Collecting Sensor Data for High-Performance Computing: A Case Study

Line C. Pouchard, Jonathan D. Dobson, and Stephen W. Poole
Computing and Computational Sciences
Oak Ridge National Laboratory
Oak Ridge, TN, USA

Abstract - *Many research questions remain open with regard to improving reliability in exascale systems. Among others, statistics-based analysis has been used to find anomalies, to isolate root causes, and attempt to predict failures. But well-understood methods and best practices for collecting reliability data in a uniform way are still lacking, which impedes analysis. We report our experience with collecting these data from heterogeneous sources on a testbed cluster and present our data collection tool. This case illustrates the fact that reported metrics largely depend upon individual system configuration. We then investigate standards and specifications in manufacturing and desktop computing to identify concepts that may be useful for representing High Performance Computing (HPC) data and present a taxonomy that utilizes these concepts.*

Keywords: Reliability, Accessibility, Serviceability (RAS), data collection, sensor data representation.

1. Introduction

As HPC approaches the exascale, questions regarding the lack of scalability of current methods for fault-tolerance and resilience, such as replication and checkpointing, have been raised. In particular, the overhead entailed by checkpointing and the expected number of cores may prevent an application to make any progress between failures and re-start. In two recent review papers, Cappello et al. [1,2] outline recommendations advocating improved communication in the resilience community, and propose research topics "to ensure the correct termination of parallel execution on exascale systems." Within this proposed research agenda, this paper attempts to initiate a discussion about the challenges encountered with acquiring data and collecting error information. There is a lack of definitions of what constitutes an error, what we mean by soft, transient, and intermittent errors, and what constitutes failure. System monitoring tools do not follow standard formats, nor are there existing standards on how system managers enter event logs. There is no consensus in the community on what events and metrics we measure to improve fault-tolerance in HPC. A better understanding of failures is needed as several recent studies reach contradictory conclusions as to the root cause of failures – hardware versus software. [3,4]

To illustrate some of the problems identified above, we collected sensor data in an HPC cluster and report on our experience. We implemented a tool that collects data from several sources and share our experience with data collection on our testbed. Collecting hardware-related metrics such as core temperatures, fan rpms and voltages deceptively appear straightforward, until the following obstacles are encountered:

- proprietary and/or embedded sensors report data that cannot be independently tested, or even identified,
- sensors send alerts for non-existent events (false-positive),
- thresholds cannot be easily modified or calibrated, and
- hardware metrics are provided at coarse resolution in time and unit.

We also investigated several specifications for reporting sensor data in an attempt to find suitable concepts for HPC data. Finally, we propose a taxonomy for collecting system monitoring data that accommodates heterogeneous sources.

The paper is organized as follows. After this introduction, Section 2 reviews system monitoring tools with a focus on their data collection aspect. Section 3 contains a description of our collection effort, the architecture, and the implementation of DCAT, our collection tool. Section 4 presents a review of available mechanisms for obtaining data from embedded sensors and existing standards that could be

used to organize this data. In Section 5 we present our taxonomy for storing data independently of its source. Section 6 concludes with further research plans.

2. Related work

Many proprietary and open-source tools for system monitoring in HPC are available, including OVIS, an advanced prediction and analysis tool aimed at discovering targets for check-pointing in nodes about to fail [5, 6]. Ganglia, a widely used tool [7] can report on temperatures and other metrics for a limited time and on a per-metric basis. Thus, it has little value for forensic analysis. Data collectors have been written for OVIS, that capture low-level data from lm-sensors, proc/ files, and other sources [5]. SMARTMON [8] controls and monitors disk storage and is built into most modern disks, but we are also interested in core temperatures and other metrics pertaining to CPU. The Sisyphus system uses latent semantic indexing to discover rare, correlated events in the high volume of messages reported in the system logs at the extreme scale. [9, 10]

These tools represent very important efforts for analyzing the health of systems in clusters and clouds and for trying to predict the next failure, but they do not foster the sharing of information and methodology that would ensure the emergence of standard descriptions of Reliability, Availability, Serviceability (RAS) events recommended in [1, 2]. In particular, one has to dig deep into hardware specifications and software documentation or implementation to find what metrics are actually reported and used to make a prediction in each tool. Given that one purpose of most of these tools is to facilitate system management of large systems, they are designed to mask the complexity of the low-level details from the end user, so the latter is not surprising. However, this situation makes it difficult to establish common definitions and specification for metrics.

