Needs in Materials Aging Simulations

Understanding the aging of materials in extreme environments is critical to the design of future nuclear reactors as well as assuring the safety of the nation's nuclear stockpile. Material simulation under extremes of temperature, stresses, and radiation require very complex multi-physics codes that often run for days and weeks on the largest computing systems in the nation. They exhibit a tight coupling between the different parts of the simulation code as it is necessary to capture the complex interactions, like temperature of the material affecting the behavior of the material under stress. Thus faults, if undetected or not contained, can quickly propagate throughout the solution. The scale of these simulations requires hundreds of thousands of processors to provide a solution in a timely fashion. At these scales resilience to permanent and transient faults in hardware and software, as well as resilience to undetected errors, becomes a serious issue. The resilience requirements for materials aging requires that multi-week simulations be able to complete (perhaps with restart) and not have the final result corrupted by faults during the run.

Resilience Needs in Simulations for National Security

National nuclear security applications enable simulations that allow the NNSA to assess and certify the safety, security, and reliability of the nation's nuclear stockpile. Calculations needed to support the mission include many capability simulations requiring high resolution and/or high physics or engineering fidelity. These capability simulations generally require days to weeks to complete, well beyond the expected mean-time-to-interrupt for future computer architectures. Thus, inclusion of fault management and mitigation strategies will be required in the hardware and at all levels of the software stack. In addition, unlike ensembles of calculations in which the general trend of a distribution is the focus, high resolution and high fidelity simulations specifically target understanding the impact of small features and details. For this reason, accuracy and reliability are of paramount importance in these simulations. Small errors can be very significant. Thus, in high fidelity simulations, resilience of the computations and reliability of the answers produced are critically important. This drives a requirement for resilience, reliability, and robustness.

Resilience Needs in Advanced Reactor Simulation

Advanced Reactor simulations involve very large calculations to be solved in "reasonable" times to solution. The requirement for "correctness" is extreme. Codes are validated on the basis of bit-level reproducibility of simulation runs. Validation is required for the results to be trusted for both regulatory and public policy. The pressure for bit-level correctness and for accuracy in the results is extreme. As with many engineering simulations, reactor design simulations require high levels of structural complexity that arise from complex geometries, complex nuclear interactions, and complex materials compositions - the wealth of specific structure that typifies an engineering simulation. This complexity means that algorithms and data structures are less likely to be "naturally resilient", so the software/system/machine will need provide the needed resilience.

In contrast to science where characterizing a trend structure or identifying a new phenomenon might require only approximate modeling or simple boundary conditions, engineering simulations try to characterize specific constants and exact values using precisely measured or constrained geometries and boundary conditions. As such, even seemingly small errors can be of great significance.

Resilience Needs in Climate Simulation

The climate community has traditionally had some of the longest running simulations in HPC. It is not unusual for climate simulations to run for months in order to predict changes in climate over the next century or more. Like advanced reactor simulations, climate simulation codes are validated on the basis of bit-level reproducibility, and hence there is a requirement for bit-level correctness even in long running simulations. This resilience requirement is already a problem today on petascale systems. We have observed weather simulations that produce bit-wise reproducible results 49 out of 50 times, but in a small but significant fraction of the simulations the results are different. We attribute this to the growing probability of getting silent errors on these large systems. With future systems having 100X more memory and at least 10X more nodes, the climate community will be seriously compromised by the growing silent error rates.

Today climate simulation codes use global checkpoint/restart to survive weekly system maintenance and the occasional hardware failure. A future resilience requirement for climate simulations is that very long (months) simulations must be supported and the overhead of the fault management technique (be it checkpointing or something else) cannot take up a significant fraction of the time to solution.

Difference from Data Center Resilience Needs

The resilience requirements for DOE's large-scale computing facilities differ in key ways from those of large-scale commercial data centers, such as Google and Amazon. These differences arise largely from the workload these facilities support. Commercial workloads tend to be composed of a large number of independent tasks; each task requires little or no data from any other task. In contrast, a typical workload for a DOE computer is comprised of a large number of highly-dependent tasks that frequently exchange large amounts data with many other tasks. This tight data dependency requires different approaches to fault management. An error in a calculation on one processor can quickly lead to errors on other processors and total failure of a processor can stop the progress of the entire running application. The standard approaches used by Google applications, such as ignoring the lost information or redundantly calculating the information do not scale for tightly coupled DOE problems, and hence are not sufficient by themselves to manage faults occurring on systems running DOE applications. Another characteristic of DOE's scientific applications, again resulting from their tight data dependencies, is their sensitivity to small fluctuations (jitter) in the execution time of any given task. Due to this sensitivity, most application workloads are run on dedicated processors. Commercial workloads, on the other hand, are often run on time-shared processors. When DOE applications are run on such time-shared processors, they exhibit much poorer performance.

3.0 Present DOE Resilience Research

This section enumerates the many resilience research projects already going on in the DOE labs and those projects funded by DOE outside the labs. The projects are grouped by topic. There is a particularly large amount of exploration around programming models and fast checkpointing techniques.

Holistic frameworks. The Coordinated and Improved Fault Tolerance for High Performance Computing Systems (CIFTS) project at Argonne National Laboratory, Lawrence Berkeley National Laboratory, Oak Ridge National Laboratory, Indiana University, the Ohio State University, and the University of Tennessee, Knoxville, (1) created a fault tolerance backplane specification that allows all levels of the software stack, including libraries, run-time systems, and applications, to exchange fault information and conduct fault management in a coordinated manner [Gupta09], (2) developed a prototype reference implementation of the fault awareness and notification interface specification [Gupta09], and (3) improved fault tolerance capabilities in key libraries and applications [Bouteiller12, Hursey09, Ouyang11, Shet11, Zheng09].

Programming models. Greg Bronevetsky from Lawrence Livermore National Laboratory received a 2011 Presidential Early Career Award for Scientists and Engineers and a 2010 DOE Early Career Award for his research in studying the impact and propagation of hard and soft faults in HPC systems. Mattan Erez from University of Texas, Austin, received a 2012 DOE Early Career Award for his research in resilience-specific programming constructs (Containment Domains) that aim at providing programming model support for fault isolation and state tracking during recovery [Sullivan11]. Containment domains are programming constructs with transactional semantics, which enable the scalable, efficient, and application-aware protection of programs against many types of faults. At its core, a containment domain indicates that all data generated within the domain must be checked for correctness before being communicated outside of the domain. Containment domains are nested and hierarchical, and provide a means to preserve and restore state in an optimal way within the storage hierarchy. In addition, containment domains also provide mechanisms for allowing various forms of error detection to be used, which enable the application-aware, need proportional hardening of programs. This allows algorithmic specific verification to be used in concert with containment domains, in addition to standard hardware based error detection. There is also an ASCR-funded effort involving the University of Chicago, ANL, and HP called Global View Resilience, which shares many of the key ideas of containment domains, but exploits global naming and an open, library-based view. These elements combine flexible programmer control and management with compatibility with irregular, dynamic computational structures.

