Aggregation of Real-Time System Monitoring Data for Analyzing Large-Scale Parallel and Distributed Computing Environments*

S. Böhm[1,2], C. Engelmann[1], and S. L. Scott[1]
[1]Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
[2]Department of Computer Science
The University of Reading, Reading, RG6 6AH, UK
swen.boehm@bnc.info,{engelmannc,scottsl}@ornl.gov

Abstract

We present a monitoring system for large-scale parallel and distributed computing environments that allows to trade-off accuracy in a tunable fashion to gain scalability without compromising fidelity. The approach relies on classifying each gathered monitoring metric based on individual needs and on aggregating messages containing classes of individual monitoring metrics using a tree-based overlay network. The MRNet-based prototype is able to significantly reduce the amount of gathered and stored monitoring data, e.g., by a factor of ≈56 in comparison to the Ganglia distributed monitoring system. A simple scaling study reveals, however, that further efforts are needed in reducing the amount of data to monitor future-generation extreme-scale systems with up to 1,000,000 nodes. The implemented solution did not had a measurable performance impact as the 32-node test system did not produce enough monitoring data to interfere with running applications.

1. Introduction

Today's supercomputers are typically parallel architectures that have some distributed features. The top tier currently scales to 50,000 multi-core compute nodes or 300,000 processor cores. For example, the Jaguar Cray XT5 system at Oak Ridge National Laboratory (see http://www.nccs.gov/jaguar) has 224,256 cores in the form of 37,376 six-core AMD Opteron processors organized in 18,688 dual-processor compute nodes with four 4 GB memory modules (300 TB in 74,752 modules) that are connected via a 3-D twisted torus interconnect. It is the world's fastest supercomputer with a LINPACK benchmark performance of 1.759 PFlop/s and a theoretical peak of 2.331 PFlop/s (see the Top 500 List of Supercomputers at http://www.top500.org).

As processor frequency scaling came to a hold in recent years due to associated power consumption and dissipation issues, component count increases took over in high-performance computing (HPC) for building ever-faster supercomputers. Ongoing planning activities by national research councils in the U.S., Europe, and Asia toward multi-petascale and exascale HPC recognize that these increases will continue (see the G8 Research Councils Initiative on Multilateral Research Funding at http://www.dfg.de/g8-initiative and [7, 8]). It is expected that an exascale HPC system will have up to 1,000,000 nodes with 1,000 cores/node by 2018 (see the International Exascale Software Project at http://www.exascale.org and [5]).

As component counts increase, HPC systems need to employ more distributed features to allow hardware, system software, middleware, and applications to exploit parallelism without being hindered by the management of their massive amount of resources. Finding the right balance between the general parallel nature of HPC and the emergence of distributed features at extreme-scale is creating a new set of research challenges for the parallel and distributed computing community. While existing parallel computing solutions reach certain scaling limits, existing distributed computing alternatives do not meet certain HPC performance requirements, such as low-latency/high bandwidth communication.

*This research is sponsored by the Office of Advanced Scientific Computing Research; U.S. Department of Energy. The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. De-AC05-00OR22725. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

The system software research and development presented in this paper focuses on the monitoring subsystem that identifies compute-node health and utilization status in supercomputers as part of their reliability, availability and serviceability (RAS) systems. Monitoring solutions for HPC systems gather data (metrics) from hardware sensors on compute-node boards, such as temperatures, voltages, and fan speeds, as well as from software monitors, such as processor and memory utilization, and disk and network I/O rates. These metrics are used to determine component health issues, such as overheating situations, voltage spikes, and fan faults, as well as component utilization problems, such as scheduling inefficiencies and resource contention. The central job and resource management system is informed about compute-node status changes and takes appropriate action, such as reallocation, to assure optimal utilization as well as efficient job throughput.

There are two distinct operating modes for HPC monitoring systems. The first variant processes all metrics locally and sends out alarms to the central job and resource manager, the second sends all (or select) metrics to the job and resource manager for processing. While the first option is more scalable, it is unable to identify correlation between components, such as a systematic heat increase in a set of compute nodes associated to a specific job. The second option does provide this feature, but is more limited in scale as more data is sent to and processed by a central manager.

