Designing a Highly-Scalable Operating System: The Blue Gene/L Story

José Moreira*, Michael Brutman**, José Castaños*, Thomas Engelsiepen***, Mark Giampapa*, Tom Gooding**, Roger Haskin***, Todd Inglett**, Derek Lieber*, Pat McCarthy**, Mike Mundy**, Jeff Parker**, Brian Wallenfelt****

*{*jmoreira,castanos,giampapa,lieber*}@us.ibm.com
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598

***{*engelspn,roger*}@almaden.ibm.com
IBM Almaden Research Center
San Jose, CA 95120

**{*brutman,tgooding,tinglett,pjmccart,mmundy,jjparker*}@us.ibm.com
IBM Systems and Technology Group
Rochester, MN 55901

*****wallenfe*@gmail.com>
GeoDigm Corporation
Chanhassen, MN 55317

Abstract

Blue Gene/L is currently the world's fastest and most scalable supercomputer. It has demonstrated essentially linear scaling all the way to 131,072 processors in several benchmarks and real applications. The operating systems for the compute and I/O nodes of Blue Gene/L are among the components responsible for that scalability. Compute nodes are dedicated to running application processes, whereas I/O nodes are dedicated to performing system functions. The operating systems adopted for each of these nodes reflect this separation of function. Compute nodes run a lightweight operating system called the compute node kernel. I/O nodes run a port of the Linux operating system. This paper discusses the architecture and design of this solution for Blue Gene/L in the context of the hardware characteristics that led to the design decisions. It also explains and demonstrates how those decisions are instrumental in achieving the performance and scalability for which Blue Gene/L is famous.

1 Introduction

The Blue Gene/L supercomputer has a proven scalability record up to 65,536 dual-processor compute nodes, as reported in [4],[9],[21],[22],[23] and [26]. Several system components contribute to that scalability. In this paper we discuss one of them: the operating system solution for its nodes. Although the system software, including the operating system, for Blue Gene/L has been described before in [15], this paper presents it from a different perspective, with additional information and experimental results.

Each Blue Gene/L compute node is a small computer with two processors and its own private memory that is not visible to other nodes. The standard approach for this kind of parallel machine is to run one instance of a node operating system on each compute node. In principle, this would indicate that scalability of the node operating system is not a concern by itself. Nevertheless, there are several constraints that complicate matters with respect to designing the operating system solution for Blue Gene/L.

First and foremost, we had to deliver and support several Blue Gene/L systems in a tight budget and schedule. The development team was approximately 10-20% of the size of a normal IBM server development team. This constraint actually guided many decisions, not only for software but also for Blue Gene/L hardware. Second, we were aware of problems that other teams had encountered and studied regarding the interference of system functions on large parallel machines. We needed an operating system solution for Blue Gene/L that would support efficient execution of communicating parallel jobs with up to 131,072 tasks. Third, we had to have a flexible solution that could (1) be extended as new customer requirements arose and (2) be modified if we observed scalability problems.

An extensible approach was crucial to the entire design. Even though Lawrence Livermore National Laboratory (LLNL) was the major sponsoring customer, we knew early on that the business model for Blue Gene required additional customers. Very likely (and indeed this is what happened) those additional customers would have requirements beyond LLNL's interests. Examples of such additional requirements include sockets communication, new file systems, and new job schedulers.

In designing the operating system solution for Blue Gene/L, we followed a simple principle: The structure of the software should reflect the structure of the hardware. The Blue Gene/L hardware architecture includes a variety of nodes with dedicated roles: compute nodes, I/O nodes, service nodes, front-end nodes and file server nodes. Compute and I/O nodes constitute what is typically called a Blue Gene/L machine. Service, front-end and file server nodes are standard, commercially available computers that play a supporting role in implementing a complete Blue Gene/L system.

The operating system solution for Blue Gene/L reflects this heterogeneity and specialization of the hardware. Compute nodes run an operating system that is dedicated to supporting application processes performing computations. I/O nodes run an operating system that is more flexible and can support various forms of I/O. The service node (which runs a conventional off-the-shelf operating system) provides support for the execution of the compute and I/O node operating systems. The file servers store data that the I/O nodes read and write. The front-end nodes support program compilation, submission, and debugging. This approach of specialized hardware and software functions is similar to that adopted in ASCI Red [8], which had both compute and system nodes, each performing different functions.

This paper describes the operating system solution for the Blue Gene/L compute and I/O nodes. We also discuss the role of the service node, since it impacts the compute and I/O nodes. Further discussion of the front-end and file server nodes is beyond the scope of this paper, since software on the front-end nodes does not really interact with the compute and I/O nodes operating systems, and file server nodes are in no way specific to Blue Gene/L.

The rest of this paper is organized as follows. Section 2 describes the general architecture of the Blue Gene/L machine, providing the necessary context to understand the solution adopted for the operating system. Section 3 describes the solution itself, including those components running on the compute nodes, I/O nodes, and service node. Section 4 describes our experience in developing and deploying the solution, including additional features that were added per customer requests. That section also reports on experimental results that help us understand the performance characteristics of our solution. And finally, Section 5 presents our conclusions.