SNL has developed a programming model concept of selective reliability and an algorithmic strategy to produce new classes of algorithms that can run through soft errors. The first working algorithm is a fault-tolerant GMRES (FTGMRES) linear solver, which can converge in the presence of soft errors. FTGMRES is a two-level algorithm such that the outer level keeps data and performs computation in "highly-reliable" mode and the

inner level works in low-reliable mode where failures including local process failure and soft errors can occur. The majority of computation occurs in the inner level, keeping cost in line, while the outer level assures correct answers and recovers from any failures. Although this approach has been demonstrated for one particular algorithm, much additional work is required to extend it to other algorithm domains, and to develop programming model, runtime system and operating system support, and to determine proper hardware configurations that could support selective reliability.

ANL has explored the use of MPI collectives for the generation of error correcting data under ASCR base funding. In this work [Gropp04] it was shown that MPI collectives (i.e., MPI_Reduce_Scatter) could be used as an efficient method of generating RAID5 style parity layout for synchronized I/O. This approach eliminates many of the overheads associated with client-driven parity calculation, assuming that well-defined synchronization points are available for the computation to occur.

Recent DOE-sponsored efforts at Oak Ridge National Laboratory, Argonne National Laboratory, and the University of Tennessee, Knoxville, also focused on a standardized fault-tolerant MPI specification and implementation [Buntinas12, Hursey11, Hursey12, MPI-FT].

LBNL has pursued system-level checkpoint/restart on modern platforms, by implementing checkpoint/restart for Linux in a package called Berkeley Lab Checkpoint/Restart (BLCR). BLCR provides saves and restores the state of processes running on a Linux system. BLCR cooperates, via callbacks, with a coordination scheme implemented by an MPI library to create coordinated checkpoints when processes span several nodes. Since most existing MPI libraries only perform global coordination, in production BLCR is limited to perform only global checkpoints, i.e. every process must write a checkpoint. This global checkpoint limitation is not an inherent limitation of system-level checkpoints, but rather a result of the primitive sync-and-stop methods adopted by existing production MPI implementations. Several research projects have demonstrated non-blocking or asynchronous checkpoints performed with BLCR, but these techniques have not yet been adopted by the production MPI libraries used on DOE's high-end computers. Similarly, the limitation to MPI is not a BLCR limitation. LBNL and UT Austin, as part of DOE's Degas effort, are collaborating (using the Containment Domains model) to develop small-scale local coordination schemes, and to demonstrate a PGAS-based scheme for resilient execution, which uses checkpoint/restart as part of the recovery scheme.

Pacific Northwest National Lab (PPNL) has explored the design of scalable checkpoint-restart mechanisms for PGAS programming models, specifically Global Arrays. Gioiosa et al. [Gioiosa05] designed efficient kernel-level checkpointing in the Linux kernel that supports triggering of checkpoints through interrupts in as little as 2.5 microseconds. Checkpoints taken as frequently as once per minute were shown to incur overheads less than 6%. Tipparaju et al. [Tipparaju07] augmented Global Arrays with checkpoint-restart support. Subsequent work [Scarpazza07,Villa09] studied automatic identification of global recovery lines in PGAS models using system-level virtualization technologies, to

enable low-overhead virtualization and communication and computation and to provide seamless migration capabilities.

PNNL explored the design of flexible checksum mechanisms for generalized Cartesian distributed multi-dimensional matrices that can co-locate the checksums with the data, while tolerating correlated failures [Ali11CF,Ali12]. The algorithms designed to determine the checksums were shown to be scalable while supporting a variety of fault and distribution constraints. More broadly, PNNL explored the design of fault tolerant data stores. The appropriate choice of redundancy for a data structure depends on its use in a specific context. Ali et al. [Ali11] systematically explored the use of matrix data structures in the key modules NWChem to infer the performance of various fault tolerance schemes. The study showed that a given application module might need different fault tolerance treatment depending on the input size, whether it is used standalone or as part of a another module's calculation, and whether it is capability run or a medium-size run focused on time to solution.

PNNL has designed a fault tolerant communication runtime system for PGAS models using the "continued execution" recovery methodology. The proposed infrastructure is used to design selective replication methodology using read-only and read-write data attributes in PGAS models using Global Arrays as the research vehicle. The proposed interface is being used to design a fault tolerant NWChem. The measured overhead in presence of faults is less than 10%, and less than 5% in absence on faults. Vishnu et al. have recently focused on designing soft error resilient mechanisms for Global array applications such as NWChem and started by performing detailed analysis of the importance of critical data structures, their associated bits and the result of bit flips to the program execution.

Recovering from failures involves detection of global recovery lines and restoring overall execution to a globally consistent state. Pacific Northwest National Lab has explored mechanisms to recover applications to a globally consistent state without rolling back all processes. Ali et al. [Ali11PDP] demonstrated that maintaining redundant application state kept synchronized through duplicated communication is efficient when the application is compute-bound and the communication can be effectively overlapped. The work demonstrated negligible overhead for this approach using key modules in NWChem.

Work stealing is a promising technique to dynamically tolerate variations in the execution environment, including faults, system noise, and energy constraints. Dinan et al. [Dinan10] presented an algorithm for selective restart using a lightweight, distributed task completion tracking mechanism. Compared with conventional checkpoint/restart techniques, this system offers a recovery penalty that is proportional to the degree of failure rather than the system size. Ma and Krishnamoorthy [Ma12] presented the first algorithms for fault tolerant work stealing for task collections operating on global data. The work demonstrated that the overheads (space and time) of the fault tolerance mechanisms are low, the costs incurred due to failures are small, and the overheads decrease with per-process work at scale.

The SciDAC Sustained Performance, Energy and Resilience (SUPER) Institute is pursuing multiple research directions including an integrated, application-level approach to resilience. The approach includes design, development and application of fault injection tools to identify the vulnerability of specific code regions, design and implementation of targeted techniques to reduce that vulnerability as well as language-level extensions to allow the autotuning of the trade-off between performance and resilience. The project has demonstrated that algebraic multigrid (AMG) is naturally resilient and that relatively inexpensive triplication of selected pointers greatly improves its time-to-solution for fault probabilities that are expected in future systems. Further, the project is investigating programming models for resilience at extreme scale as well as compiler techniques to automatically add redundant computations.

Compiler support. In the ESoftCheck project [Yu09], compiler techniques were explored to remove redundant checking for transient and soft errors in executables, assuming that memory and caches are sufficiently protected but that paths in the CPU are not. The approach keeps two copies of each register value and executes operations twice (on different copies), with errors detected by comparison. Optimizations are applied taking into account cases where registers are also hardware protected, where a later check will cover the register directly or indirectly (by checking a different register whose value depends on the first), and where checks inside loops may be moved out of the loop.

