A PROACTIVE FAULT TOLERANCE FRAMEWORK FOR HIGH-PERFORMANCE COMPUTING

Antonina Litvinova
Department of Computer Science
The University of Reading, Reading, UK
email: a.litvinova@student.reading.ac.uk

Christian Engelmann and Stephen L. Scott
Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge, TN, USA
email: engelmannc@ornl.gov and scottsl@ornl.gov

ABSTRACT
As high-performance computing (HPC) systems continue to increase in scale, their mean-time to interrupt decreases respectively. The current state of practice for fault tolerance (FT) is checkpoint/restart. However, with increasing error rates, increasing aggregate memory and not proportionally increasing I/O capabilities, it is becoming less efficient. Proactive FT avoids experiencing failures through preventative measures, such as by migrating application parts away from nodes that are "about to fail". This paper presents a proactive FT framework that performs environmental monitoring, event logging, parallel job monitoring and resource monitoring to analyze HPC system reliability and to perform FT through such preventative actions.

KEY WORDS
high-performance computing, fault tolerance, system monitoring, high availability, reliability

1 Introduction

As high-performance computing (HPC) systems continue to increase in scale, their mean-time to interrupt (MTTI) decreases respectively. There are currently 4 systems with 130,000-164,000 processor cores, 1 with 213,000 and 1 with 295,000 [18]. In each of these supercomputers, a compute node consists of 2-8 cores, 1-4 memory modules and 1 network interface. The total component count easily approaches 1,000,000. Next-generation systems are expected to have 10-100 times of that [11]. Meanwhile, the reliability of current-generation systems has been lower in comparison to their predecessors (Table 1). The current state of practice for fault tolerance (FT) in HPC is checkpoint/restart via a parallel file system [10]. However, with increasing error rates, increasing aggregate memory and not proportionally increasing I/O capabilities, it is becoming less efficient. A recent investigation [8] revealed that the efficiency, *i.e.*, the ratio of useful vs. scheduled machine time, can be as high as 85% and as low as 55%.

Year	System	Cores	MTTI	Source
2000	LLNL ASCI White	8,192	40.0h	[23]
2002	NERSC Seaborg	6,656	351.0h	[21]
2007	LLNL ASC BG/L	212,992	6.9h	pers. comm.

Table 1. Past and current HPC system reliability statistics

This paper describes our efforts in providing high-level reliability, availability and serviceability (RAS) for HPC environments using a standardized core of scalable technologies. Since each system has distinct dependability properties and architectural features, we target a framework that supports different RAS technologies to be used individually or in combination for providing a comprehensive solution. A previously proposed framework [22] offered innovative proactive fault-handling techniques through fault prediction, detection and avoidance [20, 29], while enhancing reactive fault recovery through adaptive and incremental checkpoint approaches [7, 14]. This paper presents an implementation of the previously proposed framework. It focuses on the proactive FT approach, which avoids experiencing failures through preventative measures. The framework performs environmental monitoring, event logging, parallel job monitoring and resource monitoring to analyze HPC system reliability and to permit fault avoidance.

2 Related Work

Proactive Fault Tolerance Proactive FT keeps applications alive by avoiding failures through preventative measures [12]. In contrast, reactive FT performs recovery from experienced failures. Both are complementary resilience techniques. Proactive FT using migration prevents compute node failures from impacting parallel applications by migrating application parts (tasks, processes, or virtual machines) away from nodes that are "about to fail". Indications of an imminent compute node failure, such as a fan fault or an unusual sensor reading, are used to avoid failure through anticipation and reconfiguration. As applications continue to run, their application mean-time to failure (AMTTF) is extended beyond system mean-time to failure (SMTTF). Since fault avoidance has significantly less overhead than rollback recovery, proactive FT offers more efficient resilience against *predictable* failures.