2 General architecture of Blue Gene/L

We briefly discuss the overall system architecture of Blue Gene/L in this section. For a more thorough description, the reader is referred to [7]. A Blue Gene/L machine consists of a number of compute and I/O nodes interconnected in a regular topology. See Figure 1 for a high-level view of a Blue Gene/L system. The compute and I/O nodes form the computational core of Blue Gene/L. They are controlled from the service node through an Ethernet *control network*. The control traffic is converted to lower level protocols (e.g., JTAG) before actually making to the compute and I/O nodes. The I/O nodes connect the core to file servers and front-end nodes through a separate Ethernet *functional network*.

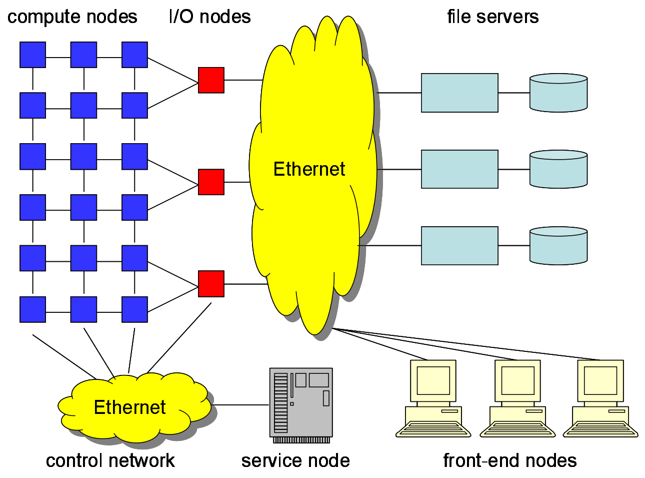

Figure 1: High-Level View of a Blue Gene/L System.

The basic building block of Blue Gene/L is the Blue Gene/L Compute ASIC (BLC). The internals of that ASIC are shown in Figure 2. The BLC contains two non-coherent PowerPC 440 cores, each with its own private L1 cache (split for instructions and data). Associated with each core is a small (2 KiB[1]) L2 cache that acts as a prefetch buffer and translates between the 32-byte cache line size of the L1 to the 128-byte cache line size of the L3. Completing the on-chip memory hierarchy is 4 MiB[2] of embedded DRAM (eDRAM) that is configured to operate as a shared L3 cache. Also on the BLC is a memory controller (for external DRAM) and interfaces to the five networks used to interconnect Blue Gene/L compute and I/O nodes: torus, collective, global barrier, Ethernet, and control network.

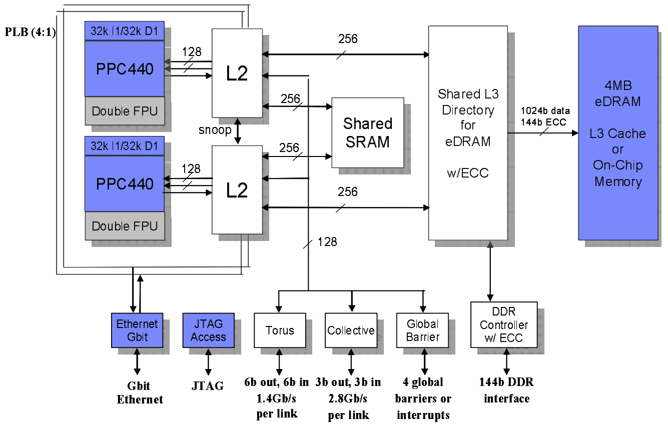

Figure 2: Blue Gene/L Compute ASIC (BLC).

[1] KiB = 1,024 bytes (kibibyte - http://en.wikipedia.org/wiki/Binary_prefixes)
[2] MiB = 1,048,576 bytes (mebibyte - http://en.wikipedia.org/wiki/Binary_prefixes)

Blue Gene/L compute and I/O nodes both use this ASIC with 512 MiB of external memory (a 1 GiB[3] option is also available for the compute nodes), but their roles are quite different, as described in Section 3. Only compute nodes are interconnected through the torus network. Conversely, only I/O nodes have their Ethernet port connected to the functional network. The I/O nodes plug into an Ethernet fabric together with the file servers and front-end nodes. The collective and global barrier networks interconnect all (compute and I/O) nodes. The control network is used to control the hardware from the service node.

Blue Gene/L is a partitionable machine, and can be divided along natural boundaries into electrically isolated partitions. These boundaries are 8x8x8 configurations of compute nodes called midplanes. A partition is formed by a rectangular arrangement of midplanes. Each partition can run one and only one job at any given time. During each job, the compute nodes of a partition are in one of two modes of execution: coprocessor mode or virtual node mode. All compute nodes stay in the same mode for the duration of the job. These modes of execution are described in more detail below.

3 The Blue Gene/L operating system solution

The operating system solution for Blue Gene/L has components running on the I/O and service nodes, in addition to the actual compute node kernel. In this section, we describe that solution. We start with the overall architecture and proceed to describe the role of the separate components. As previously mentioned, the software architecture reflects to a great degree the hardware architecture of Blue Gene/L.