Fault prediction, avoidance, and recovery. The RAS for Petascale High-End Computing and Beyond project at ORNL, North Carolina State University, and Louisiana Tech University performed research and development in (1) reliability analysis for identifying pre-fault indicators, predicting failures, modeling and monitoring component and system reliability, and fault injection tools to study impact and propagation within the operating system and runtime environment [Boehm10, Gottumukkala10, Taerat09, Naughton09], (2) proactive fault tolerance technology based on prediction-triggered migration away from components that are about to fail [Engelmann09, Nagarajan07, Wang12a], (3) reactive fault tolerance enhancements, such as incremental checkpointing support [Wang10, Naksinehaboon10] and checkpoint interval/placement adaption to actual and predicted system health threats [Nassar08], and (4) holistic fault tolerance through combination of adaptive proactive and reactive fault tolerance [Tikotekar07].

Soft error susceptability. LANL is studying the soft error susceptibility of ASC applications through fault injection. Some of this work involves construction of new fault injectors [DeBardeleben11] while other portions involve augmenting existing ones from other sources. Regardless, the goal of these experiments is to identify regions of codes that are particularly vulnerable (or even resistant) to data corruption. One study that has been ongoing for over a year involves running a suite of applications on idle nodes to look for silent data corruption. This is done to baseline the rate of soft errors seen on supercomputers. A newer code project at LANL has begun involves rewriting a legacy ASC code into modern software engineering techniques that includes writing the application from the ground up to be aware of faults. This work has had some early successes and demonstrated that for portions of the code certain fault tolerant MPI techniques are beneficial.

In collaboration with the University of Illinois at Urbana-Champaign, LANL has been studying ECC chipkill in an effort to predict when single symbol chipkill will not be sufficient and instead necessitate double symbol chipkill. An analytical model with a detailed reliability analysis has been created which looks out towards an exascale supercomputer and has been verified with a Monte Carlo simulation.

Fast checkpoint techniques. ORNL's Soft-Error Resilience for Future-Generation High-Performance Computing Systems project is doing research and development in (1) HPC checkpoint storage virtualization to improve checkpoint/restart efficiency by aggregating a variety of resources, such as memory, Solid State Disks, and disks [Li10, Wang12b], (2) MPI process-level software redundancy using state-machine replication to eliminate fault handling through rollback/recovery in HPC [Elliott12, Engelmann11], (3) software-based ECC to enhance memory protection from soft errors [Fiala], and (4) soft-error injection tools to study the vulnerability of science applications and of CMOS logic in processors and memory.

ANL and LLNL are further investigating the use of solid state storage as part of the NoLoSS project, also funded as part of ASCR Advanced Architectures. The goal of this project is to explore potential roles and benefits of in-system storage in extreme-scale computational science. This project is exploring two avenues related to resilience. First, the team is developing enhancements to the Scalable Checkpoint/Restart Library (SCR) [Moody10], a tool for the management of checkpoint files on in-system resources (e.g., solid state). These enhancements include compression of checkpoints and techniques for coordination of asynchronous writes to external storage. Second, the team has extended the I/O Forwarding Scalability Layer (IOFSL) [Ali09] to include a write-behind buffering capability. This is one possible approach for incorporating burst buffers into future systems, with the advantage of not requiring modification to the existing parallel file system software.

Reliable storage. An important component of resilience is the detection of errors in data on persistent storage. University of Connecticut and ANL explored the cost of detecting silent data corruption in storage systems [Narayan09] by calculating and storing CRC data alongside each data block in a PVFS [Carns00] storage system. Experiments found that for aligned accesses (i.e., those that modify whole blocks) the overhead of CRC storage and checking was approximately 5% for writing and 22% for reading, while for unaligned access the read-verify-write process required in writing accounted for as much as a 75% overhead over writing without this data security.

4.0 Risks and Resilience Concerns

The strategic risk is not meeting the DOE mission critical needs. The specific risk is that insufficient resilience will lead to application crash, hang, delay, or wrong answer. The attendees expressed concern about the short-term vision in much of the present research, particularly the research involving the improvement of checkpointing techniques. But this evolutionary path is seen as the only way to address the immediate resilience needs of the DOE mission critical applications and the most efficient way to provide a path for legacy

applications. In taking the next steps beyond checkpointing, there was concern that the revolutionary techniques will require an understanding of the actual errors seen on DOE's HPC systems, the rate of these errors, and ideally the most common cause. This knowledge is presently unknown and needs to be researched right away so that some confidence can be applied to extrapolations of error rates of future systems. Attendees pointed out that even when the knowledge of actual errors is known; the lack of a holistic framework or standard APIs for detection, notification and recovery will cause a barrier to the creation of portable fault tolerant application codes.

Providing resiliency features and support in programming environments is a significant barrier to enabling revolutionary application resiliency and fault management. For example, the MPI-3 FT working group is attempting to define the semantics and interfaces required for MPI to survive node failures. While this has led to a concrete proposal that is being actively discussed in the MPI forum, it has not, yet, reached the necessary consensus to make it part of the MPI-3 standard. Further, this addition only focuses on the ability for MPI to continue operation after node failure; it does not address other failure types or recovery and node re-attachment procedures, which would be needed to provide a comprehensive solution. Making fault tolerance features part of the MPI standard is still far from adoption.

Outside of MPI, even less progress has been made in programming model visible fault management capabilities. While some work has been done in ARMCI/GA, in general that model does not see widespread use. The somewhat more common global address space (GAS) languages of Unified Parallel C (UPC) and Fortran with Co-arrays have not at all considered how events like node failures could be represented and communicated. In addition, while MPI resilience to network failures has been demonstrated on some current machines, such resilience for the above GAS languages has additional issues such as how to provide resilient atomic memory operations (AMOs) and how to provide efficient and resilient fine grained communication. Finally, some newer languages such as Chapel also are effectively GAS languages and have similar concerns and need for the definition of interfaces and semantics.

In the future, there is also some expectation that node local failures should also be recoverable, which means that even languages and models only providing local execution and parallelism will also need well defined interfaces and semantics for fault management. OpenMP is a good example of a programming model needing additional work in this area. How effective the handling of node local failures can be will depend in large part on the ability of the HW to isolate the failures. Without such functionality, many (though possibly not all) node local failures will turn into full node failures.

Besides fail-stop behavior, another particularly challenging and concerning area is in the integrity of calculations. Industry trends are indicating that it will become increasingly difficult to be confident in calculations – both from computational elements and data storage elements. The reasons for these trends are varied but effects from terrestrial neutrons, naturally occurring alpha particles, electro-magnetic interference, temperature, and voltage fluctuations are seen today and expected in greater levels in the future.

Usually these faults are transient in nature but permanent faults from these sources are not unheard of.