This paper presents the architecture and results of an implementation of a HPC monitoring system that allows a tunable trade-off between these two variants, such that scalability is enhanced through the reduction of metrics sent to the central job and resource manager, while the capability to perform system-wide correlation analysis is maintained. The approach focuses on classifying metrics during collection and on aggregating classified metrics across the entire system. The presented solution also opens new opportunities in scalable in-flight processing of monitoring data.

2. Related Work

2.1. System Monitoring

Ganglia [10] (see http://ganglia.sourceforge.net) is a scalable distributed monitoring system, where each node monitors itself and disseminates data via multicast to (1) all other nodes in a flat design targeted at small clusters or (2) a subset in a hierarchical design targeted at larger cluster federations. It uses Extensible Markup Language (XML) for human readable data representation, eXternal Data Representation (XDR) for

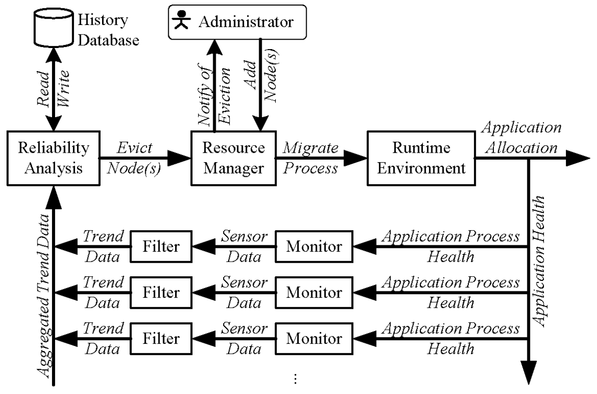

Figure 1. Feedback-loop control for proactive fault tolerance with health monitoring, data analysis, and application reallocation [6]

portable and efficient data transport, and the Round Robin Database tool (RRDtool) for data storage and visualization. In Ganglia, a node-local monitoring daemon process, `gmond`, collects various metrics from hardware sensors and software monitors, and sends Internet Protocol (IP) multicast packets to disseminate these metrics to other `gmond` processes. Additional metrics can be added to `gmond` with the `gmetric` tool. For a hierarchical structure, a second daemon process, `gmetad`, collects the data from `gmond` multicast groups or directly from every `gmond` in a group. The `gmetad` daemons themselves can again be queried by another `gmetad` daemon to monitor federated systems. Each `gmetad` saves its gathered data locally using RRDtool. Saved data is reduced (averaged) and aged out (deleted after a certain amount of time) to maintain fixed-size databases. Ganglia has been used in systems with up to 2,000 compute nodes. Its dependence on IP and reliance on either IP multicast or serial query has hindered adoption to larger-scale systems, especially with custom interconnects. The communication of all monitoring data of the entire system is a clear scalability limit. Data reduction and aging in the database makes long-term logs and data analysis without a separate external database impossible.

A recent investigation by the authors [9] in storing monitoring and system log data for statistical analysis to aid proactive fault tolerance schemes revealed certain scalability issues. Proactive fault tolerance [6] avoids experiencing failures through preventative measures, such as by migrating application parts away from compute nodes that are "about to fail". It relies on a feedback-loop control (Figure 1) with continuous health monitoring, data analysis, and application reallocation. This new HPC resilience approach requires a system-wide analysis using detailed monitoring and system log data to identify anomalies and early failure indications. The prototype system [9] uses Ganglia and Syslog-NG to accumulate data into a MySQL database. On a 64-node cluster, all syslog messages and over 20 metrics were gathered in a 30 second interval for offline statistical analysis. The amount of data exceeded 20 GB in 27 days of operation with a 30 second sampling interval, corresponding to an accumulation rate of ≈33 MB/h or ≈275 kB/interval. This experiment showed that storing raw data is a serious scalability challenge that needs to be addressed. This work was the motivating factor for the research and development presented in this paper.

Nagios [1] (see http://www.nagios.org) is a information technology (IT) infrastructure monitoring solution that allows storing metrics in a Structured Query Language (SQL) database and offers visualization via a Web-based front-end. For metric data collection, it supports active (linear) querying of monitored resources similar to Ganglia's `gmetad`, as well as passive notifications or data streams from monitored resources similar to Ganglia's `gmond`. The primary target for Nagios is monitoring IT server farms.