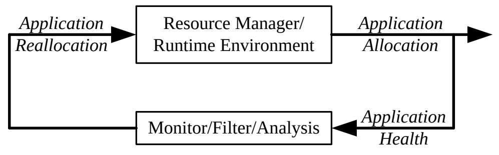

Figure 1. Feedback-loop control of proactive FT [12]

Proactive FT relies on a feedback-loop control (Figure 1) [12] with continuous health monitoring, data analysis and application reallocation. There are four distinct types based on the feedback-loop control capabilities. *Type 1* [12], the most basic form, uses sensor threshold triggers on compute nodes to notify the job and resource management system that a failure is imminent and a node needs to be evicted. It provides an alert-driven coverage for basic failures only as there is no evaluation of application health history or context. It is prone to false positives/negatives, to miss the real-time window for avoiding failures and to decrease application heath through migration to unhealthy nodes. *Type 2* [12], an enhanced form of Type 1, uses sensor data filters on compute nodes to identify trends and to notify the job and resource management system. Since it offers a trend-driven coverage of basic failures, this type is less prone to false positives/negatives. However, it is still prone to miss the real-time window and to decrease application heath through migration. *Type 3* [12], an advanced form, centers around a system-wide reliability analysis using monitoring trend data from all compute nodes. This reliability-driven coverage of basic and correlated failures is much less prone to false positives/negatives, is able to maintain the realtime window, does not decrease application heath through migration and correlates short-term application health context and history. Finally, *Type 4* (Figure 2) [12], the most advanced form, extends Type 3 by adding a history database for monitoring and reliability analysis data. In addition to the features of Type 3, it is able to match system and application patterns by correlating long-term application health context and history. The work presented in this paper focuses on Type 4.

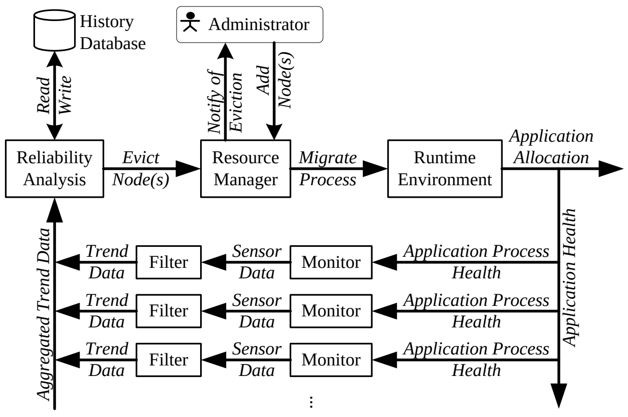

Figure 2. Type 4 feedback-loop control [12]

Environmental Monitoring and Analysis OpenIPMI (http://openipmi.sourceforge.net) enables access to Intelligent Platform Management Interface (IPMI) data, such as processor temperatures and fan speeds. Ganglia [17] is a scalable distributed monitoring system, where each node monitors itself and disseminates monitoring data, *e.g.*, IPMI data, to other nodes. Ganglia and OpenIPMI have already been used in Type 1 solutions [20, 29]. OVIS 2 [2] collects system health information directly from nodes or from other monitoring solutions, *e.g.*, Ganglia, for statistical processing. OVIS 2 provides Type 3/4 online analysis as well as Type 4 offline analysis using a history database. It has not yet been used in proactive FT feedback control loops. A few other solutions exist, such as the RAS systems deployed by HPC vendors. They are similar in design and none of them have been used for proactive FT.

Event Logging and Analysis Analyzing failure patterns using statistical methods has been a recent effort in HPC. Most work relies on system logs from the USENIX Computer Failure Data Repository at http://cfdr.usenix.org. Recent work includes a failure prediction framework [13] that explores log entry correlations. The results show a more than 76% accuracy in offline prediction (needed for Type 4) and more than 70% accuracy in online prediction (needed for Type 3/4). Another effort [24] offered a classification scheme for syslog messages, a Type 4 offline analysis. It is able to localize 50% of faults with 75% precision, corresponding to an excellent false positive rate of 0.05%. Other work in event logging and prediction exist, focusing on identification of root causes, failure modes, trends, correlations, patterns, failure indications and future threads.

Job and Resource Monitoring Monitoring parallel application and compute-node state is typically performed by the job and resource manager of a HPC system, such as Torque [5], which also often interacts with an accounting service to log system usage and failure events.

Migration Mechanisms Two recent Type 1 prototypes used (1) Xen's virtual machine (VM) migration [20] and (2) a newly developed process migration mechanism for BLCR [29]. While VM migration takes 13-24s, process migration is faster (1-6.5s) as less state is transferred. Another effort targeted transparent MPI task migration using the Charm++ middleware and its Adaptive MPI [3]. This work focused on the migration aspect only and did not provide the feedback-loop control. MPI-Mitten [9] provides a library between MPI and applications for transparent migration support. Other migration mechanisms exist, however, none of them efficiently support parallel applications.

