A Flexible Resource Management Architecture for the Blue Gene/P Supercomputer

Sam Miller, Mark Megerian, Paul Allen, Tom Budnik
IBM Systems and Technology Group, Rochester, MN
Email: {samjmill, megerian, pvallen, tbudnik}@us.ibm.com

Abstract

Blue Gene®/P is a massively parallel supercomputer intended as the successor to Blue Gene/L. It leverages much of the existing architecture of its predecessor to provide scalability up to a petaflop of peak computing power. The resource management software for such a large parallel system faces several challenges, including system fragmentation due to partitioning, presenting resource usage information using a polling or event driven model, and acting as a barrier between external resource management systems and the Blue Gene/P core.

This paper describes how the Blue Gene/P resource management architecture is extremely flexible by providing multiple methodologies for obtaining resource usage information to make scheduling decisions. Three distinctly separate resource management services will be described. First, the Bridge API, a full-featured API suitable for fine tuning scheduling and allocation decisions. Second, a light-weight Allocator API for allocating resources without substantial development costs. And lastly, configuring the system into static partitions. Job scheduling strategies utilizing each of the methods will be discussed.

1 Introduction

Blue Gene/P (BG/P) is a petaflop scale system, and the successor to Blue Gene/L (BG/L), which is described in detail in [6]. The main engine of this system is the compute node, where the envisioned petaflop system will contain 73,728 compute nodes. These nodes are interconnected via several networks, most notably a three-dimensional (72x32x32 node) torus. Each compute node runs a lightweight kernel and executes in either SMP or virtual node mode. A user's job runs on a group of compute nodes called a partition. While running a job against the full machine is possible, and certainly desirable for certain types of applications, the real strength of the machine is that it can be subdivided, or "partitioned" into many smaller blocks of nodes.

The resource management architecture is responsible for presenting the BG/P usage information to external job management systems, which use the information to allocate partitions and schedule jobs to optimally utilize the machine's resources. Such an architecture faces several challenges due to the large machine size. Foremost, due to its toroidal interconnect, the system cannot be viewed as a simple fully-connected set of nodes like a traditional cluster. This imposes certain restrictions on the job management software. Most notably, a BG/P system is restricted to partitioning its resources into groups of three-dimensional rectangular sets of contiguous nodes. Other supercomputers utilizing a toroidal interconnection have experienced significant fragmentation due to such restrictions, often resulting in low system utilization and lengthy job queues [3].

Another problem faced by the resource management architecture is supporting a polling or event driven model for providing the usage information mentioned previously. A polling model requires users of the API to periodically obtain a snapshot of the BG/P system state, then make scheduling decisions based on what has changed since the last polling event. In an event driven, or real-time model, users register a handler to be called when an event occurs, such as a partition changing state, or a job completing. These models are discussed in detail in Section 3.1.

The improved resource management architecture is just one of the many new features in BG/P compared to BG/L. The remainder of the changes are outside the scope of this paper, and will be described in detail in the future.

The rest of this paper is organized as follows. Section 2 explains resource management in the context of a BG/P system. Section 3 further explains partition management, and Section 4 explains job management. Section 5 presents various scheduling and job management strategies using the resource management architecture. Section 6 presents some related works, especially those discussing resource manage-

1-4244-0910-1/07/$20.00 ©2007 IEEE.

ment in BG/L. Lastly, Section 7 concludes this paper.

2 Resource Management

Resource management on BG/P is a broad term. It encompasses managing entities such as jobs, compute nodes, partitions, and network switches. Many of these concepts have remained unchanged from BG/L, while some have new features such as the addition of smaller partition sizes shown in Table 1. Partitions containing more than 512 nodes, and configuring their switches are not particularly interesting for the scope of this paper since they remain largely unchanged from BG/L. They have also been extensively described in previous works [2].

