A Database-centric Approach to System Management in the Blue Gene/L Supercomputer

Ralph Bellofatto**, Paul G. Crumley**, David Darrington*, Brant Knudson*, Mark Megerian*, José E. Moreira**, Alda S. Ohmacht**, John Orbeck*, Don Reed*, Greg Stewart*

*IBM Systems and Technology Group
Rochester, MN 55901
{ddarring, bknudson, megerian, orbeck, donreed, gregstew}@us.ibm.com

**IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598
{ralphbel, pgc, jmoreira, sanomiya}@us.ibm.com

Abstract

In designing the management system for Blue Gene/L, we adopted a database-centric approach. All configuration and operational data for a particular Blue Gene/L system are stored in a relational database that is kept in the system's service node. The database also serves as the communication bus for the various processes implementing the management system. This design offers many advantages, including the ability to use SQL commands to retrieve reliability, availability, and serviceability (RAS) information about the system. Information about machine partitioning and user jobs can be obtained the same way. Leveraging the database, we have developed a web interface for system management. This management system has been successfully implemented and deployed in all 19 Blue Gene/L installations at the time of this writing.

1. Introduction

One of the most innovative aspects of the Blue Gene/L supercomputer is its management system. When we set out to build and deliver an operational supercomputer with over 100,000 processors, we were aware of the challenges. Previous experiences with parallel systems had more or less saturated around 10,000 processors (*e.g.*, ASCI White, ASCI Q, QCDSP, and Earth Simulator).

Instead of extending existing management systems [2],[3],[4],[5],[6] capabilities by an order of magnitude (and some of them certainly could), we decided to start from scratch with a new design. Part of the motivation of starting from scratch had to do with unique hardware characteristics of Blue Gene/L. Another part of the motivation had to do with the simple human desire of doing something new. We had the luxury of the latter because Blue Gene/L was a research project with essentially a single customer (Lawrence Livermore National Laboratory) deliverable.

The ultimate goal of the Blue Gene project has always been scalability. We were going to build a machine with at least 65,536 dual-processor compute nodes and another 1,024 dual-processor I/O nodes. Simplicity in hardware design was fundamental to achieve that scalability. In particular, there is almost no persistent storage in Blue Gene/L: no nonvolatile memory for the processors and no disk drives. There is also no service processor in the machine, which is a big departure from how large computers (and these days, small ones too) are usually built.

Blue Gene/L was designed to be controlled and used from the outside, not too different from what happens in distributed grid environments [1]. Neither system administrators nor application programmers ever "login" to any of the nodes of the machine. System administrators do their job from a *service node*, a host computer that performs the functions of control and monitoring of Blue Gene/L. Application programmers do their work from *front-end nodes*, computers that are used to compile, build, launch, and debug application programs on Blue Gene/L. Figure 1 shows a high-level view of a complete Blue Gene/L system with its computational core (compute and I/O nodes), service node, front-end nodes, and file servers.

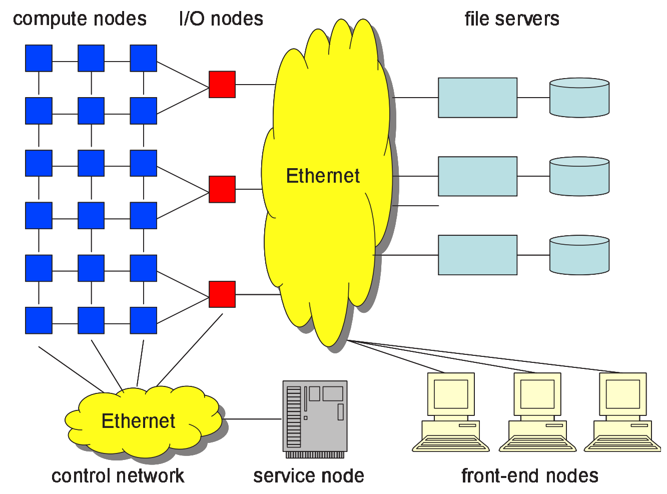


Figure 1: A complete Blue Gene/L system includes (1) the core machine with compute and I/O nodes, (2) a service node for control and monitoring of the core, (3) front-end nodes for users to build and launch their jobs for execution, and (4) file servers for a global file system. The I/O nodes, service node, front-end nodes and file servers are all interconnected by a high-speed 1-Gbit Ethernet fabric. The service node controls and monitors the core through a separate Ethernet control network.