Proactive Fault Tolerance Frameworks Two Type 1 prototypes exist [20, 29] that perform migration at the (1) VM or (2) process level. Both utilize Ganglia and a load balancer for selecting the migration target. A third Type 1 prototype [28] is a framework that was developed to investigate coordination, protocols and interfaces between individual system components.

Fault Tolerance Policies Recent work in HPC fault tolerance policies utilized simulation to evaluate trade-off models for combining migration with checkpoint/restart [26]. Using failure logs, the impact of prediction accuracy was put in context with restart counts and checkpoint frequency. The results show that this holistic approach offers the best resilience as proactive FT is handling *predictable* failures, while reactive FT continues to handle *unpredictable* ones.

3 Technical Approach

The previously proposed HPC RAS framework (Figure 3) [22] coordinates individual FT solutions and offers a modular approach for adaptation. At its center, a highly available RAS engine running on redundant service nodes processes current and historical environmental monitoring and event log data from compute nodes. While a node-local pre-processing provides scalability, a central system-wide analysis offers optimal filtering for coordinated triggering of FT mechanisms according to FT policies. The data collection and processing, the execution of FT mechanisms and respective changes in application health form a constantly optimized feedback-loop control.

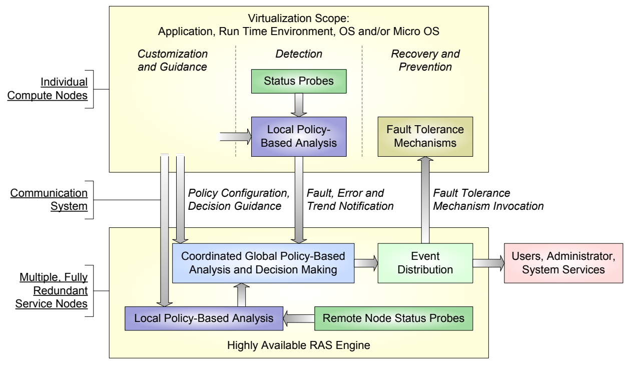
Virtualization Scope:
Application, Run Time Environment, OS and/or Micro OS
Customization and Guidance
Detection
Recovery and Prevention
Individual Compute Nodes
Status Probes
Local Policy-Based Analysis
Fault Tolerance Mechanisms
Communication System
Policy Configuration, Decision Guidance
Fault, Error and Trend Notification
Fault Tolerance Mechanism Invocation
Multiple, Fully Redundant Service Nodes
Coordinated Global Policy-Based Analysis and Decision Making
Event Distribution
Users, Administrator, System Services
Local Policy-Based Analysis
Remote Node Status Probes
Highly Available RAS Engine

Figure 3. Previously proposed HPC RAS framework [22]

This framework poses a number of research challenges, such as optimal pre-processing of node-local data, scalable data aggregation, combined environmental monitoring and event log data analysis for failure prediction, and interaction of FT mechanisms with job and resource management. While prior prototypes focused on FT mechanisms [20, 29] and coordination [28], the solution presented in this paper targets a functioning modular framework that can be deployed in existing HPC systems to incrementally solve the remaining challenges.

Our approach centers around the Type 4 database and its interaction with all other components. This database represents the previously proposed highly available RAS engine. In addition to monitoring and log data, it also collects data from the HPC job and resource manager. The data is processed by an analysis component that also implements FT policies and triggers FT mechanisms. The interface between the database and data provider components is the Structured Query Language (SQL). The database is designed to utilize separate tables unique for each data provider to allow for individual features to be represented. While this non-unified database scheme causes the analysis component to be dependent on individual data provider conventions, such as metric names, ranges and resolutions, it also allows to compare and combine the data from individual provider components. For high availability, the database can be easily replicated [15], while the provider and FT mechanism components are either stateless or utilize their own replication strategies [27].

4 Implementation

The proactive FT framework (Figure 4) was implemented on Linux as most HPC systems are running it on all or at least on service nodes. The database management system is MySQL. In addition, several stateless daemon processes and scripts were developed to facilitate the storing of provider data into the database via SQL and to perform data analysis.