3.1 Overall operating system architecture

A key concept in the Blue Gene/L operating system solution is the organization of compute and I/O nodes into logical entities called *processing sets* or *psets*. A pset consists of one I/O node and a collection of compute nodes. Every system partition, in turn, is organized as a collection of psets. All psets in a partition must have the same number of compute nodes, and the psets of a partition must cover all the I/O and compute nodes of the partition. The psets of a partition never overlap. The supported pset sizes are 8, 16, 32, 64 and 128 compute nodes, plus the I/O node.

The psets are a purely logical concept implemented by the Blue Gene/L system software stack. They are built to reflect the topological proximity between I/O and compute nodes, thus improving communication performance within a pset. The regular assignment of compute to I/O nodes enforced by the pset concept allows us to simplify the system software stack while delivering good performance and scalability. With a static assignment of I/O to compute nodes, it becomes easier to separate operating system responsibilities. To understand those responsibilities, it is useful to have a picture of the job model for Blue Gene/L.

A Blue Gene/L job consists of a collection of *N* compute processes. Each process has its own private address space and two processes of the same job communicate only through message passing. The primary communication model for Blue Gene/L is MPI. The *N* compute processes of a Blue Gene/L job correspond to tasks with ranks 0 to *N*-1 in the MPI_COMM_WORLD communicator.

Compute processes run only on compute nodes; conversely, compute nodes run only compute processes. The compute nodes of a partition can all execute either one process (in coprocessor mode) or two processes (in virtual node mode) each. In coprocessor mode, the single process in the node has access to the entire node memory. One processor executes user code while the other performs communication functions. In virtual node mode, the node memory is split in half between the two processes running on the two processors. Each process performs both computation and communication functions. The compute node kernel implements these models in the compute nodes.

I/O nodes behave more like conventional computers. In fact, each I/O node runs one image of the Linux operating system. It can offer the entire spectrum of services expected in a Linux box, such as multiprocessing, file systems, and a TCP/IP communication stack. These services are used to extend the capabilities of the compute node kernel, providing a richer functionality to the compute processes. Due to the lack of cache coherency between the processors of a Blue Gene/L node, we only use one of the processors of each I/O node. The other processor remains idle.

3.2 The compute node kernel

The compute node kernel (CNK) accomplishes a role similar to that of PUMA [24],[25] in ASCI Red by controlling the Blue Gene/L compute nodes. It is a lean operating system that performs a simple sequence of operations at job start time. This sequence of operations happens in every compute node of a partition, at the start of each job:

1. It creates the address space(s) for execution of compute process(es) in a compute node.
2. It loads code and initialized data for the executable of that (those) process(es).
3. It transfers processor control to the loaded executable, changing from supervisor to user mode.

The CNK consumes only 1 MiB of memory. It can create either one address space of 511 MiB for one process (in coprocessor mode) or two address spaces of 255 MiB each for two processes (in virtual node mode). (1023 and 511 MiB respectively with the 1 GiB memory option.) The address spaces are flat and fixed, with no paging. The entire

[3] GiB = 1,073,741,824 bytes (gibibyte - http://en.wikipedia.org/wiki/Binary_prefixes)

mapping is designed to fit statically in the TLBs of the PowerPC 440 processors.

The loading of code and data occurs in push mode. The I/O node of a pset reads the executable from the file system and forwards it to all compute nodes in the pset. The CNK in a compute node receives that executable and stores the appropriate memory values in the address space(s) of the compute process(es).

Once the CNK transfers control to the user application, its primary mission is to "stay out of the way". Since there is only one thread of execution per processor, there is no scheduling for the kernel to perform. Also, the memory space of a process is completely covered by the TLB in the processor running that process, so there is no memory management to perform. In normal execution, processor control stays with the compute process until it requests an operating system service through a system call. Exceptions to this normal execution are caused by hardware interrupts: either timer alarms requested by the user code or an abnormal hardware event that requires attention by the compute node kernel.

When a compute process makes a system call, three things may happen:

1. "Simple" system calls that require little operating system functionality, such as getting the time or setting an alarm, are handled locally by the compute node kernel. Control is transferred back to the compute process at completion of the call.
2. "I/O" system calls that require infrastructure for file systems and IP stack are shipped for execution in the I/O node associated with that compute node. (That is, the I/O node in the pset of the compute node.) The compute node kernel waits for a reply from the I/O node, and then returns control back to the compute process.
3. "Unsupported" system calls that require infrastructure not present in Blue Gene/L, such as *fork* and *mmap*, are returned right away with an error condition.

In Blue Gene/L we have implemented 68 system calls from Linux and an additional 18 CNK-specific calls [11]. The other Linux system calls are unsupported. In our experience, very seldom does a scientific application need one of the unsupported system calls. When it does, we adopt a combination of two solutions: either (1) change the application or (2) add the required system call.

There are two main benefits from the simple approach for a compute node operating system: robustness and scalability. Robustness comes from the fact that the compute node kernel performs few services, which greatly simplifies its design, implementation, and test. Scalability comes from lack of interference with running compute processes. Previous work by other teams [16] has identified system interference as a major source of performance degradation in large parallel systems. The effectiveness of our approach in delivering a system essentially free of interference has been verified directly through measurements of system noise [5],[12],[20] and indirectly through measurements of scalability all the way to 131,072 tasks in real applications [9],[21],[26] .