Some data models and definitions exist for organizing HPC health metrics. Stearley proposes a State Model derived from a semi-conductor specification defining reliability (SEMI FE10-0304). This model describes the state of a system in terms of its scheduled and non-scheduled downtime, engineering and production time, standby time, and characterizes failure as "ANY transition into unscheduled downtime." [9]. These definitions represent a step in the right direction, but do not appear to being used elsewhere.

3. Data collection, architecture, and prototype implementation

For our case study, we were able to use as testbed a 32-nodes cluster with HP motherboards, and two quad-cores, AMD Opteron 2356, with 16G of RAM each, for a total of 256 processors. Networking is implemented by two Myrinet switches, two 10GE cards, and two DDR IB cards.

Our tool collects physical parameter data, such as temperatures and fan rpms. Our application can read data from lm-sensors [11] and the Intelligent Platform Management Interface (IPMI) [12]. In our case-study, we obtained six temperature readings and six fan readings per node, located as follows:

- Temp 1: System board.
- Temp2 : CPU socket 1.
- Temp3: CPU socket 2.
- Temp4: System board Temperature located under the power supply cage.
- Temp5: System board Power Supply.
- Temp6: System board Temperature.
- Fans 1-6: Two System Internal Expansion Board fans and 4 CPU Fans.

DCAT also collects data from /proc/cpuinfo, /proc/stat, and /proc/meminfo for memory statistics and CPU. The collected data for system state is as follows:

- Cpu usage: recorded in jiffies from which percentages can be calculated per core.
- Cpu idle time: same as above.
- Cpu average usage per node.
- Cpu average idle time per node.
- System or node uptime.
- Memory total: total usable memory on the system.
- Memory free: Amount of free memory.
- Memory cached: Amount cached on disk.
- Memory active: Amount of recently used.
- Memory inactive: Amount of inactive memory that can be freed or cached.
- High memory total: Total amount of high memory: anything above 896MB on 32-bit machines. Not used for 64-bit. [13]
- High memory free: Amount of unused high memory.
- Low memory total: total amount of low memory that the kernel can access directly. [13]
- Low-free: Amount of unused low memory.
- Swap-total: Total usable swap space.
- Swap-free: Amount of unused swap space.

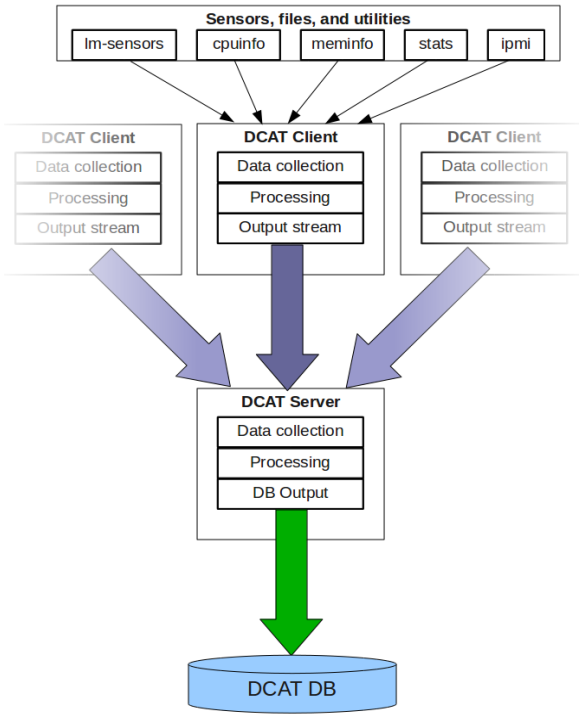

Figure 1. DCAT Architecture Diagram.

Our prototype architecture is composed of a client-server engine, a MySQL database, and communication mechanisms shown in Figure 1. For ease of implementation, the prototype communication layer uses Message Passing Interface. The data is accessed by a customized module for each source. The client processes the data into a structure and pushes it to the DCAT server for uploading into the database.

DCAT runs on an arbitrary number of nodes, with the nodes split up into separate groups. Each group has a server process that collects sensor data from all the nodes in its group. The server process in each group uploads the collected data to a database that resides on another system. The client nodes are configured to push data at the same interval as the server process uploads data. The main reason for the current configuration is the fact that our compute nodes do not all have local disks installed.

