System Support for Many Task Computing

Eric Van Hensbergen
IBM Research
bergevan@us.ibm.com

Ron Minnich
Sandia National Labs
rminnich@sandia.gov

Abstract

The popularity of large scale systems such as Blue Gene has extended their reach beyond HPC into the realm of commercial computing. There is a desire in both communities to broaden the scope of these machines from tightly-coupled scientific applications running on MPI frameworks to more general-purpose workloads.

Our approach deals with issues of scale by leveraging the huge number of nodes to distribute operating systems services and components across the machine, tightly coupling the operating system and the interconnects to take maximum advantage of the unique capabilities of the HPC system. We plan on provisioning nodes to provide workload execution, aggregation, and system services, and dynamically re-provisioning nodes as necessary to accommodate changes, failure, and redundancy. By incorporating aggregation as a first-class system construct, we will provide dynamic hierarchical organization and management of all system resources.

In this paper, we will go into the design principles of our approach using file systems, workload distribution and system monitoring as illustrative examples. Our end goal is to provide a cohesive distributed system which can broaden the class of applications for large scale systems and also make them more approachable for a larger class of developers and end users.

1. Introduction

Within a few short years, we can expect to be dealing with multi-million-thread programs running on million core systems [11]. The programming environments and runtimes we use today on large scale systems are descended from systems built in the 1990s, when the number of cores numbered in the hundreds, not hundreds of thousands. Additionally, emerging applications for these platforms have a much more dynamic execution model than traditional HPC software, requiring the system to adapt to changing workload resource requirements and failure. Today's application developers and end-users desire large-scale systems to provide similar environments, libraries, and tools as to their desktop systems. These factors have led to the exploration of alternate system models which support more general-purpose workloads on large scale computers.

Many Task Computing (MTC) [17], represents a new classification of workload for large scale systems. MTC applications tend to be highly dynamic, requiring allocation and other resource management decisions on 60 second boundaries, not 60 hour boundaries. Applications need to be able to aquire and release computational resources during execution. Furthermore, MTC applications tend to be more tightly coupled than traditional High-Throughput Computing (HTC) applications – requiring a higher degree of scalability from underlying system services providing file system and inter-task communication.

In order to run MTC workloads well on large machines, we can no longer rely on static configuration files nor centralized databases. We must instead move to an infrastructure allowing distributed monitoring and decision making. In short, the parallel and distributed nature of the application will require the systems software and run-time to have similar parallel and distributed properties [20].

We are currently exploring the design and implementation of a distributed operating environment for executing MTC applications on the IBM Blue Gene/P [9]. The Blue Gene/P is designed to scale to at least 262,144 quad-processor nodes. These nodes are subdivided into *psets*, each of which contains a dedicated I/O node and a set of compute nodes. On production machines the ratio of compute to I/O nodes is set at machine build time, and is typically from 1:16 to 1:128.

BG/P utilizes five different types of interconnect: Ethernet, JTAG management, class-routed tree, torus, and barrier. The Ethernet network is present only on the I/O nodes which form the edge of the BlueGene system. The JTAG network is used to load software and monitor execution and hardware health. The barrier network provides a low-latency barrier between nodes in a partition on the system. The class-routed network provides a tree-like topology which is typically used to connect nodes in the same

978-1-4244-2872-4/08/$25.00

pset and is the only data connection between compute and I/O nodes. The high-bandwidth torus network is used by the compute nodes for communication with each other.

Systems such as BG/P are not just giant clusters. They are far more tightly coupled than a cluster, sharing a single clock domain, and constructed with networks that provide latency measured in tens of nanoseconds. In scale they are two orders of magnitude larger than the largest HPC clusters but, at the same time, the effective inter-program latency is an order of magnitude less. The interconnects have diverse and multiple topologies that are, again, very different than the simple fat-tree or cLOS [5] network used on most clusters.

The class-routed tree network has up to 15 programmable class-routes allowing multicast style membership, and can also operate in point-to-point mode. It is typically used for partition-wide collective communication in MPI. It is also normally configured to allow communication among a subset of the compute nodes and an I/O node. In particular, a single link connects the I/O node to a root compute node, which connects to other compute nodes in the same pset in a rough hierarchical fashion (depending on static partition configuration, the tree topology may be irregular and unbalanced). The torus, which is configured as a mesh in smaller configurations, has axial broadcast mechanisms allowing a node to send a broadcast along a particular direction vector. An interesting aspect of both the class-routed tree network and the torus network is that their topologies each form natural aggregation points.