Today, most DOE applications do not employ sophisticated ways of checking the integrity of their results. Generally a subject matter expert is involved in verifying application outputs but obviously this is challenging and highly impractical. Advances in naturally resilient algorithms, automatic checking applications, algorithm-based fault-tolerance, and tools (particularly source-level language and compiler tools) are badly needed moving forward. Similarly, there is a great need for an understanding of how these faults manifest, propagate, and can be prevented – both in hardware and software.

Current parallel file systems are seen as both unreliable and a performance bottleneck for many applications [Bent09, Lofstead08]. Parallel file systems are a major cause of application interrupts on current-generation systems, and the expected growth in node counts is likely to exacerbate this problem. While parallel file systems can deliver a great majority of the underlying hardware bandwidth for carefully-written synthetic benchmarks, rarely do applications approach hardware speeds, and current parallel file systems rely on expensive, enterprise storage. New designs are needed that leverage lower-cost components, that better tolerate component failures, and that provide integrity guarantees that can be used to support application end-to-end data integrity.

5.0 Research Plan

The workshop attendees identified a number of research opportunities in fault management. The attendees also discussed and endorsed the formation of a "Resilience Technical Council" to facilitate the coordination and integration of different fault management efforts. The technical council would have representation from each of the DOE labs and would be responsible for:

- Identifying the research gaps that need to be addressed,
- Accessing the scope of effort required, and
- Ensuring integration and compatibility of the various solutions.

The rest of this section lays out the opportunities in a timeline fashion. First, we address immediate needs in resilience that require minimal changes to applications. Second, we present the research efforts needed to characterize and quantify the resilience problem. Then once characterized, we describe the opportunities to eliminate barriers to resilient solutions.

5.1 Address Immediate Needs

Currently, global synchronous checkpointing is the standard way of dealing with fail-stop interruptions for most DOE applications. Simply put, this allows applications to rollback state and resume from an earlier application saved snapshot. When considering how viable this approach is there are a few issues to look at: (1) the amount of time it takes to save application state, (2) the expected amount of time useful work can occur before the application faults and needs to recover, (3) and the amount of time it takes to restart from a failure. Historically the rates of failure have been low enough that the amount of work

accomplished between failures was high. However, reports [Daly06] indicate that due to the high component counts of extreme-scale systems and large application states, applications will be unable to perform much useful work due to being in a near-constant state of recovery.

Advances are desperately needed in drastically reducing checkpoint and restart times, drastically increasing expected system MTBF, and breaking the need on global synchronicity of DOE applications. Evolutionary approaches might be viable if all aspects were tackled together. Otherwise, revolutionary solutions will be required. Without these improvements it is highly unlikely that checkpointing will be viable on expected exascale systems. Additionally, advances that entirely remove the need to rollback and recover global system state when small amounts of components fail would be highly beneficial and could be achieved by breaking the need on global synchronicity of checkpointing.

To first approximation, the optimal checkpoint interval is $\sqrt{2 \cdot MTBF \cdot C \bullet kpt}$, where *MTBF* is the mean time between failures and *Chkpt* is the time to take a checkpoint [Young74]. Thus, the fraction of system time lost due to checkpoints and restarts is, approximately $\sqrt{C \bullet kpt/(2 \cdot MTBF)}$. In order to maintain the same efficiency, the checkpoint time has to be reduced in proportion to the reduction in MTBF. A variety of techniques are currently being developed to do so:

1. In memory checkpointing, using RAID techniques [Moody10]. Mirroring is simplest, but leads to a significant increase in memory consumption. Higher RAID levels imply more communication and more computation.
2. Checkpointing using non-volatile RAM (NVRAM). Memory consumption using mirroring is less of an issue. This technique is promising, given the evolution of NVRAM technologies, but NVRAM wear-out is a concern at high checkpoint rates.
3. Asynchronous checkpointing. Coordinated checkpointing causes bursty IO, which is bad for IO performance. Various IO buffering techniques can palliate the IO burstiness. More radically, checkpointing can be combined with message logging, to avoid the need for coordinated checkpointing altogether [Elnozahy 02].

The use of these techniques could reduce checkpoint time by an order of magnitude; and hence accommodate systems with an MTBF of less than an hour as long as undetected errors do not corrupt the result over the course of the entire simulation.

Further research is required to develop these techniques, including:

- Research into hierarchical checkpointing schemes, where in-memory checkpoints are consolidated on local memory (perhaps nonvolatile storage).
- Research into architecture support for remote access to the NVRAM of a failed node. This would greatly facilitate the use of local nonvolatile memory for checkpointing.
- As checkpointing frequency increases, the chance that an error will propagate and corrupt the backup before the error is noticed gets much more likely. Research into better detection and containment are needed.

- With very frequent checkpointing, the probability of errors during checkpointing cannot be ignored, and schemes that can recover from such errors are needed. An alternative would be to be able to trade-off reliability and time-to-solution or energy consumption (e.g., using redundant computations), and execute the checkpoint logic in a reliable mode.

Recovery time, which is ignored in a first order analysis, becomes significant when the checkpoint time is reduced by the above techniques [Daly06]. For example, with an MTBF of 30 minutes, and a checkpoint time of 2 minutes, one would checkpoint every 10 minutes or so, and restart, on average, after three checkpoints. Recovery time includes the time to diagnose the cause of failure, and to allocate resources to replace failed resources. Research is needed to develop faster (and more scalable) approaches for error diagnostics and for dynamic resource management. For example, the OS could continuously provision the runtime with spare nodes. New interfaces between the system monitoring infrastructure and application runtime will be needed to ensure that the run-time is quickly made aware of node failures. The overhead for updating the communication infrastructure (routing tables maintained by the system and communication structures maintained by the MPI library) will need to be significantly reduced.

Avoiding the need for a global restart can reduce fault recovery time. Error containment becomes much more critical in this case to prevent the error from propagating to other nodes. If the state of other nodes is known to be correct, then one only needs to restart the failed node and replay the lost messages from a message log. Such a scheme significantly reduces the overall volume of IO and/or communication, but may not reduce restart time if all processes wait for the failed process to recover. Dynamic load balancing becomes important for such a scheme to work. An additional problem is that, as programming models change, the fixed association between a hardware container (node) and a software container (process) may be lost. More dynamic programming models will require more complex recovery schemes.

5.2 Understand the Resilience Problem

One major issue that makes developing effective, practical fault management difficult is the lack of quantitative data describing the types and rates of faults that occur on present systems, and are expected to occur on future systems. Faults can be permanent (hard), meaning a hardware or software component has failed and must be replaced or rebooted to continue. Faults can be transient (soft), meaning a component may perform incorrectly for an instant then go back to working fine. The least understood are undetected faults, also called silent errors. These can be either permanent or transient. The characteristic is that they are not detected. It is known that today's HPC systems have all three types of errors, but their frequency, root causes, and interdependencies are not well understood.