3.3 The role of the I/O node

The I/O node plays a dual role in Blue Gene/L. On one hand, it acts as an effective master of its corresponding pset. On the other hand, it services requests from compute nodes in that pset. Jobs are launched in a partition by contacting corresponding I/O nodes. Each I/O node is then responsible for loading and starting the execution of the processes in each of the compute nodes of its pset. Once the compute processes start running, the I/O nodes wait for requests from those processes. Those requests are mainly I/O operations to be performed against the file systems mounted in the I/O node.

Blue Gene/L I/O nodes execute an embedded version of the Linux operating system. We call it embedded because it does not use any swap space, it has an in-memory root file system, it uses little memory, and it lacks the majority of daemons and services found in a server-grade configuration of Linux. It is, however, a complete port of the Linux kernel and those services can be, and in various cases have been, turned on for specific purposes. The Linux in Blue Gene/L I/O nodes includes a full TCP/IP stack, supporting communications to the outside world through Ethernet. It also includes file system support. Various network file systems have been ported to the Blue Gene/L I/O node, including GPFS, Lustre, NFS, and PVFS2 [17].

Blue Gene/L I/O nodes never run application processes. That duty is reserved to the compute nodes. The main user-level process running on the Blue Gene/L I/O node is the control and I/O daemon (CIOD). CIOD is the process that links the compute processes of an application running on compute nodes to the outside world. To launch a user job in a partition, the service node contacts the CIOD of each I/O node of the partition and passes the parameters of the job (user ID, group ID, supplementary groups, executable name, starting working directory, command line arguments, and environment variables). CIOD swaps itself to the user's identity, which includes the user ID, group ID, and supplementary groups. It then retrieves the executable from the file system and sends the code and initialized data through the collective network to each of the compute nodes in the pset. It also sends the command-line arguments and environment variables, together with a start signal.

Figure 3 illustrates how I/O system calls are handled in Blue Gene/L. When a compute process performs a system call requiring I/O (e.g., *open*, *close*, *read*, *write*), that call is trapped by the compute node kernel, which packages the parameters of the system call and sends that message to the CIOD in its corresponding I/O node. CIOD unpacks the message and then reissues the system call, this time under the Linux operating system of the I/O node. Once the system call completes, CIOD packages the result and sends it back to the originating compute node kernel, which, in turn, returns the result to the compute process. This simple model works well for transactional operations, such as read and write, which have a clear scope of operation and represent the bulk of I/O in scientific computing. It does not support operations such as memory mapped files, but those are uncommon in scientific computing.

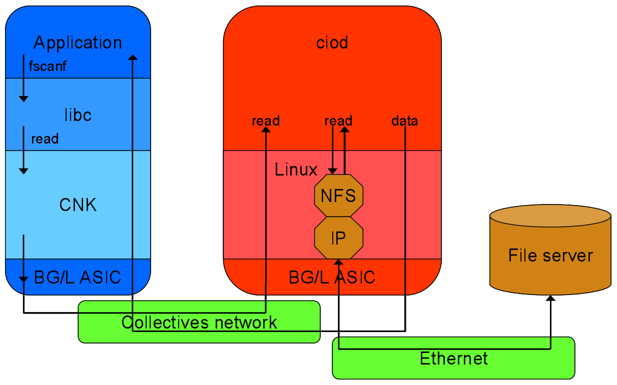

Figure 3: Function Shipping from CNK to CIOD.

There is a synergistic effect between simplification and separation of responsibilities. By offloading complex system operations to the I/O node we keep the compute node operating system simple. Correspondingly, by keeping application processes separate from I/O node activity we avoid many security and safety issues regarding execution in the I/O nodes. In particular, there is never a need for the common scrubbing daemons typically used in Linux clusters to clean up after misbehaving jobs. Just as keeping system services in the I/O nodes prevents interference with compute processes, keeping those processes in compute nodes prevents interference with system services in the I/O node. This isolation is particularly helpful during performance debugging work. The overall simplification of the operating system has enabled the scalability, reproducibility (performance results for Blue Gene/L applications are very close across runs [12]), and high-performance of important Blue Gene/L applications.

3.4 The role of the service node

The Blue Gene/L service node runs its control software, typically referred to as the Blue Gene/L control system. The control system is responsible for operation and monitoring of all compute and I/O nodes. It is also responsible for other hardware components such as link chips, power supplies, and fans. But this functionality is outside the scope of this paper. Tight integration between the Blue Gene/L control system and the I/O and compute nodes operating systems is central to the Blue Gene/L software stack. It represents one more step in the specialization of services that characterize that stack.

In Blue Gene/L, the control system is responsible for setting up system partitions and loading initial code and state in the nodes of a partition. The Blue Gene/L compute and I/O nodes are completely stateless: no hard drives and no persistent memory. When a partition is created, the control system programs the hardware to isolate that partition from others in the system. It computes the network routing for the torus, collective and global interrupt networks, thus simplifying the compute node kernel. It loads operating system code for all compute and I/O nodes of a partition through the dedicated control network. It also loads an initial state in each node (called the personality of the node). The personality of a node contains information specific to the node. Key components of the personality of I/O and compute nodes are shown in Table 1.