This paper focuses on the solution we developed for system management of Blue Gene/L. We organized the system management software as a set of processes with specific functions. Those processes communicate though and store their persistent state in a relational database.

The idea of using a relational database as the system data repository has been considered before (*e.g.*, RS/6000 SP). It was dismissed because relational databases were not considered to be fast enough at the time. With 10 years of performance improvements, we were confident that relational databases were now ready for the task. To our knowledge, the Blue Gene/L management system was the first to use a relational database as its *communication bus*. The database is both a repository of data and the means by which the various processes communicate with each other.

The management system described in this paper was fully implemented and deployed in all Blue Gene/L installations, varying in size from 1 to 64 racks (1,024 to 65,536 compute nodes). Based on our own experience with operating those systems, as well as on positive feedback from our customers, we know we have succeeded in delivering a highly scalable and effective management system for the most powerful computer in the world.

The rest of this paper is organized as follows. Section 2 discusses the architecture (or schema) of the Blue Gene/L management system database. Section 3 presents the various processes that constitute the management system and discusses their respective functions. Section 4 describes how the database is populated through a process of hardware discovery. Section 5 explains how machine partitions are created and then monitored by the management system. Section 6 describes how jobs are launched for execution in Blue Gene/L. Section 7 discusses repair operations in Blue Gene/L. Section 8 presents a web-based system administration console that was developed for ease of use. Finally, Section 9 presents our conclusions.

2. The Blue Gene/L management database

The Blue Gene/L management system database is implemented using the IBM DB2 relational database product. The data in that database is organized into four separate "sub"-databases. All tables of the database belong to a single schema. This separation into four components just helps us understand the database better. The four sub-databases are shown in Figure 2. The configuration database contains a description of all the hardware in the system. The operational database contains information about partitions of the machine and jobs executed on those partitions. Both current and historical data are stored. The environmental database keeps current and historical values for measurable data from all the hardware components of the system, including things like fan speeds, power supply voltages, and chip temperatures. The RAS database stores information on all failure events observed in the Blue Gene/L core. Some events are detected by hardware; others are detected by system software. We discuss the contents of each of these sub-databases in more detail.

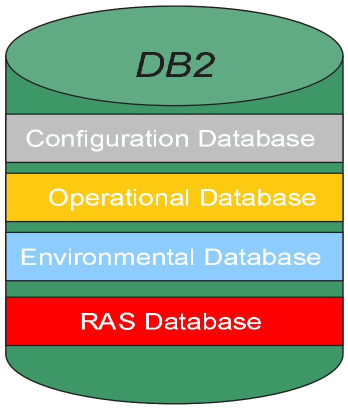


Figure 2: Logical organization of the management system database.

2.1. The configuration database

The configuration database represents the hardware following a hierarchy that matches the physical packaging of the machine. A Blue Gene/L machine consists of a collection of midplanes interconnected by cables. In addition to midplane and cable tables, the database contains a table that describes which cables interconnect which midplanes. In fact, the representation goes down to the detail of individual *ports* (see below when we talk about link cards).

A midplane consists of a service card and a number of link cards, node cards, and fan assemblies. Every card (service, link and node) has its own *IDo chip*. An IDo chip is an FPGA that allows the card to be controlled from an Ethernet network. Each IDo chip has its own IP address. Each link card contains six link chips and implements a total of 16 ports, used to connect cables. A cable always connects one port in a midplane to another port in another midplane.

A node card always contains (if complete) 16 compute cards and 0, 1, or 2 I/O card. A compute card contains two compute nodes and an I/O card contains two I/O nodes. Each node has a certain amount of memory, which in all Blue Gene/L systems produced so far is always 512 MiB.