Most existing analyses address only permanent (hard) errors and assume that (1) components fail at fixed rates often set based on commercial-off-the-shelf failures-in-time (FIT) rates and (2) component failures are independent. Large-scale measurements publishing failure time distributions and component correlations (or fault-tree models) for

HPC systems are scarce. We require empirical studies of existing systems, as well as estimates and models of error rates in future systems, in order to design fault management approaches. In addition to the rate of failures, it is also important to understand the failure modes involved. Performing these measurements on systems in situ requires coordinating the various DOE centers, establishing consistent measurement procedures and definitions, and careful audits to make sure that each site performs the measurements correctly.

Today, the cost/benefit trade-off between the key system design factors of performance, resilience, and power consumption is not very well understood. The only existing model incorporating resilience and performance, targets global synchronous checkpoint/restart. Models for other approaches, such as combining local checkpoint/restart with message logging, algorithm-based fault tolerance, and selective process/task or data redundancy, are missing, including metrics to compare these resilience methods fairly. In addition, there are currently no models and no evaluation methods for identifying the impact on power consumption for any resilience solution, especially concerning the expected high cost of data movement in future HPC architectures. Modeling and simulation tools are needed to understand the tradeoffs and facilitate hardware/software performance, resilience, and power consumption co-design.

Five key fault management research areas must be addressed today to have impact in the long run include: First, faults must be detected, either by hardware, the operating system software, or the application. Second, a notification infrastructure must be in place to send fault information to the necessary components (e.g. the application, job scheduler, and error log). Third, a standard fault model needs to be developed so that fault notification and recovery can be handled uniformly. Although the detected faults may vary widely across systems, many faults (e.g. processor timeouts) are common across a wide class of systems. Fourth, it is also important that we have realistic expectations of how faults may be handled. What are the recovery options? Will some other level of the software stack take care of the recovery? When will it do so and at what efficiency? For example, many fault notifications occur well after the point at which it is simple to deal with them. A simple example is a fault experienced during write-back to main memory; an error experienced during write-back would only occur once there was sufficient cache pressure to evict the line, and the code that wrote the data initially may not have the ability to recalculate it. Forcing all data to be flushed out immediately is one possible answer, but it comes at a tremendous performance cost. We cannot expect users to program around all possible faults, but we should provide a taxonomy of possible solutions for the set of most likely faults. Fifth, DOE researchers need to work together to create a standard fault test suite based on the knowledge gained from activities described in this Section. Scientists could then use such a suite to evaluate different fault management solutions to measure their effectiveness, performance, and robustness.

5.3 Eliminate Barriers to Resilient Solutions

Faults on extreme-scale systems are expected to occur throughout the entire system stack from the hardware all the way up to the application. Some responsibility for detecting faults resides in each of these layers. A major barrier to creating resilience solutions is

that there is no communication or coordination between the layers of the stack in fault detection and management, nor coordination for preventive or corrective actions. The elimination of this barrier requires crosscutting solutions, in particular, a holistic fault tolerance framework to build standardized solutions within. The April 2012 exascale planning workshop [ERC12] began a process of listing possible resilience interfaces between all the different layers of stack. This effort could be leveraged to start the design of a holistic fault tolerance framework.

Future systems may provide complex hardware trade-offs where one can dial the trade-off between the frequency of errors and the efficiency of the computation (in time and energy). This trade-off could be achieved in many different ways, for example:

- High power, high resilience cores, together with low-power, low resilience cores on the same chip
- Cores that can run multiple independent threads, or can run a pair of threads with comparison logic
- Cores that can run with higher voltage threshold for higher reliability or lower threshold, for lower power consumption

Similar performance, resilience, and power consumption tradeoffs may be available for parts of the storage infrastructure, for example using RAID-0, RAID-1, …, RAID-6.

Today's system software has limited fault tolerance and fault awareness. The OS and runtime systems, besides needing to survive faults themselves, need to provide notification and recovery capabilities to affected applications. To facilitate fault recovery, system software needs to be designed to confine errors/faults, to avoid or limit their propagation, and to recover from them when possible.

Fault avoidance techniques aim to predict the imminent failure of components so as to avoid application failures. For example, if node failures can be predicted ahead of time, then a process, task, or thread can be migrated away from a node or core ahead of its failure, avoiding its impact and the need for restart. Such a scheme effectively increases the MTBF. More research is needed to assess the effectiveness of fault avoidance. In addition, one will need efficient support for process/task/thread migration and for the dynamic adaptation of the communication infrastructure.

As discussed in section 4.0, providing appropriate programming model support for fault management is an important step for any application resiliency solution, especially when considering ease of use. Fault management researchers need to work with runtime and programming environment developers to incorporate fault awareness into their respective environments and tools. The efforts of the MPI-3 Fault Tolerance Working Group should be supported and adopted, possibly before full ratification of the standard if that gets mired in concerns, such as ubiquity. Standards discussions should also be started as soon as possible on the interfaces and semantics required by global address space (GAS) programming models, such as Unified Parallel C (UPC) and Fortran with Co-arrays. Support for resilience to network failures for GAS models can largely be hidden from the programmer (except possibly for some performance concerns). However research is still

needed on appropriate methods, likely via a combination of hardware and software, for the most efficient implementation of resilient communication including vanilla transfers, atomic memory operations, and other higher-level communication features, such as active messages. This requires interactions with appropriate standards bodies that govern these languages. OpenMP is another candidate programming model for discussion of fault management interfaces for handling node local faults.

A more revolutionary approach to recovery is to avoid altogether the need for checkpoints. Present DOE applications are neither fault tolerant nor fault aware, They are simply killed and restarted. Research is needed for the development of algorithms that can run through failure, including a mathematically rigorous analysis of what errors can be tolerated. Such schemes have been demonstrated for (dense) linear algebra [Dongarra09, Boley95], where the system matrix can be augmented with new rows that are linearly dependent on the other rows. A related research area is understanding where reliable storage and execution regions can improve the fault management capabilities of algorithms running on unreliable machines. Such schemes have been demonstrated for a few iterative methods – for example, extrapolating lost state from state at neighbor points [Bosilca07, Rodriguez07] and using reliable storage to create a fault tolerant GMRES [Heroux11]. Research is needed to figure out how general such techniques can be.

Functional programming, or any programming model that uses “write-once” variables, provides another view of “checkpoint-less” recovery. Such languages use garbage collection to recycle storage once variables have been consumed (read) by all their consumers. Suppose that old values are not garbage-collected; then one can always restart a computation and recompute lost values, from old preserved values. Checkpointing, namely the copying of current state can be replaced by a progressive “hardening” of old state, concurrent with progress in the computation; this hardening delays garbage collection, but does not impeded progress (as long as storage is available). Recovery from failures can be more localized [Jagannathan91]. While a functional programming style may not be appropriate for large-scale computations, it may be possible to use such “functional techniques” to support more efficient checkpoint and restart schemes. More research will be needed.