Table 1: Personalities of Compute and I/O Nodes.

Compute node personality	I/O node personality
• Memory size • Bit-steering for memory configuration • Physical location • *x*, *y*, *z* torus coordinates • Pset number and size • Routes for the collectives network	• Memory size • Bit-steering for memory configuration • Physical location • Pset number and size • Routes for the collectives network • MAC and IP address • Broadcast and gateway IP addresses • Service node IP address • NFS server address and export directory • Security key

4 Experience and experiments

In this section, we discuss the experience of working with the Blue Gene/L operating system solution. We first discuss the effort to add socket communication support. We then discuss the work on different file systems ported to Blue Gene/L. We continue with a discussion of a recent enhancement to the programming environment for Blue Gene/L, the support for MPMD applications. Finally, we report performance results from benchmarks and applications that demonstrate the scalability of Blue Gene/L.

4.1 Socket communications in Blue Gene/L

MPI has always been the primary model for inter-task communication in Blue Gene/L. MPI is implemented entirely in user mode, and it works only between the tasks of a job. This is highly satisfactory for LLNL. However, our second customer, The Netherlands Foundation for Research in Astronomy (ASTRON, http://www.astron.nl/), wanted to use Blue Gene/L to process streams of signals from the new LOFAR radio telescope (http://www.lofar.nl/) [18]. Our approach was to stick to standards and implement sockets in Blue Gene/L. We accomplished that through changes in both the compute node kernel and in the CIOD component of the I/O node software stack. For simplicity, we chose to implement only client side sockets. That is, Blue Gene/L compute processes can initiate socket connections but they cannot accept incoming socket connections.

The changes to the compute node kernel were relatively simple. We just had to add the system calls used for sockets in Linux. These system calls do not really cause any action on the compute node, except for shipping them directly to the corresponding CIOD for execution. Changes to CIOD were a bit more elaborate. Different from file system operations, socket reads and writes can block for indefinite periods of time. Since a single CIOD serves all compute processes of a pset, it is a bad idea for it to block indefinitely! We addressed this problem by replacing blocking calls at the compute process level with non-blocking calls at the time of reissue by CIOD. CIOD places the blocking operation in a wait queue and polls the socket until the operation is complete. This allows CIOD to move on to other things if a particular socket operation is not ready to be performed.

Performance optimization of socket read/write operations required additional work during the development of those features. CIOD has to multiplex between polling the collectives network (that interconnects I/O and compute nodes) for requests from several compute processes and actually performing (or initiating) those requests. There was quite a bit of tuning necessary to find the right balance between these activities. If CIOD does not look for work from the compute processes often enough, latency will suffer. On the other hand, if it burns too many cycles looking for new work, delivered bandwidth of work already in progress may suffer. There is also the issue of how often CIOD needs to check for the outstanding operations in the wait queue.

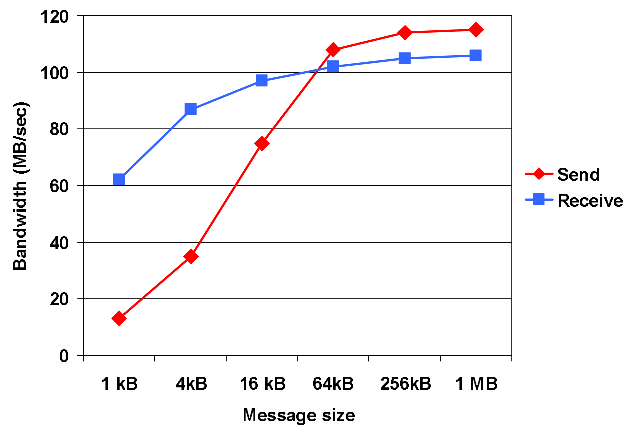

Figure 4: Performance of Socket I/O in Blue Gene/L.

In the end, sustained performance for socket I/O is quite satisfactory, as can be seen in Figure 4. Out of a peak Gigabit Ethernet bandwidth of 125 MB/s per I/O node, we see that socket I/O in Blue Gene/L achieves over 110 MB/s per I/O node.

4.2 File systems in Blue Gene/L

The Blue Gene/L compute nodes are completely oblivious to which file systems are in use. Since file operations are shipped to CIOD to be reissued within the I/O node Linux, applications can use any file system that has been ported to the I/O node Linux. The main file system in use at LLNL is Lustre, ported by a local team [19]. GPFS and PVFS2 also work with Blue Gene/L [17]. In some cases the port of a new file system can be quite easy (Argonne did their PVFS2 port in a couple of days).

For GPFS in particular, it was easy to prototype additional packages required by the file system, but practical considerations made it difficult to include the packages in the product release. More significantly, the absence of local disk in the I/O node required inventing new techniques for storing per-node information. These include configuration files, trace output, and stuff that goes in `/var` (for example) on standard Linux systems. We considered a variety of alternatives to provision this information, finally deciding on a single NFS shared file system for storing the per-node information.

For file systems, the effort in tuning for performance is much larger than just getting the system to work. In another demonstration of extensibility, LLNL modified CIOD to perform different file system setup, depending if the job about to start was in coprocessor or virtual node mode [13]. This proves to be very important for Lustre performance. Based on this experience, we have extended CIOD to allow selecting file systems on a per job basis. This is a good example of how ideas from several sources make into Blue Gene/L for benefit of all users.