Table 1. Partition size support

Size	Compute Nodes	Dimensions
Midplane[2]	512	8x8x8
Half midplane[1]	256	8x4x8
Quadrant[2]	128	4x4x8
Two node cards[1]	64	4x4x4
Node Card[2]	32	4x4x2
Half node card[1]	16	4x2x2

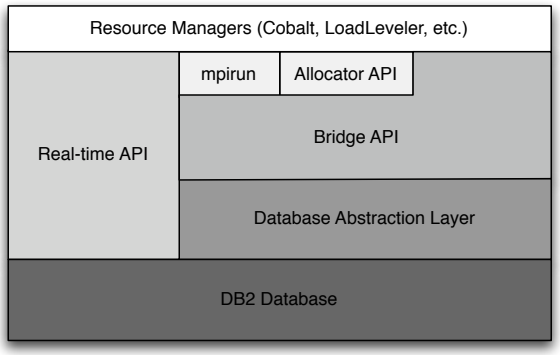

Figure 1. Resource Management API layers

Partition management consists of carving up the entire BG/P machine into smaller chunks of rectangular and contiguous sets of compute nodes. Creating a partition is a necessary step which must be done before a job can run. Figure 1 shows the various layers of the resource management architecture, each of which is explained in detail in Sections 3 and 4.

A brief description of common BG/P entities is included in the Appendix at the end of this paper.

[1] New in BG/P

[2] Supported on both BG/L and BG/P

3 Partition Management

A partition is the central resource entity when dealing with the BG/P resource management system. A partition is a group of nodes allocated to a job. Partitions are physically (electronically) isolated from each other (i.e., messages cannot flow outside an allocated partition). A partition can have the topology of a mesh or a torus. Partitions that are a torus will necessarily be a combination of one or more midplanes. Partitions that are a mesh can be groups of midplanes, but can also be smaller than a midplane (sub-midplane). The term "block" is sometimes used for partition, and the two terms are synonyms within the context of this paper.

3.1 Bridge API

The Bridge APIs provide extensive information regarding the configuration and status of the physical components (midplanes, node cards, wires, switches, etc.) of BG/P. The Bridge APIs also includes a set of functions that allow adding, removing, modifying, or retrieving information about transient entities, such as jobs and partitions. The actual configuration and status data is stored in a database and the Bridge APIs provide the abstract layer that masks the low-level database implementation.

The Bridge APIs can be further subsetted into the following categories:

Resource Manager APIs

The Resource Manager APIs includes a "get" function to retrieve the latest snapshot of the entire BG/P system. This function supplies all the required configuration and status information to allow partition allocation. The information is represented by three lists: a list of midplanes, a list of wires, and a list of switches. A list of node cards can be obtained as well with a "get" node cards function. The set of Resource Manager APIs is rounded out with additional functions for adding, removing, changing and retrieving both job and partition entities.

Job Manager APIs

The Job Manager APIs provide the capability to load, start, signal or cancel a job. Interfaces are provided for debuggers to attach to a job as well. Some of the APIs provided such as starting, signaling or canceling a job are asynchronous and a job scheduler will have to poll a job to determine the current state information.

Partition Manager APIs

The Partition Manager APIs provide the mechanism to boot and free a partition. These APIs are asynchronous and

a job scheduler will have to poll a partition to determine the current state information.

3.2 Real-time Notification APIs

On BG/P, hardware (midplanes, switches, or node cards), jobs, and partitions can go through state transitions. These transitions are shown in the following figures. Figure 2 shows state transitions for hardware, Figure 3 shows state transitions for jobs, Figure 4 shows state transitions for partitions.

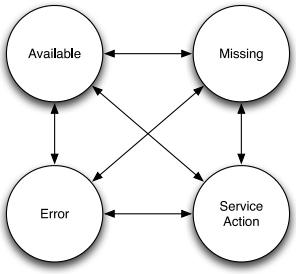

Figure 2. Hardware state transitions

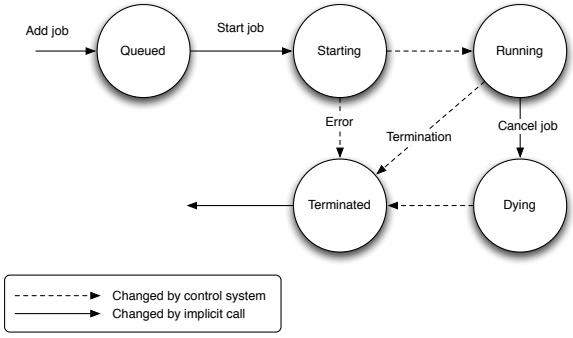

Figure 3. Job state transitions

With BG/L the only way to obtain that latest configuration and status for the machine is to continuously poll for information using the somewhat heavyweight "get" BG/L function that returns the system snapshot. Typically resource managers will do polling every few minutes to see if any hardware has transitioned states (for example from "available" to "error") . Also when waiting for a partition to boot or job to end the only way to determine the status is by constant polling using a Bridge API that returns the latest state information. For example once mpirun starts a job it will poll the job status every 5 seconds to determine if the job has ended.