6.0 Roadmap

Section 5 presents a number of important activities that DOE needs to invest in to meet mission critical resilience requirements for future systems. This section takes those activities and identifies long lead-time items and when they need to start, what activities are needed in the next couple years and thus must start immediately, and what timeframe the DOE community and DOE mission needs will require solutions to the longer-term resilience issues, such that full applications that can tolerate permanent, transient, and even silent errors, while still getting the correct result. The latter requires support from the hardware, runtime, OS, scheduler, and algorithms inside the applications themselves, thus it will take some time for this resilience infrastructure to be in place.

In the near-term, the Resilience Technical Council first needs to be formed to be able to help define and coordinate the fault management research activities. Also near-term are

the activities specified in Section 5.1 *Address Immediate Needs*. These activities are primarily focused on improving the performance and efficiency of checkpoint and restart techniques, including improving performance and access to checkpoint/restart storage and asynchronous checkpointing.

Also of high priority is understanding the actual types and rates of faults that occur on today's systems, and more importantly what faults are expected on the future extreme-scale systems that DOE will likely procure. Because of the long lead-time and the dependence many other activities have on these results, activities 2-4 in Figure 1 should be started in the near-term. Figures 1 and 2 show the priorities and rough cost estimates for the early start research areas and research areas that can start later. The cost estimates ($, $$, $$$) come from the earlier Catonsville workshop and are translated here.

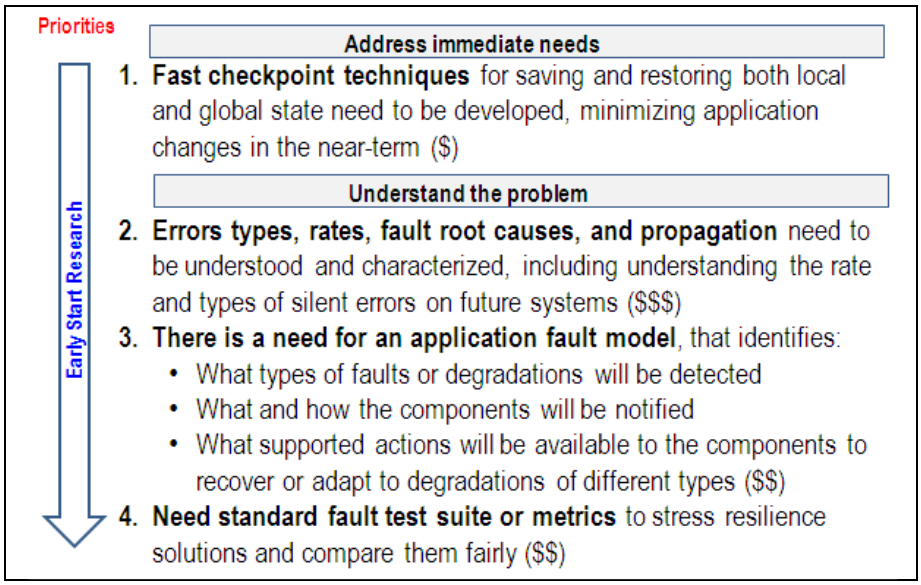

Figure 1. Early start research needed for DOE resilience needs

The better understanding of the resilience problem will allow the creation of a standard fault model and test suite that will allow application developers to begin developing portable, fault tolerant codes. The development of a fault model will allow programming tools to be built that understand transient faults. It will describe services that operating systems, runtimes, and file systems must be modified to supply in order to meet the expectations of the fault model. The fault model will also set expectations for the vendors developing hardware for extreme-scale systems.

Workshop participants identified gaps to the creation of revolutionary resiliency solutions. For example, providing fault management in programming models such as MPI and PGAS, and creation of a fault management framework that spans all layers of the software stack.

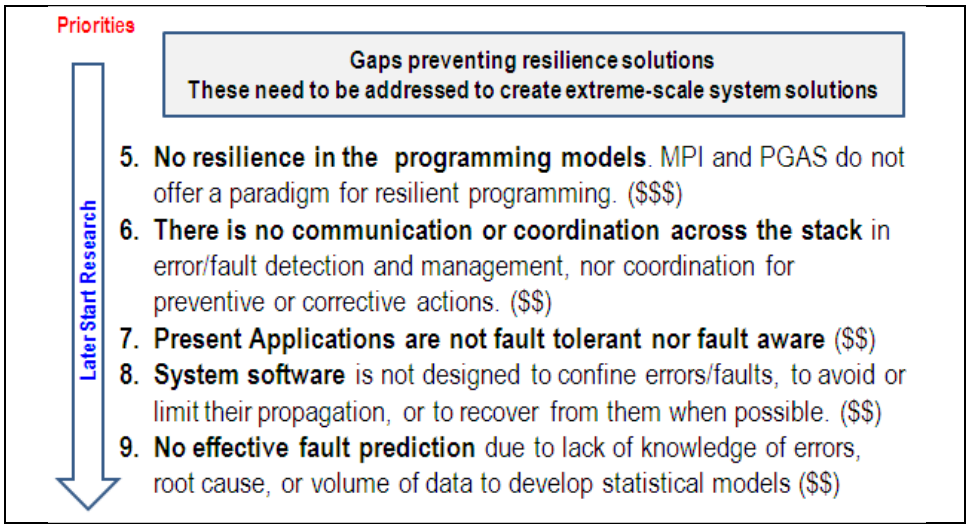

Figure 2. DOE resilience research that is a high priority but can start later

7.0 Conclusions

Unique to this workshop the attendees identified several critical DOE missions in both, the NNSA and Office of Science that will be impacted if computer resilience is not improved beyond what is available today. Also unique was an enumeration of all the fault management research projects going on at each of the DOE national laboratories. The workshop also discussed the risks and barriers to fault management solutions on future systems. The recommended plans going forward included the formation of a resilience technical council, research needed to address the immediate resilience needs of DOE applications, research needed to better understand the types and rates of faults that occur at the hardware, system software, and application levels, and what work is needed to eliminate the barriers to future fault management solutions. A roadmap was constructed to define where long lead-time items need to start, what needs to be addressed immediately and what timeframe the DOE community and DOE mission needs will require solutions to different resilience issues.

References

[Ali09] Nawab Ali, Philip Carns, Kamil Iskra, Dries Kimpe, Sam Lang, Robert Latham, Robert Ross, Lee Ward, and P. Sadayappan. Scalable I/O Forwarding Framework for High-Performance Computing Systems. Proceedings of the 11th IEEE International Conference on Cluster Computing (CLUSTER'09). Tsukuba, Japan. September, 2009.

[Ali11CF] N. Ali, S. Krishnamoorthy, M. Halappanavar, and J. Daily. Tolerating Correlated Failures for Generalized Cartesian Distributions via Bipartite Matching. Proceedings of the ACM International Conference on Computing Frontiers. May 2011.

[Ali12] N. Ali, S. Krishnamoorthy, M. Halappanavar, and J. Daily. Multi-fault Tolerance for Cartesian Data Distributions. International Journal of Parallel Programming, Computing Frontiers special issue (In Press). 2012.