OVIS 2 [3] (see http://ovis.ca.sandia.gov) is a hierarchical system monitoring and analysis tool that collects metrics directly from compute nodes or from other monitoring solutions, such as Ganglia, for processing using statistical methods for graphical presentation. OVIS 2 supports SQL databases, as well as flat and hierarchical approaches similar to Ganglia. In addition, it offers a comprehensive visualization tool, including a 3D real-time representation of the monitored system and its current state. OVIS 2 is geared toward operations support for large-scale HPC systems.

Most HPC vendors, such as Cray and IBM, supply their own proprietary RAS systems that offer features similar to Ganglia (see http://docs.cray.com and http://www.ibm.com/deepcomputing). For extreme-scale systems, such as Cray's XT series or IBM's Blue Gene series, separate RAS nodes and a separate RAS network is deployed to allow non-intrusive system monitoring and management. These RAS sub-systems are deployed in a hierarchical fashion, such that z compute nodes are monitored by a level-0 RAS node, y level-0 RAS nodes are monitored by a level-1 RAS node, and so on. To avoid massive amounts of monitoring data, metrics are only sent to the root RAS node when a certain threshold (alarm or significant change) is triggered.

Recent advances in purely distributed monitoring solutions disseminate metrics using a Gossip protocol in peer-to-peer networks [15, 14]. While Gossip protocols significantly enhance scalability, they trade off the coordinated real-time response that is needed for HPC job and resource management.

2.2. Tree-based Overlay Networks

Recent advances in collecting performance metrics of HPC applications at runtime focused on tree-based overlay networks (TBONs).

Multicast Reduction Network (MRNet) [13] (see http://www.paradyn.org/mrnet) is a software overlay network for multicast and reduction communication in parallel and distributed environments. It uses a tree of processes between a front-end (tree root) and back-ends (tree leaves) for scalable communication performance. In addition to communication, MRNet allows computation on the data that is transported trough the TBON at each tree node in a scalable fan-in/-out tree fashion using up-stream and down-stream filter plugins. This programming model offers simplicity and scalability, as well as fault tolerance if no state is kept at tree nodes.

MRNet has been recently used in conjunction with the TAU Performance System^{TM} as TAUoverMRNet (ToM) [11] for scalable profiling and tracing of HPC applications. ToM uses statistical filtering at tree nodes to generate performance profile histograms.

MRNet has also been recently used in with Ganglia for aggregating RRDs from compute nodes via group file operations, which allow to access local file systems in a distributed system as it would be one networked file system [4]. The RRD files on the compute nodes are read by the root node in a fan-in tree fashion.

3. Technical Approach

The targeted monitoring solution for large-scale HPC systems needs to be scalable, but it also needs to supply real-time data to support allocation decisions by the central job and resource manager. The taken technical approach aims at aggregating real-time monitoring data in a scalable fashion using a TBON, such that only necessary information is transmitted via the fan-in tree and the monitoring system is managed with control messages via the fan-out tree.

Figure 2 depicts the overall architecture of the targeted monitoring system. The back-end processes of the TBON are located on all compute nodes and perform monitoring activities, as well as, classify collected metrics to reduce the amount of data that is transmitted. The intermediate processes reside on a subset of the compute nodes and aggregate collected metrics in a fan-in tree fashion. Optional computation, such as statistical analysis similar to ToM (see Section 2.2 and [11]), may be performed by the intermediate processes while the data is transported toward the front-end process. The intermediate processes are connected in a way that closely matches the real network architecture to get maximum

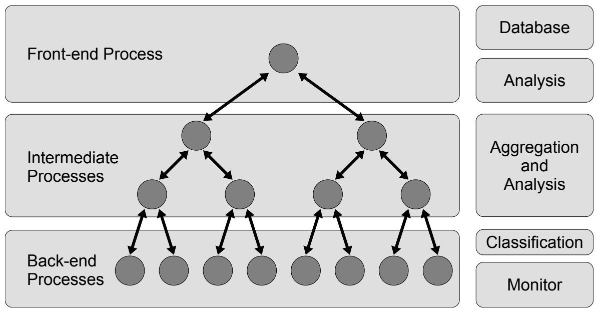

Figure 2. Utilizing a tree-based overlay network, such as MRNet, for scalable aggregation and analysis of real-time monitoring data

performance. The front-end process optionally performs further computation on the data and stores it in a database for processing by external tools.