With BG/P an alternative to polling status for hardware, jobs and partitions will be available. The new real-time notification APIs will allow job schedulers to track state

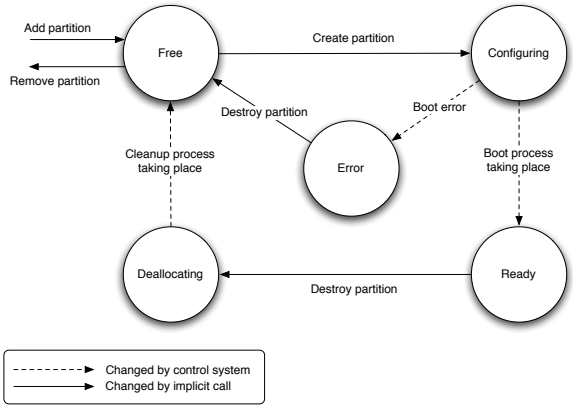

Figure 4. Partition state transitions

changes as they occur. The real-time APIs can be used in conjunction with the existing Bridge APIs to provide an efficient lightweight resource management system. A job scheduler will be able to obtain a hardware snapshot of the BG/P machine using the Bridge APIs and then register with the real-time notification APIs to receive callbacks when a state change occurs for hardware as well as when a partition or job is added, deleted or changes state. The real-time APIs will allow the caller to filter on the callbacks they want to receive as well as filter the partition or job they want notifications on. For example a resource manager could register to receive notifications only when a specific partition changes state. The concept of a sequence id is provided in both the Bridge and Real-time APIs. The sequence id can be used to compare which state is the most recent. For example when getting a full system snapshot the sequence id for midplane R00-M0 might be 925. If a hardware failure occurred on the midplane after the snapshot was taken then the real-time callback sequence id might be 952 indicating that the 952 event is more recent.

The combination of Bridge API described in Section 3.1 and the Real-time API provides an extensive set of APIs for tracking and managing BG/P resources and provides extreme flexibility with both a polling and an event driven model for handling state transitions.

3.3 Allocator API

The Bridge APIs provide a full featured interface for resource management. The Allocator API provides a simpler interface for creating partitions. It was developed as a solution for the resource management developer community for those who do not want to program to the Bridge API.

The Allocator API attempts to find a set of available resources to match the request for a partition of a given shape and/or size.

For partitions that are one or more midplanes in size, the

Allocator API for the BG/P system provides the same level of function and allocation strategy as in the BG/L system.

For sub-midplane sized partitions, the Allocator API for the BG/P system has an enhanced allocation strategy as compared to the BG/L system. In the BG/L system, the allocation strategy used by the Allocator API for sub-midplane partitions is a simple first fit approach. In the BG/P system this strategy is improved upon to use a best fit, or optimal strategy. Resource managers requiring a more sophisticated algorithm are free to use the Bridge API described previously.

3.3.1 First-fit Allocation of Small Partitions

BG/L allowed sizes for sub-midplane partitions of 128 compute nodes (one quadrant of a midplane) and 32 compute nodes (a single node card). The allocation strategy used by the Allocator API in BG/L is a first fit approach. This approach allows for the possibility of fragmenting the compute nodes to a point where a request for 128 compute nodes might not be satisfied even though there may be up to 384 available compute nodes.

For example, given a midplane that is completely available, consider the following requests to allocate resources:

- Allocate 128 compute node partition A
- Allocate 128 compute node partition B
- Allocate 128 compute node partition C
- Allocate 32 compute node partition D

At this point, all but the last three node cards have been allocated as shown in Figure 5.

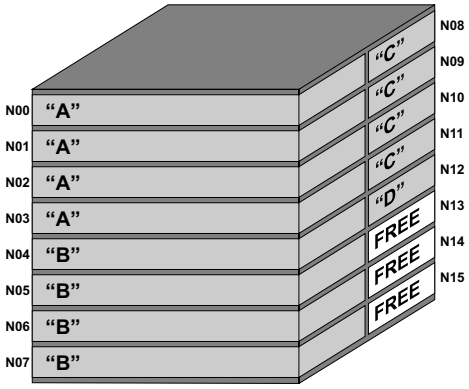

Figure 5. Node card state after allocating partitions A, B, C, and D.