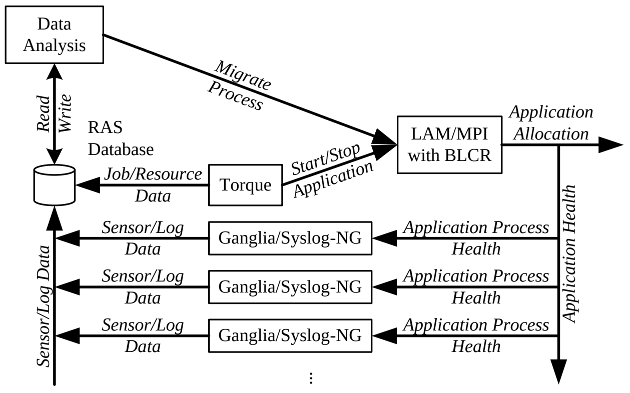

Figure 4. The developed proactive FT framework

Environmental Monitoring The environmental monitoring component consists of Ganglia [17] and a stateless daemon, `gangliamysqld`, that regularly queries Ganglia and stores its data in the RAS framework database.

Environmental data, such as processor temperature and fan speeds, is gathered in intervals on all nodes and disseminated to all nodes using Ganglia's monitoring daemon, `gmond`. It is queried on port 8649 in intervals by `gangliamysqld`, typically on the database node. The data is converted from Extensible Markup Language (XML) format to respective SQL statements that add the data to the `ganglia` table of the database. Using a generic XML schema, any kind of metric that Ganglia provides can be automatically stored. `gangliamysqld` does not need to reside on the same node as the database as it is able to connect to a remote database as well. However it needs to reside on a node that runs `gmond`. Metrics are stored with time stamps of the time of storage. If available, the time of measurement is stored as well. A discussion on the notion of time in the framework can be found in the data analysis component description. Metrics are stored in raw format without pre-processing as it is currently unknown what type of pre-processing, such as classification and/or trend analysis, makes sense for proactive FT. This has certain scalability implications that are discussed later.

Ganglia's output is also used to maintain a `nodesstate` table in the database that contains the list of currently available nodes, and a `nodeshistory` table that logs status changes of nodes.

Event Logging The event logging component consists of Syslog-NG and a stateless daemon, `syslogmysqld`, that regularly stores Syslog-NG messages in the database.

Event log messages, such as daemon/kernel warnings/errors, are gathered on all nodes and send to a central service node by the internal forwarding capability of the Syslog-NG daemon, `syslog-ng`. The central `syslog-ng` daemon writes all messages to a named pipe, `/var/syslogmysqld`, which is read out in intervals by `syslogmysqld`. The conversion from Syslog-NG format to SQL statements that add the messages to the `syslog` table of the database is performed while writing them into the named pipe. Using a generic schema, any kind of message that Syslog-NG provides can be automatically stored. `syslogmysqld` does not need to reside on the same node as the database as it is able to connect to a remote database. However it needs to reside on the central service node that collects all Syslog-NG messages. Messages are stored without pre-filtering in the database with two time stamps, time reported and time stored.

Job and Resource Monitoring The job and resource monitoring component consists of Torque [5] and two stateless scripts, `prologue` and `epilogue`, that gather job and resource data from Torque for the database.

Job and resource data, such as job/user id/name, resources requested/used and start/stop times, is gathered on the node Torque resides, which is typically the head node. The `prologue` script stores data on job start into the database using Torque's prologue feature, while the `epilogue` script stores data on job stop using Torque's epilogue feature. Data conversion from Torque's shell variable and job file format to SQL statements that add the data to the `torque` table of the database is performed by the `prologue` and `epilogue` scripts. These scripts do not necessarily need to reside on the same node as the database as they are able to connect to a remote database, however, they need to reside on the central node that runs Torque as well as on any compute node that runs Torque's pbs-mom server. Since the database does not track the schedule of submitted jobs, the proactive FT framework does not support reliability-aware scheduling [25] at the moment.

Data Analysis The data analysis component consists of a stateless daemon, `migrationd`, that regularly queries the database, performs statistical analysis on gathered data and, if needed, triggers migration of processes or VMs away from unhealthy compute nodes.