Providing a general-purpose operating environment which scales to such extreme levels while maximizing the use of the underlying resources is extremely challenging. The default system environment on the Blue Gene series does an excellent job of achieving scalability, reliability, and functionality for a certain class of HPC applications. Our goal is not to replace that existing infrastructure but to augment it with an alternative environment for executing a broader class of applications.

We are interested in building upon existing research, exploring novel ways to maximize performance and efficiency of a broader class of applications by leveraging the sheer number of nodes available, the unique properties of the high-performance interconnects, and natural aggregation features provided by the topology of the Blue Gene supercomputer.

The rest of this paper is organized as follows: Section 2 will review existing and alternative system and application models for Blue Gene. Section 3 will describe our general approach to how we will scale and adapt system services to different workload requirements. Section 4 will illustrate the application of our approach through the construction of three system services and Section 5 will discuss our current status and other future areas of exploration.

2. Related Work

The default operating environment on the Blue Gene consists of one or more front-end nodes (typically a conventional system such as a PowerPC running Linux) which end-users may log into to develop their application and submit MPI jobs to the Blue Gene. The MPI jobs are submitted to the compute nodes via the I/O nodes which proxy Ethernet traffic to the tree network. The I/O nodes run a stripped down version of Linux and special applications which allow them to proxy certain system calls (in particular those associated with I/O) from the compute nodes. The compute nodes run a specialized Compute Node Kernel (or CNK) which acts as an executive, typically running a single thread per core. Applications directly access the various Blue Gene interconnects for MPI operations, and transparently perform remote system calls to the I/O node via the CNK.

To accomodate "pleasantly parallel" applications, IBM has released a High Throughput Computing (HTC) [4] model of computation for the Blue Gene/P. This mode is intended to support workloads which require little or no interaction between the executing tasks. Tasks still execute using the Compute Node Kernel (CNK), and aggregate their I/O through daemons running on a Linux-based I/O node. This mode essentially allows end-users to treat BG/P as a gigantic cluster. However, end-users are still required to develop their applications within the CNK infrastructure which presents only a subset of the features of a general purpose kernel.

In response to user demands, some organizations [23] are running Linux on the compute nodes of the system providing applications with a complete general-purpose execution environment. On large scale systems, this results in a 262,144 node Linux cluster to administer. Additionally, since each node is running a full Linux, they incur the associated overhead both from memory footprint and computational resources.

Building upon concepts of HTC mode combined with Compute Node Linux kernels, Raicu et. al. built additional middleware infrastructure [16] to allow subdivision of psets, rapid deployment of new tasks, and aggressive caching to avoid the use of the shared infrastructure (namely the file system and interconnects).

Based on our previous work [14] we believe that simply scaling up Linux is not the answer to unlocking a broader class of applications on large scale machines. The large scale of these systems, and their more tightly coupled nature, argues against running them as "yet another Linux cluster" or, indeed, running Linux on them at all. More importantly, Linux is inherently a single-node operating system – it solves none of the distributed systems problems presented by attempting to run MTC style applications

across millions of cores. What is needed is a self-organizing model for the operating system software whose structure follows the same parallel approach that large-scale applications must follow to work well on the machine.

3. Approach

The core concept of our approach is based on the fact that systems such as BG/P are distributed systems, and as such require a distributed operating system infrastructure to function well. The operating environment foundation for our exploration is based on the Plan 9 research distributed operating system from Bell Labs [15], which provides us with a light-weight, reliable, and extensible kernel. Additionally, Plan 9's inherent resource sharing and organizational mechanisms act as an ideal base for distributed systems.

Plan 9's approach to distributed systems is based on three core principles:

- all resources and services are represented as synthetic file hierarchies
- local and remote resources are accessed using a simple, well-defined protocol (9P)
- each thread's view of the namespace can be dynamically manipulated to provide alternative service or resource providers

All core system services are implemented in this fashion – access to hardware devices (e.g. console, disks, network, audio), process management, network protocol stacks (e.g. TCP/IP, HTTP), as well as application services (e.g. wikis, name service, authentication). The pervasive use of these principles enables all resources and services to be implicitly accessed in a distributed fashion without the use of middleware or specialized distributed APIs [7].