Now consider the next set of requests to free and allocate resources.

- Free partition C
- Allocate 32 compute node partition E
- Free partition B
- Allocate 32 compute node partition F
- Free partition A
- Allocate 32 compute node partition G

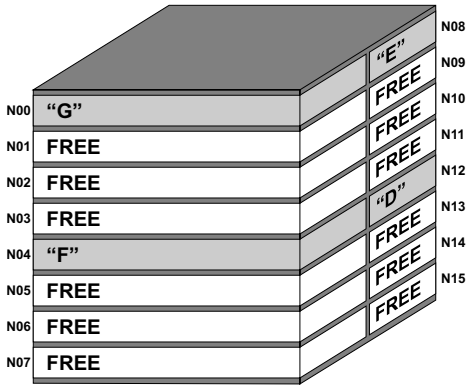

Figure 6. Node card state after deallocating partitions C, B, and A, and allocating partitions E, F, and G using "first fit" allocation.

At this point, only four node cards (a total of 128 compute nodes) are allocated. However, all four quadrants of the midplane are now partially busy.

A subsequent request for a partition of 128 compute nodes cannot be satisfied. Even though there are 384 compute nodes (12 node cards) that are free, the midplane has been fragmented to the point where it cannot satisfy a request for 128 compute nodes.

3.3.2 Optimal Allocation of Small Partitions

The BG/P Allocator API is enhanced to analyze the current state of a midplane and find the most optimal set of compute nodes to allocate. The goal of this analysis is to minimize the fragmentation of the available compute nodes.

Consider again a midplane in the state shown in Figure 5. Using the optimal allocation strategy employed by BG/P, the subsequent requests to free C, B, and A and allocate E, F, and G will result in a state shown in Figure 7. In this case the midplane is less fragmented than the result from the "first fit" strategy shown in Figure 6. The midplane is still able to satisfy allocation requests for any of the supported small partition sizes.

The Allocator API calculates the "optimality" for all sets of available compute nodes of a given size. The optimality is a numeric ranking representing how large a set of free compute nodes must be subdivided in order to allocate a set

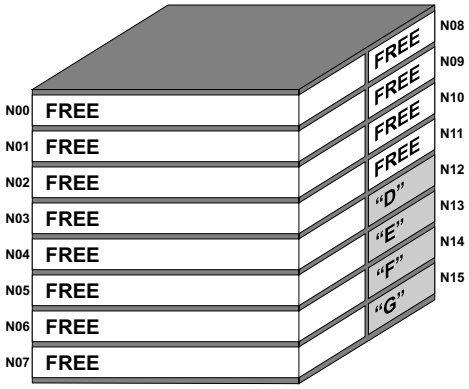

Figure 7. Node card state after deallocating partitions C, B, and A, and allocating partitions E, F, and G using "optimal" allocation.

of compute nodes. An optimality of one (1), indicating the most optimal allocation, is assigned when an allocation request does not require subdividing a larger set of compute nodes. An optimality of two (2) is assigned when an allocation request requires subdividing a set of compute nodes that is twice the size of the allocation request, and so forth.

Table 2 shows the calculated optimality for the possible small partition allocation scenarios.

Table 2. Optimality of allocation request based on the number of free nodes that must be subdivided to satisfy the request

Nodes to Allocate	Free Nodes Subdivided by Allocation					
	512	256	128	64	32	None
16	6	5	4	3	2	1
32	5	4	3	2	-	1
64	4	3	2	-	-	1
128	3	2	-	-	-	1
256	2	-	-	-	-	1

The BG/P system allows five different small partition sizes instead of the two sizes supported by BG/L. In system environments with a high usage of dynamically allocated small partitions, the "first fit" allocation strategy would more quickly fragment the compute node resources. The optimal allocation strategy will reduce such fragmentation on the BG/P system.

3.4 Block Builder

The Blue Gene Navigator is a very powerful graphical interface that allows BG/P system administrators to manage the machine using a web browser based console. One of the many features of the Blue Gene Navigator is a "Block Builder" function for creating static partitions. The administrator can see a picture of their hardware (racks and midplanes) and click on the hardware that will be contained in the partition. They can provide the partition name, the ratio of compute nodes to I/O nodes, and the configuration of either torus or mesh. The block builder then defines the partition in the database, after going through an extensive validation procedure. It validates that cables exist to form the selected hardware into a valid rectangle. For a torus, it validates that there is a complete wrapped torus in all three dimensions. It even allows the notion of a "passthrough" midplane whereby the partition will pass through the switch of a midplane but not use the node on that midplane. Passthrough is a useful concept that allows many more partitioning options and can mitigate the effect of fragmentation. If a midplane along a dimension is in use, a partition can use the other midplanes along that same line, passing through the occupied midplane, and still achieve a torus.