[Ali11] N. Ali, S. Krishnamoorthy, N. Govind, K. Kowalski, and P. Sadayappan. Application-Specific Fault Tolerance via Data Access Characterization. Proceedings of Euro-Par. August 2011

[Ali11PDP] N. Ali, S. Krishnamoorthy, N. Govind, and B. Palmer. A Redundant Communication Approach to Scalable Fault Tolerance in PGAS Programming Models. Proceedings of the Euromicro International Conference on Parallel, Distributed and Network-Based Computing. February 2011.

[Boehm10] S. Boehm, C. Engelmann, and S.L. Scott. Aggregation of Real-Time System Monitoring Data for Analyzing Large-Scale Parallel and Distributed Computing Environments. In Proceedings of the 12th IEEE International Conference on High Performance Computing and Communications (HPCC) 2010, pp. 72-78, Melbourne, Australia, Sep. 1-3, 2010.

[Boley95] Boley, D.; Golub, G.H. ; Makar, S. ; Saxena, N. ; McCluskey, E.J. Floating point fault tolerance with backward error assertions. IEEE Transactions on Computers 44(2), Feb 1995, 302-311.

[Bosilica07] Bosilca, G.; Chen, Z.; Dongarra, J.; Langou, J. Recovery patterns for iterative methods in a parallel unstable environment. 2007.

[Bouteiller12] A. Bouteiller, T. Herault, G. Bosilca and J. Dongarra. Correlated Set Coordination in Fault Tolerant Message Logging Protocols for Many-core Clusters. Journal of Concurrency and Computation: Practice and Experience, 2012. To appear.

[Buntinas12] D. Buntinas. Scalable Distributed Consensus to Support MPI Fault Tolerance. In Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2012, Shanghai, China, May 21-25, 2012.

[Carns00] Philip H. Carns, Walter B. Ligon III, Robert B. Ross, Rajeev Thakur.

PVFS: A Parallel File System for Linux Clusters. Proceedings of the 4th Annual Linux Showcase and Conference. USENIX Association. Atlanta, Ga. October, 2000.

[Daly06] J.T. Daly “A higher order estimate of the optimum checkpoint interval for restart dumps”, Future Generation Computer Systems 22 (2006) 303–312.

[Daly12] J. Daly et al.. Inter-Agency Workshop on HPC Resilience at Extreme Scale. 2012.

[DeBardeleben11] Nathan DeBardeleben, Sean Blanchard, Qiang Guan, Ziming Zhang, Song Fu, “Experimental Framework for Injecting Logic Errors in a Virtual Machine to Profile Applications for Soft Error”, Resilience Proc. of Resilience, the 17th International European Conference on Parallel and Distributed Computing (Euro-Par), September 2011

[Dinan10] J. Dinan, A. Singri, P. Sadayappan, and S. Krishnamoorthy. Selective Recovery From Failures In A Task Parallel Programming Model. Proceedings of the Resilience workshop. May 2010

[Dongarra09] J. Dongarra, George Bosilca, Remi Delmas, and Julien Langou,, Algorithmic Based Fault Tolerance Applied to High Performance Computing, Journal of Parallel and Distributed Computing, Volume 69, pp 410-416, 2009.

[Elliott12] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann. Combining Partial Redundancy and Checkpointing for HPC. In Proceedings of the 32nd International Conference on Distributed Computing Systems (ICDCS) 2012, pp. 615-626, Macau, SAR, China, Jun. 18-21, 2012.

[Elnozahy02] E.N. Elnozahy, L. Alvisi, Y.M. Wangand D.B. Johnson, A survey of rollback-recovery protocols in message-passing systems. ACM Computing Surveys 34, 3, 375—408, 2002.

[Engelmann09] C. Engelmann, G. VallŽe, T. Naughton, and S.L. Scott. Proactive Fault Tolerance Using Preemptive Migration. In Proceedings of the 17th Euromicro International Conference on Parallel, Distributed, and network-based Processing (PDP) 2009, pp. 252-257, Weimar, Germany, Feb. 18-20, 2009.

[Engelmann11] C. Engelmann and S. Boehm. Redundant Execution of HPC Applications with MR-MPI. In Proceedings of the 10th IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN) 2011, pp. 31-38, Innsbruck, Austria, Feb. 15-17, 2011.

[ERC12] Exascale Research Conference in Portland OR. April 16-18, 2012 http://exascaleresearch.labworks.org/apr2012/conference

[Fiala] D. Fiala, K. Ferreira, F. Mueller, and C. Engelmann. A Tunable, Software-based DRAM Error Detection and Correction Library for HPC. In Lecture Notes in Computer Science: Proceedings of the 17th European Conference on Parallel and Distributed Computing (Euro-Par) 2011 Workshops, Part II: 4th Workshop on Resiliency in High Performance Computing (Resilience) in Clusters, Clouds, and Grids, pp. 251-261, Bordeaux, France, Aug. 29 - Sep. 2, 2011.

[Gioiosa05] Roberto Gioiosa, JosŽ Carlos Sancho, Song Jiang, and Fabrizio Petrini. Transparent, Incremental Checkpointing at Kernel Level: a Foundation for Fault Tolerance for Parallel Computers. Proceedings of SC. 2005.

[Gottumukkala10] N.R. Gottumukkala, R. Nassar, M. Paun, C. Leangsuksun, and S.L. Scott. Reliability of a System of k Nodes for High Performance Computing Applications. IEEE Transactions on Reliability, 59(1):162-169, Mar. 2010.

[Gropp04] William Gropp, Robert B. Ross, Neill Miller. Providing Efficient I/O Redundancy in MPI Environments. Proceedings of EuroPVM/MPI 2004. September, 2004.

[Gupta09] R. Gupta, P. Beckman, H. Park, E. Lusk, P. Hargrove," A. Geist, D.K. Panda, A. Lumsdaine and J. Dongarra. CIFTS: A Coordinated infrastructure for Fault-Tolerant Systems. In Proceedings of the International Conference on Parallel Processing (ICPP) 2009, pp. 237-245, Vienna, Austria, Sep. 22-25, 2009.

[Hursey09] J. Hursey, T. Mattox, and A. Lumsdaine. Interconnect Agnostic Checkpoint/Restart in Open MPI. In Proceedings of the 18th ACM international symposium on High Performance Distributed Computing (HPDC), 2009.

[Hursey11] J. Hursey, T. Naughton, G. Vallee, and R. Graham. A Log-Scaling Fault Tolerant Agreement Algorithm for a Fault Tolerant MPI. In Proceedings of 18th

European Parallel Virtual Machine and Message Passing Interface Conference (Euro PVM/MPI) 2011, LCNS vol. 6960, pp. 255-263, Santorini, Greece, Sep. 18-21, 2011.

[Hursey12] J. Hursey and R. Graham. Analyzing Fault Aware Collective Performance in a Process Fault Tolerant MPI. Parallel Computing. Jan., 2012.