2.2. The operational database

The operational database contains information about partitions and jobs in the system. A given partition can be either a subset of a midplane or formed by multiple midplanes. Sub-midplane partitions can have either 32 or 128 nodes. Multi-midplane partitions are always organized as a rectangular arrangement of midplanes. To fully describe a multi-midplane partition, it is also necessary to specify the settings of the link chips in those midplanes (and possibly other midplanes), since the link chips control how midplanes are connected together. The operational database also contains information about which users are authorized to run jobs in a particular partition.

The operational database also stores information about the jobs currently running on the system, including on which partition it is running, its submission and start times, the job owner, its executable file, its parameters and environment variables, and its starting working directory. When a job finishes, the job information is moved into a history table.

2.3. The environmental database

As we mentioned before, the environmental database contains current and historical data on measured values for all hardware components. In particular, the environmental database contains information on:

- *Fan modules*: Values for both desired and measured speed (RPMs) for each fan module. Motor voltages and air temperature in each fan module.
- *Service cards*: Values for air temperature, chip temperatures, and supply voltages in each service card.
- *Node cards*: Values for chip temperatures (these are dedicated temperature chips), accepted temperature limits, supply voltages, and wiring faults for each node card.
- *Link cards*: Values for chip temperatures and supply voltages for each link card.

2.4. The RAS database

The RAS database collects information regarding all errors and failures detected in the Blue Gene/L core (compute and I/O nodes). The collected information includes:

- Bad and/or missing hardware detected during the discovery process that populates the database. Examples include bad memory, broken cables, missing compute and/or I/O cards.
- Events generated by the kernels in an active machine partition. For example, memory failures (both recoverable and nonrecoverable), communication failures, illegal memory access by an application.
- Events generated from monitoring the behavior of the hardware. For example, bad fan speeds, wiring faults, over temperatures, improper supply voltages.
- Failures detected by the system when initializing a partition. For example, link chips that do not work properly, file systems that do not mount, software failures.

We note that the history of RAS events can be used to control system parameters. For example, the history of memory errors in a particular node is used at the next initialization operation involving that node to program the memory controller to bypass bad memory chips.

Our four-rack Blue Gene/L development system generated 690,000 hardware and software RAS events over a six-month period. That works out to approximately 1000 RAS events per rack per day. Even when we multiply that number by 64 (for the LLNL installation) we come up with a database traffic that is well within the limits of what medium-size (e.g., 16-way SMP) DB2 servers today can handle, and validates our choice of DB2 as the system database infrastructure for Blue Gene/L.

3. The Blue Gene/L Management Processes

The Blue Gene/L management system is implemented by a collection of processes, each with a dedicated function. A graphical representation of the processes and their interrelationships is show in Figure 3.

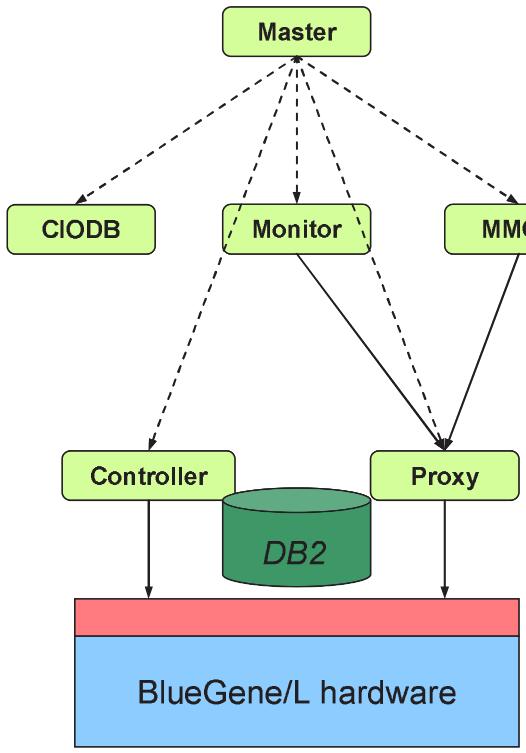


Figure 3: The collection of processes that implement system managemente in Blue Gene/L.