The block builder feature for BG/P also supports the creation of sub-midplane partitions. The administrator can select the midplane, then choose the number of nodes within that midplane from among the supported sizes of 16, 32, 64, 128, and 256. They indicate the partition name, the ratio of compute nodes to I/O nodes, and the node cards that should be used.

In both cases of sub-midplane partitions, and partitions constructed of one or more complete midplanes, the block builder ensures that there are adequate available I/O nodes to fulfill the request for I/O node ratio. If, for instance, the request is for a 1:32 ratio of I/O nodes to compute nodes, but the machine is populated with an I/O node on every other node card (1:64 ratio) then the creation of the partition will fail.

3.5 MMCS

MMCS also makes available several console commands that can be used to define partitions. One advantage of using the console commands over the Block Builder GUI is that they can easily be scripted. This way, an administrator can build up a set of standard partitions, and at any time they could clean out any existing partitions, rerun the script, and they will back to a known state, in terms of defined partitions.

The following five commands are used to create various partition sizes:

genblock generates a single partition over a single midplane

genblocks generates a partition for each midplane on the machine

genBPblock generates a partition over a set of midplanes

genfullblock generates a single partition that encompasses the entire machine

gensmallblock generates a sub-midplane partition

There is no MMCS command that handles the unique passthrough case. Block Builder, the Bridge APIs, or the Allocator API must be used to create a partition that handles passthrough.

4 Job Management

For the scope of this paper, a BG/P *job* consists of a single executable running on each compute node in a partition. The act of managing jobs is similar to the Linux process model where each job has a unique identifier used to send signals. Each job also has standard input, output, and error associated with it, and returns an exit status upon its completion. Managing jobs consists of starting, signaling, and stopping them. Typically these steps take place between booting and destroying a partition, though it does not have to since a job will not start until its partition has booted. Thus a job can be started immediately after its partition is booted, even though booting of a partition is not an instantaneous process. Since each job requires a partition, managing jobs is much easier than managing partitions. Once a job is started using one of the interfaces described below, it will run until completion, or until signaled or cancelled.

4.1 mpirun

Mpirun is a utility designed to launch jobs on BG/P. It can be used in a stand alone fashion by users to launch jobs, or in a batch mode to launch jobs from a scheduler. Little has changed in the external appearance of mpirun from BG/L. As described in [1], it acts as a proxy, or shadow of the job executing on the BG/P core. This means when input is sent to mpirun by the user, it is routed to the job executing on the BG/P core appropriately. Similarly, standard output and error generated from the job is routed back to mpirun. Mpriun has the capability to allocate and boot partitions when supplied with the appropriate size or shape arguments, as well as to use existing partitions with or without booting or freeing them. These features allow it to be used in a variety of scheduling strategies and environments, which are described in detail in Section 5.

4.2 MMCS

MMCS has the ability to run jobs directly against booted partitions. Generally, users will go through a scheduler or some coordinated job submission, and use mpirun within that framework, but for an ad hoc jobs, or for doing testing, the MMCS console can be used for job submission. Once a partition is in the Initialized state, the following commands can be used from an MMCS console:

associatejob associate a job with a partition, when createjob was used to create the job entry, but the partition id was omitted

createjob create a job, providing an executable, a working directory, and optionally, a partition id, but don't start the job

getjobinfo get information about a job that is either running or queued

killjob kill a running job

setjobargs set arguments and environment variables for a job that has been created but not yet started

startjob start a job that was created

submitjob create and start a job in one step

5 Scheduling Strategies

Using combinations of the resource management interfaces presented in Section 2, 3, and 4, several strategies exist to effectively schedule jobs on BG/P. The resource management architecture described previously does not execute inside the BG/P core, instead it executes solely on the front-end and service nodes. Due to this feature, the central scheduling logic of many resource management systems can be retained without modifications. From a high level view, the logic used by these systems can be placed into two distinctly separate categories: static partitioning and dynamic partitioning. Static partitioning schemes are easier to implement since they consist of a system administrator setting up predefined partition sizes, and the scheduler matching incoming job requirements to the partition sizes as they are available. Dynamic partition schemes start with a BG/P system that is completely available. Partitions are created on demand as incoming jobs arrive, and destroyed as the jobs end. Such a scheme requires more work since the scheduling system will need to use either the Bridge API or the Allocator API to create partitions.