Results for file system performance with GPFS are shown in Figure 5. This particular experiment is performed on a small Blue Gene/L machine (two racks with a total of 2048 compute nodes and 256 I/O nodes). That Blue Gene/L machine is paired, through a Cisco 6509 Gigabit Ethernet switch, with a GPFS I/O cluster consisting of 32 dual-processor 3.2 GHz Xeon servers with 4 GiB of memory each. The servers are connected by Fibre Channel to eight (8) DS4300 disk controllers. Each controller has three expansion drawers. Since each controller an drawer holds 14 disks, the storage subsystem has a total of 448 disks. These are 72 GB 15k RPM disks.

It is clear from the graph that performance scales well with number of nodes until "something else" saturates. For this particular experiment, that something else is the DS4300 disk controllers. Writes saturate before reads because of the RAID-5 operations in the controllers.

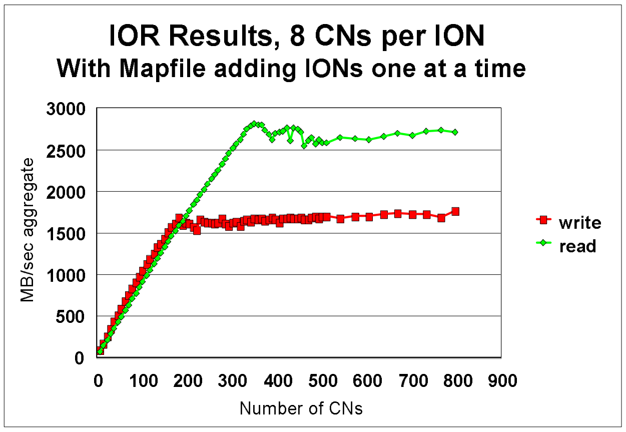

Figure 5: Performance of GPFS in Blue Gene/L.

4.3 MPMD support

The initial program execution environment for Blue Gene/L supported only *Single Program Multiple Data* (SPMD) applications. All compute tasks in a job had to execute exactly the same code. To satisfy a request from the National Center for Atmospheric Research (NCAR), we recently made an enhancement to the system to support *Multiple Program Multiple Data* (MPMD) applications.

There are limitations to the implementation, but it follows the philosophy of the compute node kernel staying out the way and letting the user take full advantage of the system. The user does need to provide a "launcher" program that starts executing on all the nodes. That launcher program then selects the application code to run based on the MPI rank of the compute node. Execution of the application code is initiated by the *execve* system call, which was added for this specific purpose.

4.4 Benchmarks and applications

There are several results about Blue Gene/L performance published in the literature [1],[4],[9],[10],[14],[21],[26]. We here focus on just a few examples to illustrate the impact of the operating system in the performance of benchmarks and applications. First, we compare the performance of a single Blue Gene/L node executing the serial version of the NAS parallel benchmarks (class A) on top of two different operating systems: the CNK normally used on the Blue Gene/L compute nodes, and Linux normally used on the Blue Gene/L I/O nodes. Those results are shown in Table 2. It is clear that the performance benefit of CNK can be substantial, with the execution time increasing up to 300% for the IS benchmark. The reason for performance difference is that Linux has to handle TLB misses during execution, while CNK does not, as it maps the entire node memory in the TLB of the processor. The impact is worse for code that touches memory randomly, like IS.

Table 2: Single-node Performance for NAS Benchmarks on CNK and Linux.

	CNK		Linux			
	Time(s)	Speed (Mops)	Time(s)	Speed (Mops)	% Time	% Speed
is.A	5.7	14.7	23.0	3.6	303.0	-75.5
ep.A	172.6	3.1	177.2	3.0	2.7	-3.2
lu.A	380.3	313.7	504.0	237.0	32.5	-24.5
sp.A	375.0	227.0	517.4	164.3	38.0	-27.6
cg.A	27.1	55.0	32.3	46.3	19.2	-15.8
bt.A	522.6	322.0	613.0	274.4	17.3	-14.8

Illustrating the issue of scalability, Figure 6 shows the performance behavior of Flash (an astrophysics code from the University of Chicago) [6] on various systems. This is a weak scaling experiment, so ideal behavior would be for the execution time to stay constant as the number of processors increase. We observe that that is true only for the Blue Gene/L machines. All other machines eventually reach a point where the execution time grows significantly worse with the system size. Figure 7 is a speedup curve for Miranda (a hydrodynamics code from LLNL) on Blue Gene/L [4]. This is a strong scaling experiment, and we observe an almost linear improvement in performance from 8192 to 65536 processors. Contributing to the scalability of Flash and Miranda on Blue Gene/L are the low-overhead user-mode MPI (enabled by the kernel) and the non-interference of system services on applications.