Seemingly complicated distributed operations can be achieved purely through the ability to import/export resources and manipulate individual thread views of the namespace. For instance, remote access to a CPU server (which on a UNIX system would be done with something like rsh or ssh) can be accomplished by building a "sandbox" namespace which contains the file hierarchies representing the terminal's standard I/O, environment, and home directory, and exporting it to a CPU server which mounts the terminal's sandbox in a thread and then overlays its own architecture-dependent binaries and resources. Control of which resources come from the terminal versus the CPU server or some other remote resource can be accomplished with simple namespace scripts. These namespace manipulations affect only the current thread and so aren't visible to other applications running on the terminal or the CPU server.

Since all these operations are accomplished through the file system, they can be implemented as easily from a script as from a compiled application. This liberates the interaction and manipulation of the distributed system from a particular language binding or middleware since all languages have the ability to interact with files. Additionally, the 9P protocol has been ported to Linux, Windows, Mac OSX, and other operating systems – allowing access to Plan 9 resources and services across a wide range of systems.

While this approach provided the key primitives of distributed systems, the relatively small scale of traditional Plan 9 deployments (under 100 nodes including terminals) didn't require much thought about scalability or fault-tolerance. Typical deployments had a dedicated server for file service, another for authentication, a small cluster of compute servers, and a large number of terminals. There was only limited support for redundant servers, resulting in cluster-wide outages when the file-server was taken down for maintenance. Furthermore, even when multiple resources were available (such as the compute service), the user had to manually select which service to use rather than rely on a load balancing service or some other semi-automated policy.

In order to approach issues of scaling system services and applications we plan on exploring treating aggregation as a first-class systems construct. For Plan 9 services this involves incorporating aggregation and relaibility policies with the existing synthetic file servers. The Blue Gene systems software alternatives already provide a static form of aggregation at a coarse level of granularity through the distribution of workloads through the I/O nodes, which also act as proxies for compute node system services [8]. We intend to build upon the success of this technique by enabling more dynamic approaches to aggregation as well as finer levels of granularity for aggregation than a pset.

Our initial approach is similar in many was to Clustered Objects [1]. Each service will be encapsulated into its own proxy server which will run on every node either handling requests locally or forwarding them to remote resources. These proxies will monitor activity and resource utilization, and adapt to changing workloads and environments through partitioning, replication, and distribution. In our implementation, we will scale this infrastructure beyond the node level, allowing end-users to interact with groups of nodes as a unit and facilitating dedication of nodes and node resources to specific system roles.

This will allow us to abandon having central control or even awareness of the behavior of individual nodes. There are simply too many nodes to keep track of them all. In order to eschew centralized monitoring and control, we empower local management of groups of nodes using local knowledge collected from management services. These groups are organized into hierarchies allowing dif-

ferent granularities of control and monitoring. The creation of these groups must be dynamic, responding to changing conditions – not managed by an external or central node which may become scalability bottlenecks.

Following on the earlier work, we can use 9P servers to represent these system services and resources, providing applications with simple, consistent interfaces. The 9P representation gives applications on any node, including the end-user's desktop, transparent access to both local and remote resources. Resources from different sources can also be composed (e.g., one provided locally and another remotely); it is also possible to write a general service that replicates or aggregates suitable underlying services.

Underlying these file server implementations will be an adaptation library which allows multiple back-end implementations of the service offered and facilitate the dynamic switching. In practice, the alternate implementations will be respresentative of different modes of aggregation with details specfic to the service being offered.

Each of these servers will be built to maintain a certain amount of information relating to the activity levels of its clients. Standard practice will be to make this information available via the file system interfaces. Some servers may choose to push this information to specialized monitoring services to be coalesced and propagated with other system management information. The information will be used to collaborate with nearby nodes to balance the distribution of services and requests.

We plan on exploring multiple approaches to the organization and orchestration of nodes. The base case will be the traditional centralized top-down control and monitoring model (see Figure 1). In such a configuration, root nodes (marked FE) will act as central servers and will control node allocation, function as central repositories for monitoring data, and provide file and other I/O services to compute nodes (marked as CN). It is our intuition that such a configuration will not scale well with large number of nodes – particularly in the case of MTC style applications which will swamp the central node(s) with resource requests and/or I/O operations. Even a single static level of aggregation, such as those provided by the Blue Gene psets may not be able to scale well with MTC style workloads.

An alternate approach would be to allow the nodes to organize themselves into ad-hoc groups without regard to the underlying machine topology (see Figure 2). As clients (CN) contact servers (FE), the server may elect to designate a subset of the clients as an aggregate server (AN) and redirect a portion of its workload to that client. Such an approach would likely result in irregular topologies for uneven work distribution. This can be seen in the example figure, where a secondary aggregation node (AN1) has been allocated due to high-levels of requests from a subset of the compute nodes (CN5 and CN6). The nature of the adaptation library will allow servers to redirect pre-existing clients on request boundries. This could also be useful to migrate clients away from a server which is failing or needs to be otherwise reassigned. Servers can be decommisioned at any time depending on workload conditions or user configuration changes.