We have been able to collect data from our testbed for over a year. However, a major drawback of the implementation is the use of MPI: if one node fails, then the entire program fails. This is a common pitfall when using MPI. In order to address this issue, we plan to use sockets for the communication layer in the next implementation. Using sockets, we can implement the same group architecture, but check for a reply when sending data to the server process. If the client nodes cease to receive a reply from the group server, the next node spawns a server process to take over.

In about one month of data collection on a 32-node cluster, we have obtained about 1.5 GB of data, the bulk of it consisting of time-series at one minute intervals. Given the data model structure, static data, such as IP addresses, names and hardware descriptions, are collected only once. With twenty-two sensor values and eight memory statistics values per node, this amounts to 30 million sensor values and one million memory statistics values. No log data is presently collected by our system except for indicating the latest reboot.

Our collection effort showed that the number of temperature and fan values reported by IPMI and lm-sensors depends on hardware implementation. So does location, which may be important for detecting anomalies and error correlation in HPC. The type of sensors a platform is instrumented with also depends heavily on hardware configuration. For instance, although our platform was IPMI-compliant, and reported some metrics through IPMI, we were unable to collect voltages, which others have collected with IPMI. Our collection module collects temperatures and fan data data every minute.

Our example implementation has illustrated some of the challenges encountered while collecting metrics related to system health. To better understand what is actually available for system instrumentation and health monitoring, we now review available data sources.

4. Data sources and standards for sensor data representation

4.1. Data sources

Linux monitoring sensors (lm-sensors) is a well-known open source utility for sensor data acquisition. Starting with Linux kernel 2.5, lm-sensors are packaged with most distributions. lm-sensors report temperature, voltage, and fan rpm data. The utility depends on the community to write drivers for boards and platforms, which can be slow for new models. Metadata in lm-sensors is somewhat explicit, with variable names such as VCore (voltages with a minimum and maximum), CPU Fan 1, Front Fan 3, coretemp Core 0, coretemp Core 1, etc. A driver for IPMI exists but it was more efficient to capture data at the source.

IPMI is an open-source specification sponsored by Dell, HP, NEC, Intel, and implemented in proprietary tools and an open-source tool. Its main engine, the Baseboard Management Controller, coordinates messages from on-board sensors and communicates Sensor Data Records to the Sensor Data Repository. [14] IPMI operates at the BIOS level, and is intended

for integration into system management tools. Many manufacturers (but not all) have adopted IPMI and deploy their sensors with IPMI drivers. IPMI essentially reports temperatures, voltages, and fan rpms, depending on the platform. For those interested in using IPMI data, without system management tools, attributing measurements to a particular temperature sensor or fan on a board is a matter of guess work, since IPMI returns data such as Temp 1, Temp 2, Temp 3, Fan 1, Fan 2, Fan 3, etc. The sensors listed in Section 3 for our testbed were provided through a long chain of emails to a direct contact at the manufacturer. IPMI is not supported by Cray, Inc who utilizes the Cray RAS and Management System. IPMI reports temperatures as integers in Celsius, which is not quite sensitive enough to detect rapidly rising trends and allow enough margin for corrective action.

For our case-study, we also procured Sun Small Programmable Objects (SunSPOTS) [15] and instrumented the room where our testbed is located. SunSPOTs are self-contained sensor devices, powered by mini-USB and communicating with a base station through a radio connection at 2.4 GHz. SunSPOTs operate in unlicensed bands at high frequencies, on channels 11-26. A SunSPOT base-station exports its data to the system where it is mounted via USB port. Data communication between a SPOT and its base is encrypted. By default, SunSPOTS are equipped with a light and temperature sensors. By placing them at strategic locations, for instance under a power unit, or on top of a switch, it is possible to obtain temperature data at the location, thus providing a basis for comparison with other methods such as IPMI or lm-sensors. SunSPOTs claim a range of 70 meters between base station and SPOT, but due to the strong interfering frequencies in the machine room, approximately thirty feet was the maximum range we obtained. As of version 6.0, SunSPOTs do not support 64-bit, which can make it inconvenient to use. For SunSPOT data, we use the SunSPOT API to upload data to a database residing on a 32-bit system hosting the base station.