The classification of monitoring data essentially reduces the granularity of metrics. Since each metric has a different meaning and may require a different classification scheme, individual configuration on a metric-by-metric basis is employed. A simple classification scheme, like `under=0|normal=1|over=2`, does not give much details. To allow adaptation to specific metrics and use cases, a completely configurable classification scheme is supported that even allows to have one class per metric value, *i.e.*, to transmit the actual metric value instead of its class. The classification scheme can be changed at runtime by reconfiguring the monitoring system to allow for adaptation, such as for a more fine-grain observation when needed.

Data aggregation along the fan-in tree can be synchronous or asynchronous. In the synchronous variant, all back-ends send a message containing their current classes in a certain interval to their respective intermediates. All intermediates wait for the sub-tree data before passing it on as aggregated message. Unchanged classes are generally omitted. In the asynchronous version, intermediates route all messages individually. As this variant generally creates more messages and requires intermediates to maintain state if computation is involved, the synchronous variant is preferred.

Data analysis may be optionally performed at the intermediates, such as to further reduce the amount of data or to pre-process the data for evaluation at the root. Computation may also optionally be performed at the root, such as for a system-wide reliability or load analysis. The aggregated data is stored in a database without a predefined reduction or aging policy to allow for off-line analysis and to enable experimental on-line analysis based on prior off-line analysis results.

With the data already stored in the database and continuous updates, the monitoring system produces an aggregated representation (view) of the health and utilization status of the monitored computing system in real-time in the form of classified metric data. The real-time window is defined by the collection interval on the back-ends and the time to communicate metric data to the front-end via the TBON. If the collection interval on the back-ends is much larger than the time to communicate the data to the front-end, which is expected to be the case even for extreme-scale systems due to the employed tree-based communication subsystem, the collection interval becomes the determining factor for the real-time properties of this monitoring system.

4. Implementation

MRNet has already proven to be very powerful TBON solution. The presented work is a newly implemented system monitoring framework based on MRNet as a communication substrate. Figure 3 details its architecture, including the physical location and communication setup of the TBON's front-end, back-end and intermediate processes. Figure 4 shows the interaction between front-end and back-ends, for which the communication is routed via the fan-in/-out tree formed by the intermediates. More details about each component are provided in the following.

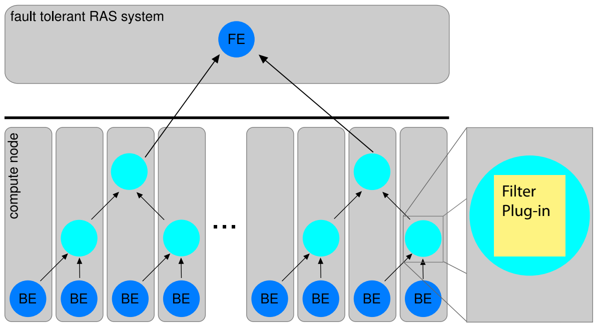

Figure 3. Mapping TBON front-end, intermediates, and back-ends of the implemented monitoring framework to nodes in a HPC system

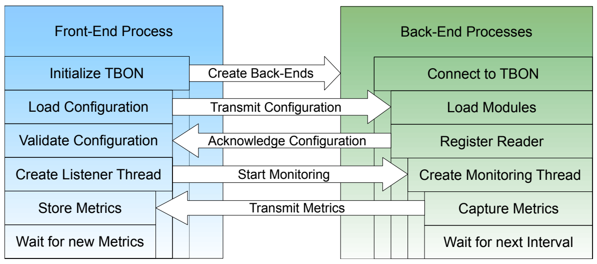

Figure 4. Interaction between front-end and back-ends (communication via intermediates)

The monitoring system follows an object-oriented design and is implemented in C++. It utilizes the Boost C++ libraries (see http://www.boost.org) for efficiency and for object serialization support. It also relies on the GNU libtool dynamic library loader (LTDL, see http://www.gnu.org/software/libtool) for portable support of loading monitoring modules at the back-ends (for each metric or metric group) and processing modules at the intermediates (metric aggregation and statistical filters). The C++ interface to MySQL is used for storing metric update message objects directly into the database. The distribution relies on the GNU autotools (see http://www.gnu.org/software/autoconf) for portability.