The *Master* process is responsible for creating and monitoring all the other processes. If one of the other processes dies, Master will restart it and log the event in a system log. The other processes are as follows:

1. *Controller*: Performs discovery and first initialization of all the hardware in the Blue Gene/L core. It stores the description of the hardware in the database.
2. *Proxy*: Once hardware is discovered, the IDo chips in every service, node, and link card must be *proxied*. All communication to those chips is performed through this process, as identified in the database.
3. *MMCS*: Creates and boots machine partitions. It also monitors RAS events coming from the system software running on those partitions. MMCS finds the description of the partitions it has to boot, and stores their state and RAS events, in the database.
4. *Monitor*: All environmental data in the system (voltages, temperatures, fan speeds, etc) are collected by this process and stored in the database.
5. *CIODB*: Launches jobs in partitions and redirects their standard input/output/error. It finds the jobs it has to launch, and stores their state, in the database.

All of the above 5 processes interact with the database, which interconnects the processes together into a unified management system.

4. Populating the Database

The configuration sub-database is populated automatically by the Controller process. The Controller is responsible for initial hardware setup and in the process it "discovers" all the hardware that comprises a Blue Gene/L core. (For simplicity, we have abstracted all the functions into a single Controller process. In reality there are several processes, each performing a specific function.)

This *discovery* process starts with the service cards in each midplane. Because the service cards are daisy-chained, once a service card is discovered and initialized, the Controller can then find the next service card in the daisy chain. It also finds the node cards, link cards, fan modules, and power supplies of that midplane. Once the Controller can talk to each node card, it finds the compute and I/O nodes in that node card. The Controller performs a detailed inspection of each node, finding, for example, how much memory it has and if that memory is working properly. All this information is stored in the database. As the Controller finds and initializes service, link, and node cards, it passes them over to the Proxy process, so that future communications with those cards go through the Proxy.

The setup phase of the hardware concludes with cable discovery. In that process, the Controller finds out how the multiple midplanes in a machine are connected through cables. It discovers the topology of the machine. It also identifies broken cables and certain misconnections. The topology information is stored in the database, so that it can be used by an external scheduler as it decides on how to partition the machine.

5. Creating and Monitoring Partitions

One of the main responsibilities of the Blue Gene/L management system is the creation and monitoring of machine partitions. The MMCS process is responsible for the management of partitions. Partitions must be described in the database, before they can be created or activated (that is, brought to an operational state) for running jobs. A partition is described in the database through tables that store its properties. The tables describe the physical hardware (midplanes and node cards) used in each partition. The same hardware component can belong to many partitions in the database, but it can be present in only one active partition at any given time. (No hardware sharing among active partitions in Blue Gene/L.) The database also stores information on the partition name, its geometric shape, the ratio of compute to I/O nodes, the full path to the kernels and other system software running on the partition, the time the partition was created, the name of its owner, the name of users that are authorized to running jobs on the partition and, last but not least, the current state of the partition.

Through its life time, a partition goes through a sequence of states, shown in Figure 4. Partitions always start in the "Free" state. That means the partition is defined in the database, but it is not active. Partitions start their active life when they are moved in either the "Configure" or "Allocated" states. Partitions can be moved into one of those states by the system administrator or by an external scheduler. There are differences between those two states, regarding what is exactly done to activate the partition, but they both start the process. Ultimately, the goal is to reserve the appropriate hardware for the partition (so that other partitions will not conflict), establish communication with that hardware, initialize it, and start the loading of the kernels and other system software in the compute and I/O nodes of the partition.

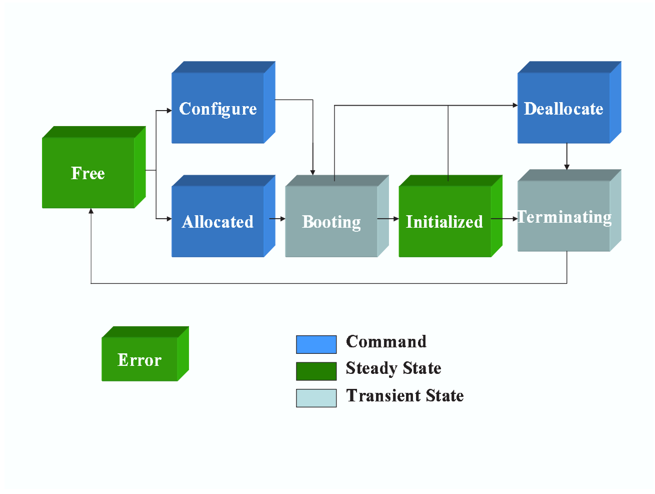


Figure 4: The life time of a partition. There are three stable states for a partition: "Free" which means not active, "Initialized" which means booted and ready to run jobs, and "Error" which means an intervention is necessary before the partition can be used again. The other states are used to move the partition between "Free" and "Initialized".