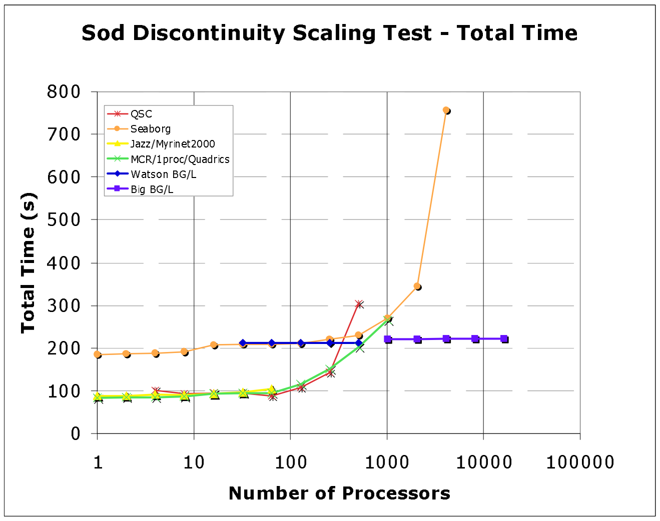

Figure 6: Performance of Flash in Different Parallel Systems (Weak Scaling).

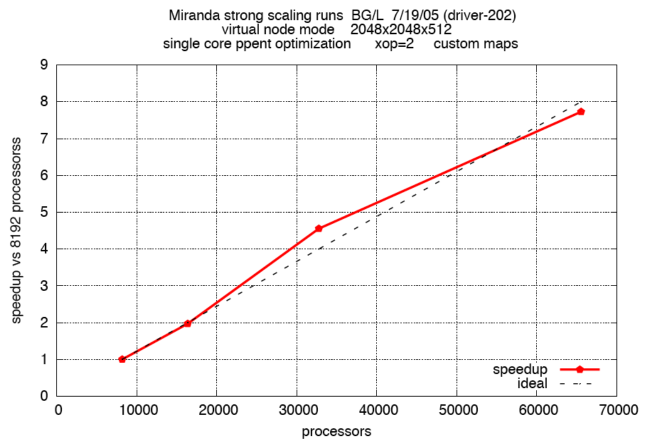

Figure 7: Strong Scaling of Miranda on Blue Gene/L (used with permission from the authors of [4]).

5 Conclusions

In designing the operating system solution for Blue Gene/L, we adopt a software architecture reflecting the hardware architecture of the system. A Blue Gene/L system consists of a core machine with compute and I/O nodes and several auxiliary machines (service node, front-end node, file servers). In the core machine, compute nodes are dedicated to running application processes and I/O nodes are dedicated to providing system services to those processes.

Following that architecture, the operating system for the compute nodes is a lightweight kernel with the mission of launching one (or two) compute process(es) and staying out of the way. The operating system for the I/O node is a port of Linux that implements the file system and TCP/IP functionally that the compute processes need to communicate with the outside world.

This separation of responsibilities leads to an operating system solution that is (1) simple, (2) robust, (3) high performing, (4) scalable, and (5) extensible. The system is robust primarily because its simplicity facilitates design, implementation and test. The high performance and scalability characteristics come from a lack of interference between system services and running applications, which are free to exploit the hardware and run at full speed. Extensibility comes from the simplicity and separation of roles between I/O and compute nodes.

Results from several benchmarks (Linpack [22], HPC Challenge [23], sPPM, UMT2k [1]) and applications (Miranda [4], ParaDis [3], ddcMD [21], Qbox [9], SPaSM [14]) have proved that the operating system for Blue Gene/L supports nearly perfect scalability to over 100,000 processors. The fact that many of these applications have run on a full machine partition for days without interruption confirm the robustness of the software (and hardware, of course). Finally, the several features added to the operating system on customer request (socket support, file system support, and others like performance counter support) demonstrate the extensibility of the solution.

IBM is currently developing a follow-on machine to Blue Gene/L. Significant changes to the hardware will require a redesign of the compute node kernel and a new port of Linux to the I/O nodes. Nevertheless, the basic architecture separating the roles of compute and I/O nodes is being preserved, as it has demonstrated its value in Blue Gene/L.

References

[1] G. Almasi, S. Chatterjee, A. Gara, J. Gunnels, M. Gupta, A. Henning, J.E. Moreira, B. Walkup. *Unlocking the performance of the BlueGene/L supercomputer*. IEEE/ACM SC04, Pittsburgh, PA, November 2004.

[2] ASC/Alliances Center for Astrophysical Thermonuclear Flashes, University of Chicago. See http://flash.uchicago.edu/website/home/.

[3] V. Bulatov, W. Cai, J. Fier, M. Hiratani, G. Hommes, T. Pierce, M. Tang, M. Rhee, R.K. Yates, and T. Arsenlis. *Scalable line dynamics in ParaDiS*. IEEE/ACM SC04, Pittsburgh, PA, November 2004.

[4] A.W. Cook, W.H. Cabot, M.L. Welcome, P.L. Williams, B.J. Miller, B.R. de Supinski, R.K. Yates. *Tera-scalable algorithms for variable-density elliptic hydrodynamics with spectral accuracy*. IEEE/ACM SC05, Seattle, WA, November 2005.

[5] K. Davis, A. Hoisie, G. Johnson, D. J. Kerbyson, M. Lang, S. Pakin and F. Petrini. *A performance and scalability analysis of the BlueGene/L architecture*. IEEE/ACM SC04, Pittsburgh, PA, November 2004.