In contrast to the OVIS 2 [2] approach of processing data by triggering user-defined database functions on table updates, we opted for querying in intervals for two reasons: (1) the full or partial lock on the database that assures that the analysis operates on consistent data should be managed by the single consumer and not by multiple producers; and (2) statistical analysis in regular intervals is a better fit to the notion of time (global clock) in a massively parallel and distributed computing system.

Today's large-scale HPC systems have hundreds-of-thousands of processor cores in tens-of-thousands of compute nodes. Traditional mechanisms to maintain synchronized clocks in distributed systems, such as the network time protocol (NTP) [19], do not scale in these HPC systems as they add a significant amount of network traffic and OS noise. Hardware mechanisms for a global clock, such as in the Cray XD1 [6], are not available in many systems. Data analysis needs to deal with the fact that node clocks are only synchronized in a coarse grain fashion, such as at boot time. It also needs to take into account that causality (root cause and effect) can only be inferred outside a time window of uncertainty. That is why our framework uses the data entry/change time stamp in the database as a global clock for analysis and falls back to node time stamps only if they represent a more accurate global clock. The time window of uncertainty depends on the data sampling intervals, the data dissemination delay and the `gangliamysqld`/`syslogmysqld` intervals. In contrast, the job and resource monitoring data provided by Torque is *relatively* time accurate.

The data analysis component currently only supports Type 1 threshold triggering and Type 2 trend analysis, both for environmental monitoring data only. Type 3 correlation of short-term application health context/history and Type 4 correlation of long-term application health context/history require significant experimentation time and the usage of more advanced statistical techniques, such as clustering and machine learning [4]. Combined environmental monitoring and event log data analysis for failure prediction is also not implemented yet due to missing historical data and respective experience. The data analysis component does offer the option of caching intermediate analysis results in database. The `migrationd` daemon does not necessarily need to reside on the same node as the database as it is able to connect to a remote database. However it needs to reside on a node from where the migration mechanism can be invoked.

Migration The migration component reuses prior work in process-level migration based on the BLCR checkpoint/restart layer for LAM/MPI [29]. The migration is simply invoked from the command line by the stateless daemon of the data analysis component, `migrationd`.

5 Results

The developed framework was deployed on a 64-node Intel-based Linux cluster with Ganglia and Syslog-NG on all nodes and Torque on the head node. The database and the `gangliamysqld` and `migrationd` daemons were deployed on the head node. The `syslogmysqld` daemon was deployed on a separate service node. The `prologue` and `epilogue` scripts were deployed on all nodes. The framework was able to gather the data, to analyze it and to trigger migration based on sensor thresholds, such as at low fan speeds and high processor temperatures. As the `migrationd` daemon reuses prior work that has been already demonstrated [29], our main focus was to investigate the challenges ahead, such as optimal pre-processing, scalable data aggregation, combined data analysis and interaction of FT mechanisms with job and resource management.

While this work is still ongoing due to the need for long-term data, initial results point out certain issues.

In the first experiment, all Syslog-NG messages and over 20 metrics, such as system/user processor/memory utilization, temperatures and fan speeds, were gathered for offline statistical analysis. The amount of data exceeded 20GB in 27 days of operation with a 30 second sampling interval. This corresponds to an accumulation rate of $\sim$33MB/h or $\sim$275kB/interval. This experiment showed that storing raw data is a serious scalability challenge that needs to be addressed in the future through pre-processing and scalable data aggregation/reduction. Further investigations also need to target appropriate data archiving and aging policies. For example, Ganglia's internal aging mechanism reduces data by combining/increasing sampling intervals. Future work needs to focus on when to archive or age data and on how much information can be lost.

To avoid that an application uses resources it does not own, migration was performed using spare nodes within the set of allocated nodes. The initial set of used nodes and changes due to migration are tracked in the database. This is a temporary fix as modern job and resource managers should support migration and spare nodes soon due to similar use cases in cloud computing.

In a second experiment, all Syslog-NG messages and over 40 metrics, including network and disk I/O statistics, were collected with a 30 second interval. The NAS Parallel Benchmark (NPB) [1] suite was run to measure any overhead introduced by the data collection. We executed the NPB CG, FT and LU benchmarks with class C problem size, at half-scale (32 nodes) to gain run time length, and averaged the execution time over 10 test runs. The results (Table 2) suggest that the data collection did not had an effect at this scale. However, as the amount of data grows linear with system size, effects should be seen at larger scale.