4.1. Front-End Process

The front-end (FE) is located on the RAS management node of the HPC system, which is also often the head node running the system's job and resource management service. It is responsible for setting up the TBON, including the intermediates and back-ends, and for storing the received data in a MySQL database.

For instantiation, MRNet is booted up and configured with one fan-out stream for control messages and one fan-in stream for monitoring messages. Once all intermediates and back-ends are up and running, a configuration message is sent out to set up the in-flight (fan-in) processing at the intermediates, and the data collection and classification at the back-ends The intermediates and back-ends load needed modules using the GNU libtool dynamic library loader and configure them accordingly. The front-end is able to change the configuration, especially the classification scheme for any metric, at this point only by reconfiguring the entire system. Support for individual reconfiguration of intermediates and back-ends for each metric at runtime is planned.

As the back-ends only transmit updates for each metric, *i.e.*, class changes, the front end is also storing only these changes with the respective time stamp in the database. Both, the classification scheme and the updates-only behavior, provide a significant reduction in data traffic and database size. Database SQL queries are able to reconstruct a snapshot at a specific point in time by considering prior metric updates. Data processing at the front-end, such as for statistical analysis, is currently performed off-line and not integrated as an online tool as this is an ongoing research project.

4.2. Back-End Processes

The back-ends (BEs) are located on the nodes to be monitored, *i.e.*, on the compute and service nodes of the HPC system. Optionally, a back-end may also reside on the front-end node. The current monitoring support in the back-ends offers gathering metrics using the `/proc` file system for data available via the operating system and `libsensors` for hardware sensor data available via the Intelligent Platform Management Interface (IPMI) (see http://openipmi.sourceforge.net).

The configuration message from the front-end contains the list of metrics to be gathered, and a classification scheme and collection interval for each metric. The back-end is designed in a modular fashion (using dynamic libraries) to allow extending collection to different metrics and mechanisms, and to reduce the back-end's footprint to the necessary minimum.

Collection, classification, and transmission is performed in regular intervals for each metric. Overlapping metric intervals and respective outgoing messages are aggregated to allow for efficient operation, *i.e.*, only one message goes out for each interval containing the respective metric IDs and classes (see Figure 5). As mentioned earlier, these messages contain metric updates only. They are sent to the intermediates for aggregation in the fan-in tree (see next section).

back-end ID	count	metric ID	class	···	metric ID	class

Figure 5. Message format for transferring metric updates from back-ends to intermediates

4.3. Intermediate Processes

The intermediates are located on the same nodes as the back-ends and facilitate the TBON communication (see Figure 3 in Section 4). If a back-end is also located on the front-end node (for monitoring the front-end node), an intermediate may be placed there as well. Some advanced HPC systems, such as the Cray XT series and the IBM Blue Gene series, already have hierarchical RAS systems (see Section 2.1), on which the TBON can be mapped to by placing an intermediate on each RAS system node.

Configured for synchronous operation, the intermediates utilize message aggregation plug-ins (primitive forwarding filters) loaded at configuration time that simply attach incoming messages to each other (see Figure 6). As wavefronts of metric updates progress along the fan-in tree in a synchronous fashion, the number of messages shrinks and their size grows with each step toward the root (front-end).

back-end ID	count	metric ID	class	···	metric ID	class	···
			···				
···	back-end ID	count	metric ID	class	···	metric ID	class

Figure 6. Message format for aggregating metric updates by the intermediates

The synchronous operation and data aggregation permits in-flight processing, such as statistical analysis, by applying the same operation on the incoming data at each intermediate. However, data processing at the intermediates using more advanced plugins is currently not implemented as this is an ongoing research project.

5. Experimental Results

5.1. Monitoring Data Accumulation

We deployed the framework on the same 64-node cluster (in a 32-node degraded fashion due to faulty hardware) that was used for our earlier investigations (see Section 2.1 and [9, 6]). For this test, we sampled 18 metrics on 32 nodes over a 4 hour period with constantly varying classes and a sample interval for all metrics of 30 seconds. The test resulted in about 1 MB of data collected at the front-end, which corresponds to an accumulation rate of ≈250 kB/h or ≈2 kB/interval.