Our investigation of data sources for system health metrics showed that these are system-dependent and that data representation must be tailored to the source. In the following section, we present our investigation of open source sensor data representation in communities other than HPC where sensors are used, in order to find standard representations.

4.2. Standards for sensor data representation

We have examined in details open source standards for sensor information processing describing sensor data information in [16]. The proliferation of sensor manufacturers and usages in many fields of science and industry have raised multiple challenges for data collection and processing. When data fusion from heterogeneous sensors and interoperability are desired, standards become advantageous.

We hereby discuss standards and specifications developed by the IEEE, the Open-Geospatial Consortium (OGC), the Distributed Management Task Force (DMTF), and the Intelligent Platform Modeling Initiative (IPMI), an Intel-led effort at standardizing hardware-related metrics.

IEEE 1451 is a specification enabling data transfer to a network and the remote operation of sensors [17]. These “smart sensors” transfer data about themselves and their payload using Transducer Electronic Data Sheets (TEDS). TEDS are generic data structures, four TEDS per service are required while optional TEDS are also available. Details on this standard's architecture are available in [16 and 17]. IEEE1451 contains instructions that sensor manufacturers can use for standardizing their data transfer protocols. IEEE1451 also defines four Application Programming Interfaces (API) using the HTTP1.1 protocol. Thus an IEEE1451-compliant sensor provides data directly usable by other applications that conform to the IEEE 1451 model or to applications communicating by http. However, sensor manufacturers for HPC environments do not use IEEE 1451, and connectivity between nodes does not use Internet protocols.

The OGC standards, including Sensor ML, Transducer Model Language, and Sensor Web Enablement, contain very rich metadata for characterizing sensor data, with a special emphasis on location due to its origin in the geo-spatial community. [18] SensorML describes sensor systems, processing algorithms and workflows and can encode the on-demand execution of algorithms for remotely controlling sensors. [19] However, OGC sensor models are also designed for sensor data and command transfer over the Internet.

The DMTF [20] is an industry organization leading the development of management standards and infrastructure components for instrumentation, control and communication in Enterprise and Internet environments. In particular, the Common Information Management (CIM) and Common Diagnostic Management specifications [21] contain many entities that can be used for a data model, including classes for metrics, and events, and profiles for sensors and power management. CIM and other DMTF technology was designed to unify the management of information for desktops, servers, storage, communications, and data centers. The open source Standards Based Linux Instrumentation for Manageability [22] has been

developed by IBM around the CIM standard but it relies on Web-based Enterprise Management.

5. Taxonomy

Following the example of the OGC Sensor Web Enablement effort, we designed a data model based on the abstraction of a "sensor". In SensorML, a sensor is a process that converts real phenomena into data. Every data point is referenced to a "sensor," which can be an actual sensor like a thermistor, a physical object like a fan, or a process that collects system state data like CPU usage. This abstraction is advantageous to our purpose because a process is a basic unit in computing and the output of a process is data. We use sensor to represent temperature sensors, fans, voltmeters, but also CPU, where the data is the amount of CPU used at a particular time point.

Our data model includes five objects: sensor, system, measurement, memory statistics, and unit. The measurement object contains data from all physical parameters and includes CPU usage. The content of this object is highly dynamic. The memory-stats object contains values related to memory. This object contains both static and dynamic data. Because of the amount of static information related to memory (as seen in Section 3), we did not treat it as a sensor and kept it separate. Both objects have a time stamp. The sensor, system and unit objects contain static data that change only if the physical configuration of the system changes. Preserving system-state and configuration parameters are needed for forensic root-cause analysis of failures.

6. Conclusion and Future Work

We presented our data collection effort for a 32-node, quad-core testbed to illustrate the heterogeneity of data sources reporting on system-health in HPC and the lack of standards in data representation. We found that even for platforms that purportedly comply with a widely-used manufacturer specification (IPMI), instrumentation does not report all the metrics that it claims to report. We also found that the description of the reported data (the metadata) can be very thin and that the units can be too coarse for spotting trends (integer delta in Celsius).

We investigated sensor data representation standards used in other fields with the purpose of finding entities and/or concepts that could be used in HPC. We found that the abstraction of "sensor as process" designed for SensorML to be advantageous but insufficient to hold all the needed metrics.

Future work includes the study of the metrics and event classes in the CIM specifications with the goal of testing them for HPC. The addition of system logs to our data collection may also require adapting our taxonomy.