In either partitioning scheme, mpirun or the Bridge API can be used to start and stop jobs, thereby forming a scheduling strategy. Each strategy has benefits such as ease of implementation, and drawbacks such as fragmentation. Each strategy is described in detail below.

5.1 Dynamic partitioning using Allocator API and mpirun

The Allocator API, described in Section 3.3, offers an easy to implement system for allocating various sizes and shapes of partitions without the extensive development effort required to implement and utilize all the features provided by the Bridge API. In this scheme, a resource manager would use the Allocator API to first create and boot the partition, then call mpirun to run the job, and lastly free and remove the partition once the job completes.

5.2 Static partitioning using Bridge API and mpirun

In a static partitioned environment, the customer will predefine partitions to be used by the central resource manager. Typically these are overlapping partitions. For example, a two rack BG/P system could have static partitions defined for the entire system (4 midplanes), half the system (2 midplanes), each midplane, and sub-midplanes. In this scheme, a resource manager would first match job requirements to an available predefined partition size. Next it boots the partition, then calls mpirun to run the job. Once the job has completed, the resource manager could deallocate the partition depending on the scheduling algorithm. In some cases, a customer may prefer to leave partitions in a constantly booted state, and simply match incoming jobs to the available partition sizes.

5.3 Dynamic partitioning using Bridge API and mpirun

This scheme provides the most efficient and flexible environment as a scheduling strategy for optimal system utilization. However, it requires a great deal of effort to fully implement the features provided by the Bridge API to properly allocate multi-midplane partitions. In such a scheme, the resource manager uses the Bridge APIs to determine available resources based on job requirements using its own specialized allocation algorithm (best fit, first fit, backfill, migration, etc). Next, the resource manager creates and allocates a partition, then uses mpirun to run the job. Once the job completes, the partition is deallocated. With this scheme, switches can be utilized in a passthrough environment to mitigate fragmentation.

5.4 Static partitioning and mpirun

Allocating partitions directly using MMCS or block builder is another scheduling strategy that can be utilized in certain specialized environments. In this situation, a centralized resource manager is not utilized. Such a system is known as a first come first serve, or honor system, since nothing prevents users from attempting to access the same resources concurrently. While this scheme is not strictly a scheduling policy, but rather a *lack of scheduling* policy, it exemplifies how the BG/P resource management architecture is flexible enough to allow a scheme with almost no restrictions present.

6 Related Works

The authors of [1] describe the job management architecture of BG/L in detail, concluding that it is capable of supporting job management systems developed internally at IBM such as LoadLeveler, as well as third party systems such as SLURM [4]. They define three *openness* characteristics. First, the job management system runs outside the BG/L core. Secondly, the job management logic can be retained without modification. Lastly, multiple scheduling models are supported by presenting raw resources using the Bridge API. These three characteristics allow resource management systems to be easily ported to the BG/L environment.

In [5] the authors describe various job scheduling algorithms for BG/L. Including a first come first serve (FCFS), a FCFS with backfill, a FCFS with migration, and lastly a FCFS with backfill and migration. They concluded that while scheduling algorithms with migration strategies resulted in lower system fragmentation under a workload of small jobs, they require more overhead than backfilling strategies.

7 Conclusion

This paper presented the resource management architecture for the BG/P supercomputer and showed how it is flexible enough to support a wide variety of resource management systems and their scheduling strategies. Both static and dynamic partition allocation strategies can be combined with job management strategies using the Bridge API or mpirun to form a unique scheduling strategy suitable for almost any environment. The BG/P resource management architecture features a number of improvements over its BG/L counterpart. Most notably, the addition of a Real-Time API, as well as an improved Allocator API to reduce system fragmentation during small partition allocation.

Blue Gene/P Glossary

The BG/P machine is composed of the following entities: compute nodes, I/O nodes, node cards, service node, front end nodes, base partitions, switches, and MMCS. Each entity is briefly summarized below.