Once the system software for the partition is loaded, the MMCS process waits for the booting process of the nodes in that partition to complete. During that time, the partition is in "Booting" state. Once all compute and I/O nodes are running and the appropriate file systems have been mounted, the nodes in the partition inform MMCS, which then moves the partition in "Initialized" state. It is while in that state that the partition is truly active and ready to run user applications. A partition can be terminated either through an external request (by administrator or scheduler) that sets its state to "Deallocate" or because of some internal event that directly moved the partition into "Terminating" state. After termination, a partition moves back into "Free" state.

A partition can also move into an "Error" state. This is caused when unrecoverable errors happen during the partition life time. For example, if the partition cannot be booted (because of hardware or software problems), it is moved to "Error" state, which prevents it from being used again until there is an administrator intervention.

MMCS keeps monitoring all partitions in "Initialized", "Booting", and "Terminating" states for events generated by the system software in the I/O and compute nodes. Some events are generated in response to hardware issues (*e.g.*, memory errors, communication errors), while others are generated in response to application behavior (*e.g.*, invalid memory reference, I/O error).

6. Launching and Running Jobs

The running of jobs in Blue Gene/L is the responsibility of the CIODB process. Only partitions that are active (*i.e.*, in the "Initialized" state) can run jobs. Furthermore, only users that are in the list of authorized users for a partition can submit jobs to that partition. The idea is that an scheduler (or system administrator) can control allocation of partitions to users by manipulating the user list.

Jobs must be described in the database in order to be executed. The job entry in the database contains information such as the full path to the executable (Blue Gene/L only supports SPMD model), command line arguments and environment variables for the job, the name of the partition where the job is being (or will be) executed, the name of the user who submitted the job, redirection information for `stderr` and `stdout`, the start working directory for the job, and the current state of the job.

Like partitions, jobs go through a sequence of states in their life time. That state machine for a job is shown in Figure 5. Jobs always start their life in the "Queued" state.

In that state, the job entry is in the database, with the information to run the job, but it has not been started yet. In order to start running, jobs then go through a "Loading" state, in which they are loaded by the I/O nodes from the file system into the compute nodes. Depending on the mode of execution, jobs can stay in "Loaded" state (ready to run but not running) or move directly to "Running" state, which means running on the compute nodes. It is useful to have a "Loaded" state so that debuggers can be attached to a job before it actually starts running. A debugger can also attach and detach to running jobs. Running jobs can receive signals. A job that finishes running, either voluntarily or through of an invalid operation or a hardware/software failure, moves into the "Terminated" state.

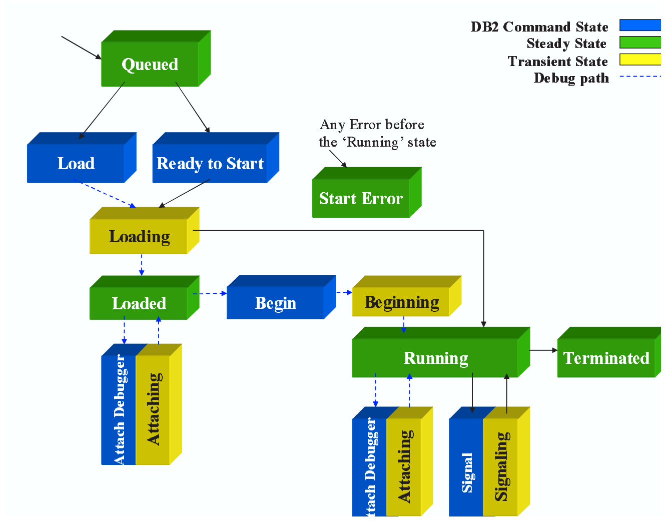


Figure 5: Life time of a job. The stable states are in green: "Queued" (not running), "Loaded" (ready to run but not yet running), "Running" (in the compute nodes of a partition), and "Terminated" (job has finished, either voluntarily or not). Blue states are states to signal that an action on the job is required. Yellow states are transitional states while those actions are being performed.