Adjusted to the number of nodes and metrics involved, roughly 56× less data than in our previous experiments with Ganglia was transmitted and stored. While this is not necessarily a fair comparison as not exactly the same metrics were gathered, it points out the significant contribution provided by classifying monitoring data and by omitting unchanged values.

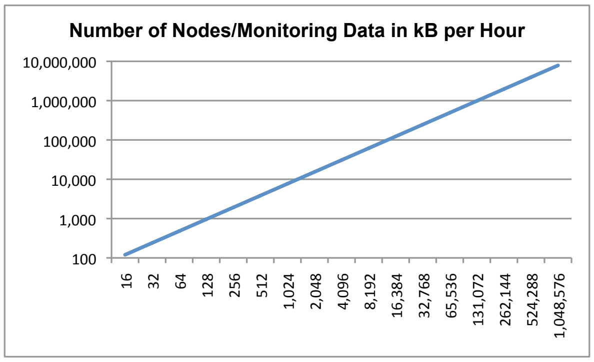

Figure 7. Scaling up the experienced monitoring data rate (accumulation rate in kB/h on the y-axis vs. node count on the x-axis)

The amount of data collected by the implemented monitoring system currently scales linear with the number of compute nodes. A simple scaling study (Figure 7) based on the experienced ≈64 B/interval per compute node reveals an accumulation rate of ≈6.1 MB/interval on 100,000 nodes and ≈61 MB/interval on 1,000,000 nodes. Using a 30 second interval, this corresponds to ≈732 MB/h on a 100,000-node system and ≈7.2 GB/h on a 1,000,000-node system. For realistic real-time response scenarios to emerging health threads or utilization issues, this accumulation rate is still to high as the collected data needs to be processed by statistical tools for analysis. Future research and development needs to focus on further monitoring data reduction.

5.2. Performance Impact on Applications

Although the main goal of the presented work is to reduce the amount of data collected by a monitoring system in a large-scale computing environment, we have also measured the performance impact of the monitoring system on applications (see Table 1) using the NAS Parallel Benchmark (NPB) suite (see http://www.nas.nasa.gov/Resources/Software/npb.html). Similar to our earlier results [9], the cluster is to small, *i.e.*, produced not enough data, to have a measurable impact on applications. This result is not surprising as the implemented monitoring system produces much less data than Ganglia used in our earlier experiments.

These results emphasize that the performance impact is not an issue. Both, Ganglia and the presented solution, do not have a performance impact on small-scale systems, yet our prior effort in analyzing system reliability was hampered by the fact that a standard monitoring system, like Ganglia, provides even at this small scale way too much data to analyze in real-time.

Class C NPB on 32 nodes	CG	FT	LU
Without monitoring	264	235	260
With monitoring	264	235	260
Overhead	0%	0%	0%

Table 1. NPB performance on the test cluster with/without the developed monitoring system (averages over 10 runs in seconds)

6. Conclusions and Future Work

We have developed a monitoring solution for parallel and distributed environments that allows to trade-off accuracy in a tunable fashion to gain scalability without compromising fidelity. The approach relies on classifying each gathered metric based on individual needs and on aggregating messages containing classes of individual metrics in a fan-in tree fashion. In comparison to Ganglia, the implemented prototype was able to reduce the amount of data gathered and stored by a factor of ≈56. However, a simple scaling study revealed that further research and development efforts are needed in reducing the amount of data to monitor future-generation extreme-scale HPC systems with up to 1,000,000 compute nodes. The implemented monitoring solution did not had a measurable impact on running applications as the test cluster was too small, *i.e.*, produces not enough monitoring data to interfere with applications. Further implementation details, additional experimental results, and a user guide are available in a Master's thesis [2].

The presented solution is a first step toward scalable system monitoring. The developed prototype has been equipped with certain features to enable our ongoing research and development efforts in optimizing monitoring systems, and in analyzing the health and utilization of large-scale HPC systems. Our planned work on the presented monitoring system includes further data reduction and statistical processing using computational plugins at TBON intermediates, as well as extending its capabilities to aggregate local message logs across a large-scale system. Our planned work in analyzing HPC systems primarily focuses on identifying anomalies and pre-failure indicators by correlating monitoring data logs with application failure logs. A recent study by our team [12] focused on nonparametric multivariate anomaly analysis using off-line monitoring data gathered with Ganglia. We plan to extend this work with the newly developed monitoring system.