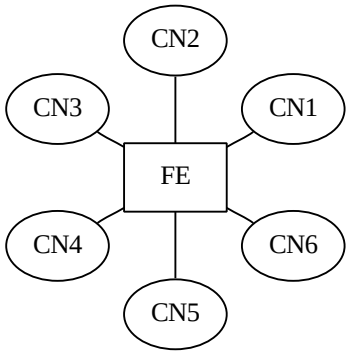

Figure 1. Centralized Aggregation Model

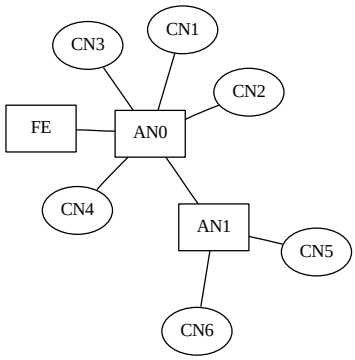

Figure 2. Ad Hoc Aggregation Model

The tree topology of the class-routed network provides a topology with natural points for aggregation as can be seen in Figure 3. By allocating nodes (CN) from leaves at the bottom of the tree, the nodes at key branches (AN) may be reserved for role-specific aggregation duties. We are also interested in exploring novel methods of using the multicast nature of the class-routed tree network to benefit services – particularly for monitoring and file systems.

The nature of the torus topology suggests a different strategy for node allocation and key aggregation node placement. The high-bandwidth and low-latency characteristics of neighbors in the torus network suggest that related tasks should be placed adjacent in the torus topology. Since we anticipate MTC tasks to grow organically, requesting new nodes during run-time, it would make sense to use a sparse allocation for unrelated tasks, such that related tasks have

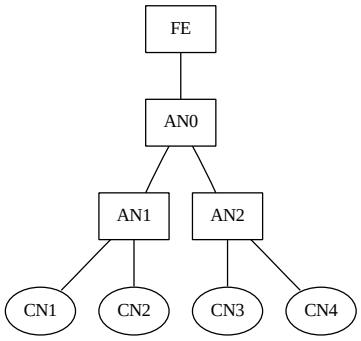

Figure 3. Tree Aggregation Model

a better chance of finding open resources. As such initial task allocation may be done in order to reserve the cube of neighbor nodes for future growth as can be seen in Figure 4.

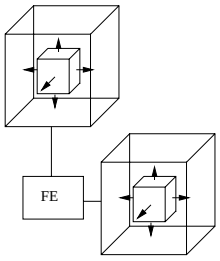

Figure 4. Torus Aggregation Model

Of course, since the BG/P uses both the tree and torus models, there may be interesting placement decisions from analyzing the cross-product of the tree and torus topologies (see Figure 5). There are also likely services (and applications) which will lend themselves more to one style of interconnect than the other.

While our initial model is to locate proxy servers on all nodes which can be designated to have aggregation duty, we feel it will be quite likely that certain services (such as the file server) will outgrow the ability to run as a cooperative task on a computation server. In such scenarios, key nodes may be dynamically designated to particular services in much the same way the current system statically designates I/O nodes. For a given set of nodes, some fraction might be assigned the role of file system caches, while others might be performing redundant computation. This allocation will reduce the amount of interference on compute nodes and allows for specialized kernels which may be optimized for particular services. The modular nature of the Plan 9 kernel allows for easy creation of such appliance-built kernels – and original Plan 9 network organizations included special purpose kernels for file servers. Since the servers are all built with the adaptation library, migration to a role-specific

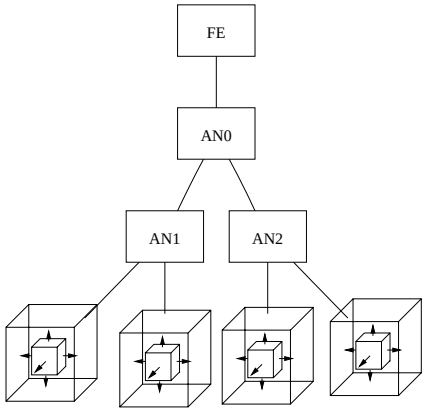

Figure 5. Cross-Product Aggregation Model

server will be transparent to the end-application.