Terminated jobs are kept in the database for future queries and analysis, but they are kept in a separate table, the *job history table*. Therefore, programs that monitor and do accounting of jobs must query both the current job table and the job history table.

7. When Repair is Necessary: Service Actions

Service actions are procedures performed on a Blue Gene/L machine to bring broken hardware back into operational state. Service actions usually involve hardware replacement and need to be coordinated through the management system.

Most hardware repair in Blue Gene/L involves replacing a node (I/O or compute) in a node card. We use this operation as an example of a service action. First, the job (or jobs) running on the midplane that will receive the action must be terminated, and future jobs should not be started on that midplane. That action requires coordination and collaboration between the CIODB and MMCS processes, and results in the midplane being marked down in the database. The corresponding node card is also removed from the list of hardware being proxied. That way, no one tries to talk to it during the service action. Finally, the node card is powered down, so that it can be removed from the midplane.

The node card is then removed from the midplane and the appropriate compute or I/O node is replaced. The node card can then be inserted back in the midplane. To complete the service action, the node card and its nodes must be reinitialized. Furthermore, since hardware was changed, it is necessary to perform discovery of that node card again and to update the database. Finally, the card is moved back to the Proxy, and the midplane is again set to an operational state. Use of that midplane can then continue normally. The entire service action for a node card, from killing the running jobs to restoring the midplane to operation, can be done in about 15 minutes.

Link card replacement follows the same general process. However, the effects are further reaching, as a link card impacts not only its midplane but the midplanes it connects to. Also, replacing a link card requires detaching and reattaching lots of cables, so it is a more lengthy procedure.

8. Web Console for Ease of Use

Initially, there was not a lot of emphasis on administration tools, ease-of-use, or GUIs for Blue Gene/L. The initial single customer (LLNL) was closely involved in the direction and design of the system management software. Therefore, a steep learning curve was not a concern. The install of the entire machine would occur over many months, and LLNL would be learning the inner workings of Blue Gene/L throughout this whole process, beginning in late '04 and extending through most of 2005. By the time the final install and acceptance was complete, the team at LLNL was very familiar with the management of the system, hardware maintenance, and the database.

But as the customer base for BG/L has grown, we acknowledged that new customers expect more tools and an easier process of learning how to use the machine. For this reason, we have focused on simplifying the administration of the Blue Gene/L system.

While some system administrators will always prefer the direct command line text interfaces, we have added a browser-based web console that allows Blue Gene system administrators to do most of their tasks from a web browser. This *Blue Gene Navigator* console consolidates many of the system management tasks into a single place. The Blue Gene Navigator allows the system administrator to see all jobs that are currently running, all jobs that have completed, all partitions defined in the database, and all partitions currently active. From the Navigator, they can submit jobs, kill jobs, boot and free partitions, run system diagnostics, view environmental data, and view all of their system logs. All of these tasks have been available in the past, but only via a text-based terminal session on the service node, and in some simple, display-only web pages. We gathered feedback from our customers, and took into account the areas that they found most difficult to learn and most cumbersome. Chief among those was (1) the submission of system diagnostics; (2) searching through system logs, some of which had inconsistent data formats; (3) querying the RAS database; and (4) quickly identifying potential hardware anomalies.