References

[1] W. Barth. *Nagios: System and Network Monitoring, 2nd Edition*. No Starch Press, San Francisco, CA, USA, Oct. 2008.

[2] S. Böhm. Development of a RAS framework for HPC environments: Realtime data reduction of monitoring data. Master's thesis, Department of Computer Science, University of Reading, UK, Mar. 12, 2010.

[3] 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 *Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2008: 4th Workshop on System Management Techniques, Processes, and Services (SMTPS) 2008*, Miami, FL, USA, Apr. 14-18, 2008. ACM Press, New York, NY, USA.

[4] M. J. Brim and B. P. Miller. Group file operations for scalable tools and middleware. In *Proceedings of the 16th IEEE International Conference on High Performance Computing (HiPC) 2009*, Kochi, India, Dec. 16-19, 2009. IEEE Computer Society.

[5] J. Dongarra, P. Beckman, T. Moore, J.-C. Andre, J.-Y. Berthou, T. Boku, F. Cappello, B. Chapman, X. Chi, A. Choudhary, S. Dosanjh, A. Geist, B. Gropp, R. Harrison, M. Hereld, M. Heroux, A. Hoisie, K. Hotta, Y. Ishikawa, F. Johnson, S. Kale, R. Kenway, B. Kramer, J. Labarta, B. Lucas, B. Maccabe, S. Matsuoka, P. Messina, B. Mohr, M. Mueller, W. Nagel, H. Nakashima, M. E. Papka, D. Reed, M. Sato, E. Seidel, J. Shalf, D. Skinner, T. Sterling, R. Stevens, W. Tang, J. Taylor, R. Thakur, A. Trefethen, M. Snir, A. van der Steen, F. Streitz, B. Sugar, S. Sumimoto, J. Vetter, R. Wisniewski, and K. Yelick. International exascale software project roadmap (draft 0.93), Nov. 2009.

[6] C. Engelmann, G. R. Vallée, 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*, pages 252–257, Weimar, Germany, Feb. 18-20, 2009. IEEE Computer Society.

[7] G. A. Geist and R. F. Lucas. Major computer science challenges at exascale. Technical report, International Exascale Software Project, Feb. 2009.

[8] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick. ExaScale computing study: Technology challenges in achieving exascale systems. Technical report, Defense Advanced Research Project Agency (DARPA) Information Processing Techniques Office (IPTO), 2008.

[9] A. Litvinova, C. Engelmann, and S. L. Scott. A proactive fault tolerance framework for high-performance computing. In *Proceedings of the 9th IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN) 2010*, Innsbruck, Austria, Feb. 16-18, 2010. ACTA Press, Calgary, AB, Canada.

[10] 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.

[11] A. Nataraj, A. D. Malony, A. Morris, D. C. Arnold, and B. P. Miller. A framework for scalable, parallel performance monitoring. *Concurrency and Computation: Practice and Experience*, 22(6):720–735, 2010.

[12] G. Ostrouchov, T. Naughton, C. Engelmann, G. R. Vallée, and S. L. Scott. Nonparametric multivariate anomaly analysis in support of hpc resilience. In *Proceedings of the 5th IEEE International Conference on e-Science (e-Science) 2009: Workshop on Computational Science*, Oxford, UK, Dec. 9-11, 2009. IEEE Computer Society.

[13] P. C. Roth, D. C. Arnold, and B. P. Miller. MRNet: A software-based multicast/reduction network for scalable tools. In *Proceedings of the ACM/IEEE International Conference on High Performance Computing and Networking (SC) 2003*, Phoenix, AZ, USA, Nov. 15-21, 2003. IEEE Computer Society.

[14] R. Subramaniyan, P. Raman, A. D. George, and M. Radlinski. GEMS: Gossip-enabled monitoring service for scalable heterogeneous distributed systems. *Cluster Computing*, 9(1):101–120, 2006.

[15] J. W. Zhu, P. G. Bridges, and A. B. Maccabe. Lightweight online performance monitoring and tuning with embedded gossip. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 20(7):1045–9219, 2009.