[6] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, and H. Tufo. *FLASH: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes*. Astrophysical Journal Supplement, 131:273, 2000.

[7] A. Gara, M. A. Blumrich, D. Chen, G. L.-T. Chiu, P. Coteus, M. E. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, T. A. Liebsch, M. Ohmacht, B. D. Steinmacher-Burow, T. Takken, and P. Vranas. *Overview of the Blue Gene/L system architecture*. IBM Journal of Research and Development. Vol. 49, no. 2/3, March/May 2005, pp. 195–212.

[8] D.S. Greenberg, R. Brightwell, L.A. Fisk, A.B. Maccabe, and R.E. Riesen. *A system software architecture for high-end computing*. IEEE/ACM SC97, San Jose, CA, November 1997, pp. 1–15;

[9] F. Gygi, E.W. Draeger, B.R. de Supinski, R.K. Yates, F. Franchetti, S. Kral, J. Lorenz, C.W. Ueberhuber, J.A. Gunnels, J.C. Sexton. *Large-scale first-Principles molecular dynamics simulations on the BlueGene/L*

platform using the Qbox code. IEEE/ACM SC05, Seattle, WA, November 2005.

[10] A. Henning. *BlueGene/L: Improving application memory performance on a massively parallel machine.* M.E. Thesis. Cornell University. 2005.

[11] IBM Corporation. *Blue Gene/L: Application development.* 2006. http://www.redbooks.ibm.com/abstracts/sg247179.html?Open

[12] E. Ipek, B.R. de Supinski, M. Schulz, and S.A. McKee. *An approach to performance prediction for parallel applications*. 2005 Euro-Par, Lisbon, Portugal, August 2005.

[13] Lawrence Livermore National Laboratory. Service request.

[14] S. Louis, B.R. de Supinski. *BlueGene/L: Early application scaling results*. BlueGene System Software Workshop February 23-24, 2005, Salt Lake City, Utah. http://www-unix.mcs.anl.gov/~beckman/bluegene/SSW-Utah-2005/BGL-SSW22-LLNL-Apps.pdf

[15] J. E. Moreira, G. Almási, C. Archer, R. Bellofatto, P. Bergner, J. R. Brunheroto, M. Brutman, J. G. Castaños, P. G. Crumley, M. Gupta, T. Inglett, D. Lieber, D. Limpert, P. McCarthy, M. Megerian, M. Mendell, M. Mundy, D. Reed, R. K. Sahoo, A. Sanomiya, R. Shok, B. Smith, and G. G. Stewart. *Blue Gene/L programming and operating environment.* IBM Journal of Research and Development. Vol. 49, no. 2/3, March/May 2005.

[16] F. Petrini, D. Kerbyson and S. Pakin. *The case of the missing supercomputer performance: achieving optimal performance on the 8,192 processors of ASCI Q*. IEEE/ACM SC03, Phoenix, AZ, November 2003.

[17] R. Ross, J.E. Moreira, K. Cupps, W. Pfeiffer. *Parallel I/O on the IBM Blue Gene/L system*. Blue Gene/L Consortium Quarterly Newsletter. Argonne National Laboratory. 1st quarter 2006. http://www-fp.mcs.anl.gov/bgconsortium/file%20system%20newsletter2.pdf.

[18] K. van der Schaaf. *Blue Gene in the heart of a wide area sensor network.* QCDOC and Blue Gene: Next Generation of HPC Architecture Workshop. Edinburgh, UK, October 2005.

[19] M. Seager. *The BlueGene/L computing environment.* Lawrence Livermore National Laboratory. October 2003. http://www.llnl.gov/asci/platforms/bluegene/papers/16seager.pdf.

[20] K. Singh, E. Ipek, S.A. McKee, B.R. de Supinski, M. Schulz, and R. Caruana. *Predicting parallel application performance via machine learning approaches.* Concurrency and Computation: Practice and Experience. 2006. To appear.

[21] F.H. Streitz, J.N. Glosli, M.V. Patel, B. Chan, R.K. Yates, B.R. de Supinski, J. Sexton, J.A. Gunnels. *100+ TFlop solidification simulations on BlueGene/L.* Gordon Bell Prize at IEEE/ACM SC05, Seattle, WA, November 2005.

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

[23] University of Tennessee. HPC Challenge Benchmark. http://icl.cs.utk.edu/hpcc/.

[24] S.R. Wheat, A.B. Maccabe, R. Riesen, D.W. van Dresser, and T.M. Stallcup. *PUMA: An operating system for massively parallel systems.* Proceedings of the 27th Hawaii International Conference on System Sciences, 1994, pp. 56–65.

[25] S.R. Wheat, A.B. Maccabe, R. Riesen, D.W. van Dresser, and T.M. Stallcup. *PUMA: An Operating System for Massively Parallel Systems.* Scientific Programming, vol 3, 1994, pp. 275-288.

[26] A. Yoo, E. Chow, K. Henderson, W. McLendon, B. Hendrickson, U. Catalyurek. *A scalable distributed parallel breadth-first search algorithm on BlueGene/L.* IEEE/ACM SC05, Seattle, WA, November 2005.