[Jagannathan91] Jagannathan, R. and Ashcroft, E.A. Fault tolerance in parallel implementations of functional languages**.** Twenty-First International Symposium on Fault-Tolerant Computing**,** June 1991, 256 – 263.

[Kogge08] P. Kogge et al.. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems, 2008.

[Li10] 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 23rd IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC) 2010, pp. 1-12, New Orleans, LA, USA, Nov. 13-19, 2010.

[Ma12] W. Ma and S. Krishnamoorthy. Data-driven Fault Tolerance for Work Stealing Computations. Proceedings of the International Conference on Supercomputing (ICS). June 2012.

[Moody10] A. Moody, G. Bronevetsky, K. Mohror, and B. de Supinski. Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10). New Orleans, LA. November, 2010.

[MPI-FT] MPI 3.0 Fault Tolerance Working Group at http://meetings.mpi-forum.org/mpi3.0_ft.php.

[Nagarajan07] A.B. Nagarajan, F. Mueller, C. Engelmann, and S.L. Scott. Proactive Fault Tolerance for HPC with Xen Virtualization. In Proceedings of the 21st ACM International Conference on Supercomputing (ICS) 2007, pp. 23-32, Seattle, WA, USA, Jun. 16-20, 2007.

[Narayan09] Sumit Narayan, John Chandy, Samuel Lang, Philip Carns, and Robert Ross. Uncovering Errors: The Cost of Detecting Silent Data Corruption. Proceedings of the Petascale Data Storage Workshop. November, 2009.

[Naksinehaboon10] N. Naksinehaboon, M. Paun, R. Nassar, C. Leangsuksun, S.L. Scott, and N. Taerat. Incremental Checkpoint Schemes for Weibull Failure Distribution. International Journal of Foundations of Computer Science, 21(3):329-344, 2010.

[Nassar08] R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, and S.L. Scott. An optimal checkpoint/restart model for a large scale high performance computing system. In Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2008, pp. 1-9, Miami, Florida USA, Apr. 14-18, 2008.

[Naughton09] T. Naughton, W. Bland, G. VallŽe, C. Engelmann, and S. L. Scott. Fault Injection Framework for System Resilience Evaluation Ð Fake Faults for Finding Future Failures. In Proceedings of the 18th International Symposium on High Performance

Distributed Computing (HPDC) 2009: 2nd Workshop on Resiliency in High Performance Computing (Resilience) 2009, pp. 23-28, Munich, Germany, June 9, 2009.

[Ouyang11] X. Ouyang, R. Rajachandrasekar, X. Besseron and D.K. Panda. High Performance Pipelined Process Migration with RDMA. In Proceedings of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid) 2011, pp. 314-323, Los Angeles (Newport Beach), CA, USA May 23-26, 2011.

[Rodriguez07] Rodriguez, G.; Gonzalez, P.; Martin, M.; Tourino, J. Enhancing fault-tolerance of large-scale MPI scientific applications. Parallel Computing Technologies, 153--161, 2007, Springer.

[Scarpazza07] Daniele Paolo Scarpazza, Patrick Mullaney, Oreste Villa, Fabrizio Petrini, Vinod Tipparaju, D. M. L. Brown, and Jarek Nieplocha. Transparent system-level migration of PGAS applications using Xen on InfiniBand. Proceedings of CLUSTER. December 2007.

[Shet11] A. Shet, W. Elwasif, S. Foley, B-H. Park, D. Bernholdt and R. Bramley. Strategies for Fault Tolerance in Multicomponent Applications. In Proceedings of the International Conference on Computational Science (ICCS) 2011, pp. 2287-2296, Singapore, Jun. 1-3, 2011.

[Sullivan11] M. Sullivan, D.H. Yoon, and M. Erez. Containment Domains: A Full-System Approach to Computational Resiliency. Technical report TR-LPH-2011Ð001, The University of Texas at Austin, January, 2011.

[Taerat09] N. Taerat, N. Naksinehaboon, C. Chandler, J. Elliott, C. Leangsuksun, G. Ostrouchov, S.L. Scott, and C. Engelmann. Blue Gene/L Log Analysis and Time to Interrupt Estimation. In Proceedings of the 4th International Conference on Availability, Reliability and Security (ARES) 2009, pp. 173-180, Fukuoka, Japan, Mar. 16-19, 2009.

[Tikotekar07] A. Tikotekar, G. VallŽe, T. Naughton, S.L. Scott, and C. Leangsuksun. Evaluation of fault-tolerant policies using simulation. In Proceedings of the 9th IEEE International Conference on Cluster Computing (Cluster) 2007, pp. 303-311, Austin, TX, USA, Sep. 17-20, 2007

[Tipparaju07] Vinod Tipparaju, Manojkumar Krishnan, Bruce Palmer, Fabrizio Petrini, and Jarek Nieplocha. Towards Fault Resilient Global Arrays. PARCO. 2007.

[Varma06] J. Varma, C. Wang, F. Mueller, C. Engelmann, and S.L. Scott. Scalable, Fault-Tolerant Membership for MPI Tasks on HPC Systems. In Proceedings of the 20th ACM International Conference on Supercomputing (ICS) 2006, pp. 219-228, Cairns, Australia, Jun. 28-30, 2006.

[Villa09] O. Villa, S. Krishnamoorthy, J. Nieplocha, and D.M. Brown Jr. Scalable transparent checkpoint-restart of global address space applications on virtual machines over infiniband. Proceedings of the ACM International Conference on Computing Frontiers. April 2009.

[Wang10] C. Wang, F. Mueller, C. Engelmann, and S.L. Scott. Hybrid Checkpointing for MPI Jobs in HPC Environments. In Proceedings of the 16th IEEE International Conference on Parallel and Distributed Systems (ICPADS) 2010, pp. 524-533, Shanghai, China, Dec. 8-10, 2010.

[Wang12] C. Wang, F. Mueller, C. Engelmann, and S.L. Scott. Proactive Process-Level Live Migration and Back Migration in HPC Environments. Journal of Parallel and Distributed Computing (JPDC), 72(2):254-267, 2012.

[Wang12b] C. Wang, S. Vazhkudai, X. Ma, F. Meng, Y. Kim, and C. Engelmann. NVMalloc: Exposing an Aggregate SSD Store as a Memory Partition in Extreme-Scale Machines. In Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2012, pp. 957-968, Shanghai, China, May 21-25, 2012.

[Yu09] Jing Yu, Maria Jesus Garzaran, and Marc Snir. ESoftCheck: Removal of Non-vital Checks for Fault Tolerance. Proceedings of the 7th annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO 2009). Seattle, WA. March, 2009.

[Zheng09] Z. Zheng, Z. Lan, B. Park, and A. Geist. System log Pre-processing to Improve Failure Prediction. In Proceedings of the 39th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 572-577, Lisbon, Portugal, June 29 - July 2, 2009.