Using this information, and bringing in IBM experts on user interfaces, as well as comparing successful IBM products already on the market, we developed an interface that any system administrator could use and immediately be productive. A screen shot of that user interface is shown in Figure 6. Graphs at the bottom of the page always show the machine utilization, RAS events, and number of submitted jobs, plotted by day. A banner at the top shows when there is any immediate attention required, and if so, shows the area of the problem. A navigation panel on the left shows the following functions:

1. *Health Center*: At-a-glance display of any hardware that is currently in an error state, and its exact location.
2. *MMCS Console*: A direct console to the MMCS server, with all MMCS commands available. This console constitutes the low-level access to run the machine.
3. *RAS database*: Simple query interface for the RAS events, as an alternative to using SQL queries.
4. *Diagnostics*: Allows both the submission of hardware diagnostics, and the display of results.
5. *Blocks*: Shows all partitions defined, and currently allocated.
6. *Jobs and Job History*: Shows currently running jobs as well as job history.
7. *Logs*: An interface for viewing all system-generated logs, with searching, highlighting, auto refresh, and many other features.
8. *Utilization*: Shows full machine utilization percentage, with user-specified timeframe, also shows broken down by midplane.
9. *Hardware Browser*: Allows the users to familiarize themselves with the hardware, seeing the layout of racks, midplanes, and cards.
10. *Environmentals*: Allows simple queries of environmental data.
11. *Blue Gene/L Redbooks*: A link to all the IBM provided documentation.

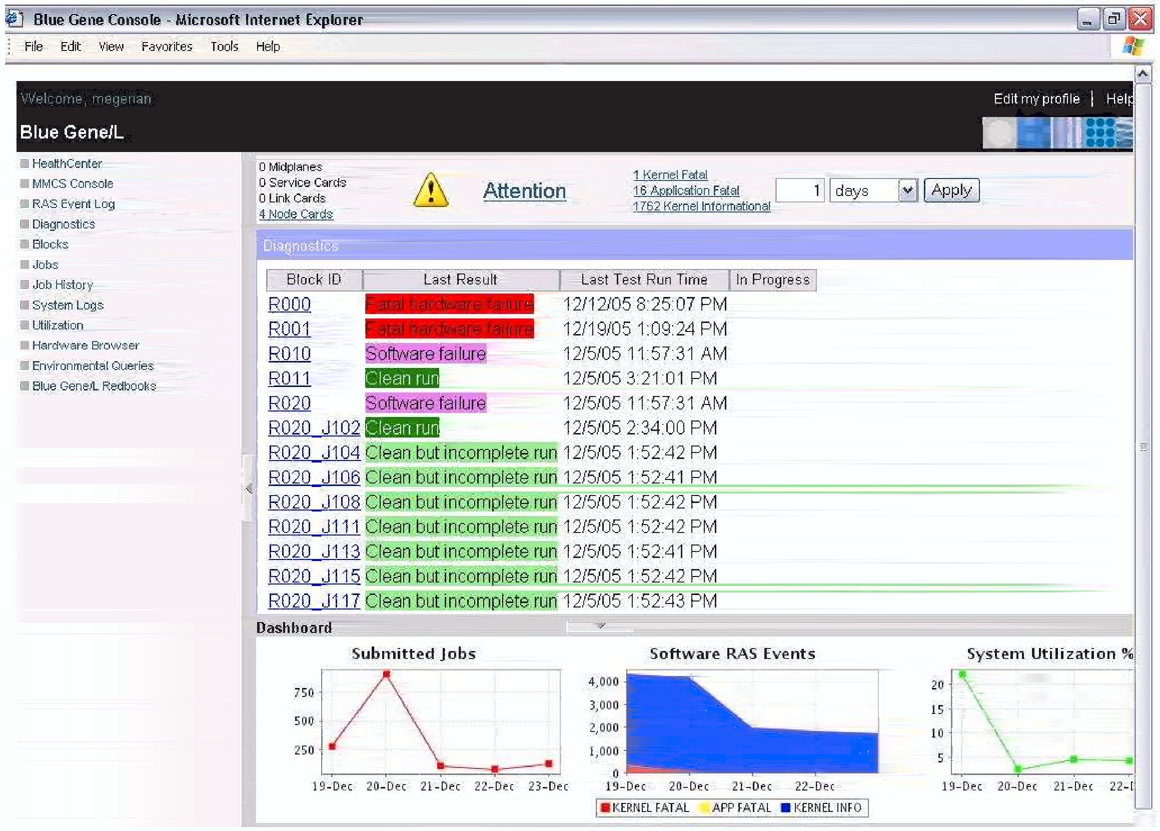
The web console runs directly against a Tomcat web server on the service node. The web server uses the system management database in that node to obtain the necessary data, and then formats the data for a web browser. The use of the Navigator requires a profile on the service node and is intended as an administrator tool. End users primarily compile and submit programs from front-end nodes, and interact with the machine either via a scheduler or `mpirun`.