Class C NPB on 32 nodes	CG	FT	LU
Average time in seconds	264	235	261
Average time under load in seconds	264	236	260

Table 2. NPB test results (averages over 10 runs)

6 Summary and Future Work

We presented a proactive FT framework that performs environmental monitoring, event logging, parallel job monitoring and resource monitoring to analyze HPC system reliability and to permit fault avoidance through migration. The framework was deployed on a 64-node Linux cluster to gain hands-on experience and to perform an initial investigation of the challenges ahead. Further implementation details, additional experimental results and a user guide are available in a Master's thesis [16].

Future work will focus on the identified challenges, such as optimal pre-processing, scalable data aggregation and combined data analysis. The framework continues to be deployed on our small-scale system to gather long-term data. We also plan to deploy it on an institutional mid-scale production-type HPC system with hundreds of compute nodes. Additional ongoing work toward a holistic FT framework also focuses on integrating the adaptive reactive FT approach, where checkpoint intervals are changed based on system reliability [7].

7 Acknowledgements

This research is sponsored by the Office of Advanced Scientific Computing Research; U.S. Department of Energy. The work was performed at Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725.

References

[1] Advanced Supercomputing Division, National Aeronautics and Space Administration (NASA), Ames, CA, USA. NAS Parallel Benchmarks (NPB) documentation, 2009. URL http://www.nas.nasa.gov/Resources/Software/npb.html.

[2] J. M. Brandt, B. J. Debusschere, A. C. Gentile, J. R. Mayo, P. P. Pébay, D. Thompson, and M. H. Wong. OVIS-2: A robust distributed architecture for scalable RAS. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS): Workshop on System Management Techniques, Processes, and Services (SMTPS)*, 2008. URL https://ovis.ca.sandia.gov/mediawiki/images/6/60/Ovis-ipdps08.pdf.

[3] S. Chakravorty, C. L. Mendes, and L. V. Kalé. Proactive fault tolerance in MPI applications via task migration. In *Lecture Notes in Computer Science: International Conference on High Performance Computing (HiPC)*, volume 4297, pages 485–496, 2006. URL http://www.springerlink.com/content/9q840u6310467255/.

[4] K. Charoenpornwattana, C. B. Leangsuksun, A. Tikotekar, G. R. Vallée, and S. L. Scott. A neural networks approach for intelligent fault prediction in HPC environments. In *High Availability and Performance Workshop (HAPCW), in conjunction with the High-Performance Computer Science Week (HPCSW)*, 2008. URL http://xcr.cenit.latech.edu/hapcw2008/program/papers/101.pdf.

[5] Cluster Resources, Inc., Salt Lake City, UT, USA. TORQUE Resource Manager documentation, 2009. URL http://www.clusterresources.com/torque.

[6] Cray Inc., Seattle, WA, USA. Cray XD1 computing platform documentation, 2007. URL http://www.cray.com/products/legacy.html.

[7] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computing Systems (FGCS)*, 22(3):303–312, 2006.

[8] J. T. Daly. ADTSC nuclear weapons highlights: Facilitating high-throughput ASC calculations. Technical Report LALP-07-041,

Los Alamos National Laboratory, 2007. URL http://www.lanl.gov/orgs/adtsc/publications/nw_highlights_2007/ch13/13_2daly_facilitating.pdf.

[9] C. Du and X.-H. Sun. MPI-Mitten: Enabling migration technology in MPI. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 11–18, 2006. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1630790.

[10] E. N. M. Elnozahy and J. S. Plank. Checkpointing for peta-scale systems: A look into the future of practical rollback-recovery. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 1(2):97–108, 2004.

[11] E. N. M. Elnozahy, R. Bianchini, T. El-Ghazawi, A. Fox, F. Godfrey, A. Hoisie, K. McKinley, R. Melhem, J. S. Plank, P. Ranganathan, and J. Simons. System resilience at extreme scale. Technical report, Defense Advanced Research Project Agency (DARPA), 2008. URL http://institutes.lanl.gov/resilience/docs/Toward%20Exascale%20Resilience.pdf.

[12] C. Engelmann, G. R. Vallée, T. Naughton, and S. L. Scott. Proactive fault tolerance using preemptive migration. In *Euromicro International Conference on Parallel, Distributed, and network-based Processing (PDP)*, pages 252–257, 2009.