6. References

[1] F. Cappello, "Fault Tolerance in Petascale/Exascale Systems: current knowledge, challenges and research opportunities," The International Journal of High Performance Computing Applications, Vol. 23, No. 3, Fall 2009, pp. 212-226.

[2] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir, "Toward Exascale Resilience," The International Journal of High Performance Computing Applications, Vol. 23, No. 4, Winter 2009, pp. 378-388.

[3] A. Oliner and J. Stearley. "What supercomputers say: a study of five system logs," in Proceedings of the International Conference on Dependable Systems and Networks. (DSN 2007).

[4] B. Schroeder and G. Gibson, G. "Understanding failures in petascale computers," Journal of Physics: Conference Series, Vol. 78, (SciDAC 2007).

[5] J. Brandt, A. Gentile, J. Mayo, P. Pébay, D. Roe, D. Thompson, and M. Wong. "Resource Monitoring and Management with OVIS to Enable HPC in Cloud Computing Environments ," IEEE International Symposium on Parallel & Distributed Processing, (1-8), 2009.

[6] J. Brandt, B. Debusschere, A. Gentile, J. Mayo, P. Pébay, D. Thompson, and M. Wong, "Using Probabilistic Characterization to Reduce Runtime Faults in HPC Systems," Proceedings of the Eighth IEEE International Symposium on Cluster Computing and the Grid, (759-764). 2009.

[7] Ganglia Monitoring System. Available March 1, 2010. http://ganglia.sourceforge.net.

[8]Smartmontools. Available March 1, 2010. http://smartmontools.sourceforge.net.

[9] J. Stearley, "Towards a Specification for Measuring Red Storm Reliability, Availability, and Serviceability (RAS),", Cray Users Group Conference (May 2005).

[10] J. Stearley, "Defining and Measuring Supercomputer Reliability, Availability, and Serviceability (RAS)," DARPA HPCS Team & PI Meeting, 2005. http://www.cs.sandia.gov/~jrstear/ras.

[11]LM sensors. http://www.lm-sensors.org/wiki and http://www.lm-sensors.org/wiki/Documentation. Available March 1, 2010.

[12] IPMI - Intelligent Platform Management Interface Specification Second Generation V2.0. Document Revision 1.0. February 12, 2004. http://www.intel.com/design/servers/ipmi/. Available March 1, 2010.

[13] D. P. Bovet and M. Cesati, "Understanding the Linux Kernel, 3rd ed." Sebastopol: O'Reilly Media, 2005.

[14] H. Zhuo, J. Yin, A. V. Rao, Remote management with the baseboard Management Controller in Eight Generation Dell PowerEdge Servers. Dell Power Solutions, 2004.

[15] SunSPOTS. http://www.sunspotworld.com/ Available March 1, 2010.

[16] L. C. Pouchard, S. Poole, J. Lothian, and C. Groer, *Open Standards for Sensor Information Processing*, Oak Ridge National Laboratory, Oak Ridge, TN, Tech. Rep. ORNL/TM-2009/145, 2009.

[17] IEEE Instrumentation and Measurement Society. Technical Committee on Sensor Technology (TC-9). "IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats (IEEE1451.0.2007)".

[18] Botts, Mike, Carl Reed, George Percivall, John Davidson. "OGC Sensor Web Enablement: Overview and High Level Architecture," Proceedings of the 5th International Information Systems for Crisis Response and Management (ISCRAM) Conference. F. Friedrich and B. Van de Walle, eds. Washington, DC: (May 2008).

[19] Botts, Mike. "Sensor Model Language (SensorML) Details." Earth System Science Center, UAB Huntsville. September 2007. http://schemas.opengis.net/sensorML/.

[20] Distributed Management Task Force, Inc. http://www.dmtf.org. Available March 1, 2010.

[21] Common Information Model (CIM). http://www.dmtf.org/standards/cim. Available March 1, 2010.

[22]Standards Based Linux Instrumentation. http://sblim.wiki.sourceforge.net. Available March 1, 2010.

Acknowledgments:

This work was supported by the Extreme Scale Systems Center at Oak Ridge National Laboratory. The submitted manuscript has been authored by a contractor of the U.S. Government under Contract No. DE-AC05-00OR22725. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes. Special thanks to Josh Lothian and Chris Groer from ORNL for their help with the testbed