9. Conclusions

This paper is an overview of the database-centric management system for the Blue Gene/L supercomputer. The database is both the repository for system information and also the communication bus for the processes that implement the management system functionality. The database is organized around four categories of data: (1) Configuration data, which describe the components of the Blue Gene/L computational core; (2) Operational data, which has information about system partitions and user jobs; (3) Environmental data, which stores the values of environmental variables like temperatures and voltages; and (4) RAS data, with a list of failure events.

We have also described how the various processes of the management system interact with the database (and with each other) in order to implement functionality to (1) populate the database; (2) create and manage machine partitions; (3) run user jobs; and (4) perform hardware repairs. We also discussed a new web-based interface for system management that is designed to simplify the life of system administrators.

The effectiveness of the database-centric management system for Blue Gene/L has been demonstrated in the field. It is has been used successfully in the 19 Blue Gene/L systems that are list in the TOP500 [7] list of the fastest computers on the planet, including the 65,536-node flagship system at Lawrence Livermore National Laboratory.



0 Midplanes
0 Service Cards
0 Link Cards
4 Node Cards

1 Kernel Fatal
16 Application Fatal
1762 Kernel Informational

Block ID	Last Result	Last Test Run Time	In Progress
R000	Fatal hardware failure	12/12/05 8:25:07 PM	
R001	Fatal hardware failure	12/19/05 1:09:24 PM	
R010	Software failure	12/5/05 11:57:31 AM	
R011	Clean run	12/5/05 3:21:01 PM	
R020	Software failure	12/5/05 11:57:31 AM	
R020_J102	Clean run	12/5/05 2:34:00 PM	
R020_J104	Clean but incomplete run	12/5/05 1:52:42 PM	
R020_J106	Clean but incomplete run	12/5/05 1:52:41 PM	
R020_J108	Clean but incomplete run	12/5/05 1:52:42 PM	
R020_J111	Clean but incomplete run	12/5/05 1:52:42 PM	
R020_J113	Clean but incomplete run	12/5/05 1:52:41 PM	
R020_J115	Clean but incomplete run	12/5/05 1:52:42 PM	
R020_J117	Clean but incomplete run	12/5/05 1:52:43 PM	

Figure 6: A screen shot of the Blue Gene Navigator. The main page has an area for potential problems (top), diagnostic results (middle), and history of jobs, RAS events and system utilization (bottom). Access to more information is through the navigation panel on the left.

References

[1] K. Czajkowski, I. Foster, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. Proceedings of the 4th Workshop on Job Scheduling Strategies for Parallel Processing. Orlando, FL. March 1998. Lecture Notes in Computer Science. Vol. 1459, pp. 62-82. http://citeseer.ist.psu.edu/czajkowski97resource.html

[2] IBM Corporation. Rebook: Building a Linux HPC Cluster with xCAT. September 2002. http://www.redbooks.ibm.com/abstracts/SG246623.html?Open

[3] IBM Corporation. Linux Clustering with CSM and GPFS. January 2004. http://www.redbooks.ibm.com/abstracts/SG246601.html?Open.

[4] IBM Corporation. Parallel System Support Programs for AIX (PSSP for AIX). http://publib.boulder.ibm.com/infocenter/clresctr/index.jsp?topic=/com.ibm.cluster.pssp.doc/psspbooks.html.

[5] M.J. Sottile and R.G. Minnich. Supermon: A high-speed cluster monitoring system. IEEE International Conference on Cluster Computing (CLUSTER'02). Chicago, IL. September 2002. http://public.lanl.gov/cluster/papers/.

[6] D. Thain, T. Tannenbaum, M. Livny. Distributed computing in practice: The condor experience. Concurrency and Computation: Practice and Experience (2004). http://www.cs.wisc.edu/condor/publications.html.

[7] University of Mannheim, University of Tennessee, and NERSC/LBNL. TOP500 Supercomputer sites. http://www.top500.org/.