Compute Node A compute node is also referred to as a c-node. This is a system-on-chip node that supports two execution modes, virtual node mode and SMP mode. All the c-nodes are interconnected in a three-dimensional toroidal pattern. Each c-node has a unique address and location in the three-dimensional toroidal space. Compute nodes execute the jobs' tasks while running a minimal custom operating system called a compute node kernel (CNK) to provide very small overhead.

I/O Node An I/O node is a special node that connects the compute nodes to the outside world. I/O nodes allow processes that are executing in the compute nodes to perform I/O operations (e.g., accessing files) and to communicate with the job management system. Each I/O node serves anywhere from 16 to 128 c-nodes. An I/O node and its associated c-nodes are called a p-set. I/O nodes run Linux.

Node Card A node card contains 32 compute nodes and up to 2 I/O nodes, for a 1:16 ratio of I/O nodes to compute nodes. Sixteen node cards form a midplane, and 2 midplanes form a BG/P rack.

Service Node (SN) A service node is dedicated hardware that runs software to control and manage the system. The Service node runs management software components including the job management system. Persistent storage of hardware information and state is maintained on the service node in a DB2 database.

Front End Node (FEN) Front end nodes are machines from which users and administrators interact with BG/P. Applications are compiled on and submitted for execution in the BG/P core from FEN. User interactions with applications, including debugging, are also performed from the FEN.

Base Partition (BP) A base partition (midplane) is a group of c-nodes connected in a 3D mesh or torus pattern and their controlling I/O nodes. A base partition is the minimal allocation unit for which a torus can be achieved. For BG/P, the base partition is a midplane (512 nodes). BPs can be combined together to form larger partitions, and in most cases still achieve a torus. Sub-midplane partitions are supported, but these will always be a mesh, meaning that not all nodes have 6 neighbors.

Switches A base partition (midplane) has switches that complete the connections between the nodes and the data cables. This is contained within the link chip hardware on each midplane. To some extent, the control system software has abstracted the link chips on a midplane to single X, Y, and Z switches for each midplane. A switch can wrap back to the midplane when the dimension is one, or the switch can connect to data cables (wires) when the dimension contains more than one midplane.

Partition A partition is a group of nodes allocated to a job. A partition is created on demand to execute a job and is normally destroyed when the job is terminated. Partitions are physically (electronically) isolated from each other (i.e., messages cannot flow outside an allocated partition). A partition can have the topology of a mesh or a torus. Partitions that are a torus will necessarily be a combination of one or more base partitions. Partitions that are a mesh can be groups of base partitions, but can also be smaller than a base partition (sub-midplane). The term "block" is sometimes used for partition, and the two terms are synonyms.

MMCS The Midplane Monitoring and Control System (MMCS), is the main control system component that serves as the interface to the BG/P machine. It contains persistent storage with information (configuration and status) on the entire machine. It also provides various services to perform actions on the BG/P (e.g., to launch a job)

References

[1] Y. Aridor, T. Domany, O. Goldshmidt, Y. Kliteynik, E. Shmueli, and J. E. Moreira. Open job management architecture for the BlueGene/L supercomputer. In *Job Scheduling Strategies for Parallel Processing, 11th International Workshop*, pages 91–107, Cambridge, Massachusetts, June 2005.

[2] Y. Aridor, T. Domany, O. Goldshmidt, J. E. Moreira, and E. Shmueli. Resource allocation and utilization in the BlueGene/L supercomputer. *IBM Journal of Research and Development*, 49(2/3):425–436, 2005.

[3] D. G. Feitelson and M. A. Jette. Improved utilization and responsiveness with gang scheduling. In *Job Scheduling Strategies for Parallel Processing, 3rd International Workshop*, pages 238–261, Geneva, Switzerland, Apr. 1997.

[4] M. A. Jette, A. B. Yoo, and M. Grondona. SLURM: Simple linux utility for resource management. *Job Scheduling Strategies for Parallel Processing*, 9:37–51, 2003.

[5] E. Krevat, J. Castaños, and J. E. Moreira. Job scheduling for the BlueGene/L system. In *Job Scheduling Strategies for Parallel Processing, 8th International Workshop*, pages 38–54, Edinburgh, Scotland, UK, July 2002.

[6] N.R. Adiga et al. An overview of the BlueGene/L supercomputer. In *Supercomputing*, pages 1–22, Baltimore, Maryland, Nov. 2002.