[13] S. Fu and C.-Z. Xu. Exploring event correlation for failure prediction in coalitions of clusters. In *IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–12, 2007.

[14] R. Gioiosa, J. C. Sancho, S. Jiang, and F. Petrini. Transparent, incremental checkpointing at kernel level: A foundation for fault tolerance for parallel computers. In *IEEE/ACM International Conference on High Performance Computing and Networking (SC)*, page 9, 2005. URL http://hpc.pnl.gov/people/fabrizio/papers/sc05.pdf.

[15] Hewlett-Packard Development Company, L.P., Palo Alto, CA, USA. Managing Serviceguard – Fifteenth edition, 2007. URL http://docs.hp.com/en/B3936-90122/B3936-90122.pdf.

[16] A. Litvinova. Reliability availability and serviceability framework engine prototype. Master's thesis, Department of Computer Science, University of Reading, UK, 2009.

[17] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[18] H. Meuer, E. Strohmaier, J. J. Dongarra, and H. Simon. Top 500 list of supercomputer sites, 2009. URL http://www.top500.org.

[19] D. L. Mills. The Network Time Protocol (NTP) distribution, 2009. URL http://www.eecis.udel.edu/~mills/ntp/html/index.html.

[20] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott. Proactive fault tolerance for HPC with Xen virtualization. In *ACM International Conference on Supercomputing (ICS)*, pages 23–32, 2007. URL http://www.csm.ornl.gov/~engelman/publications/nagarajan07proactive.pdf.

[21] National Energy Research Scientific Computing Center (NERSC), Lawrence Berkeley National Laboratory (LBNL), Berkeley, CA, USA. Current and past HPC system availability statistics, 2009. URL http://www.nersc.gov/nusers/status/AvailStats.

[22] S. L. Scott, C. Engelmann, G. R. Vallée, T. Naughton, A. Tikotekar, G. Ostrouchov, C. B. Leangsuksun, N. Naksinehaboon, R. Nassar, M. Paun, F. Mueller, C. Wang, A. B. Nagarajan, and J. Varma. A tunable holistic resiliency approach for high-performance computing systems. Poster at the 14^{th} ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP) 2009, Raleigh, NC, USA, 2009. URL http://www.csm.ornl.gov/~engelman/publications/scott09tunable.pdf.

[23] M. Seager. Operational machines: ASCI White. Talk at the 7^{th} Workshop on Distributed Supercomputing (SOS) 2003, 2003. URL http://www.cs.sandia.gov/SOS7/presentations/seager_white.ppt.

[24] J. Stearley and A. J. Oliner. Bad words: Finding faults in Spirit‘s syslogs. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid): Workshop on Resiliency in High Performance Computing (Resilience)*, 2008. URL http://xcr.cenit.latech.edu/resilience2008/program/resilience08-3.pdf.

[25] X.-H. Sun, Z. Lan, Y. Li, H. Jin, and Z. Zheng. Towards a fault-aware computing environment. In *High Availability and Performance Workshop (HAPCW), in conjunction with the High-Performance Computer Science Week (HPCSW)*, 2008. URL http://xcr.cenit.latech.edu/hapcw2008/program/papers/104.pdf.

[26] A. Tikotekar, G. Vallée, T. Naughton, S. L. Scott, and C. Leangsuksun. Evaluation of fault-tolerant policies using simulation. In *IEEE International Conference on Cluster Computing (Cluster)*, 2007.

[27] K. Uhlemann, C. Engelmann, and S. L. Scott. JOSHUA: Symmetric active/active replication for highly available HPC job and resource management. In *IEEE International Conference on Cluster Computing (Cluster)*, 2006. URL http://www.csm.ornl.gov/~engelman/publications/uhlemann06joshua.pdf.

[28] G. R. Vallée, K. Charoenpornwattana, C. Engelmann, A. Tikotekar, C. B. Leangsuksun, T. Naughton, and S. L. Scott. A framework for proactive fault tolerance. In *International Conference on Availability, Reliability and Security (ARES)*, pages 659–664, 2007. URL http://www.csm.ornl.gov/~engelman/publications/vallee08framework.pdf.

[29] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott. Proactive process-level live migration in HPC environments. In *IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2008. URL http://www.csm.ornl.gov/~engelman/publications/wang08proactive.pdf.