4. Examples

4.1. File Cache

The most straightforward example of these principles can be illustrated through the implementation of an aggregated distributed file cache. In cluster configurations, Plan 9 systems typically use a central file server accessed via the 9P protocol. Since we do not anticipate a single file server will scale to millions of cores, local caches coupled with aggregation of the file service is an early requirement for our prototype. This is particularly important for MTC environments which typically rely on file systems as their principal mechanisms for interaction. Low latency locking and file operations will require hiearchical file service and aggressive distributed caching.

Instead of directly mounting the remote file service, we use a local proxy server built with our adaptive aggregation libraries. As such, it monitors its own resource utilization and can adapt its operation to changing conditions or can be directly reconfigured based on user preferance or instructions from external monitoring and control software. Similar to other system services, the interfaces for monitoring and controlling the proxy are presented via a service-oriented file system obtained by providing the mount command with a specifier parameter as can be seen in Figure 6.

In its most naive mode, the proxy functions as a pass through, using no local resources and forwarding requests to a remote file server. An alternative mode for the proxy is to cache remote file server responses in memory, providing functionality similar to the Linux page cache. A third mode of operation allows the proxy to act as a server to other clients – allowing it to aggregate requests to higher

```
% mount /srv/fsproxy /n/remote
% ls /n/remote
/n/remote/power
/n/remote/bin
/n/remote/lib
/n/remote/spinnevarken
...

% mount /srv/fsproxy /n/remotectl ctl
% ls /n/remotectl
/n/remotectl/status
/n/remotectl/events
/n/remotectl/ctl
```

Figure 6. Mounting alternate views of a file server

level servers and enabling it to share its cache contents. Cache coherence and locking services can be provided by subdividing the responsibility for subsets of the file hierarchy (or indeed the byte ranges of individual files) in much the same manner as Envoy [18]. Replication and redundancy of the data can be managed in a fashion similar to the concepts present in Chord [22]. If resources on the node become scarce, the proxy can request additional nodes assume server responsibilities. Monitoring services, which have a broader view of resource utilization across nodes, as well as knowledge of interconnect topology may dedicate key nodes (including their entire memory resources) to file cache service.

We have implemented a subset of this approach previously during the construction of the Xget [12] file distribution service. Xget was built to deal with the problem of distributing files at boot time on large scale systems – large, in one case, consisting of a 40,000 node cluster. Xget has built-in failure management support. On startup, the individual xget programs self-organize into an ad-hoc tree. If servers fail, the clients can home to a new server. This rehoming can take place in the middle of file transfer – an interrupted transfer restarts on another server. It is also possible to force a rehoming operation if a server gets overloaded. In fact, a server can conscript a client to become a server, and clients of that server can be rehomed to the newly conscripted server. The tree is thus highly dynamic and resilient. Servers try to maintain a pool for 32 or more conscripted clients at any given time; servers also stop serving if they are idle too long. As such, the pool of servers grows and shrinks with demand.

There is no central configuration file for Xget, each processs receives the address of a root server node on startup. In a large-scale system such as BG/P, a simple static aggregation (similar to the existing static allocation of I/O nodes) may be used to establish an initial set of root servers so as not to bottleneck a single fileserver node. The processes contact the root server, and from there are either conscripted to become servers themselves, or are directed to another server. The lack of a static configuration, coupled with ad-hoc algorithms and resilience, provides us with a very high performance and reliable file distribution mechanism. Xget has become the default boot file distributed program for the Perceus cluster management system [10], and also forms a template for how other such services can be structured, as we will see in the discussion of monitoring.

4.2. Monitoring

In order to obtain the distributed control we desire in support of MTC applications, our subsystems will require access to system resource utilization and other statistics at multiple levels of granularity. This will enable us to make local decisions using local utilization data versus having to communicate with central servers. Maintaining this data at different granularity at different levels of the cluster hierarchy should enable a scalable solution to both data collection and access.

Monitoring is another system service which maps naturally to an aggregation framework. Monitoring agents running throughout the system can be organized into hierarchies with each node processing, filtering, and coalescing data from children before reporting to a parent. Servers can aggregate the data from multiple clients as it makes its way up the tree, as in MRNet [19]. This aggregation can greatly reduce the amount of data that flows to the root. Data can even be filtered and dropped. While less data is sent towards the root of the monitoring tree, agents can retain higher fidelity data to allow better local decision making and can serve it on demand to central monitoring services who desire to drill down and collect higher levels of detail.

Following our core design paradigm, the monitoring service is structured as a service-oriented file system. The monitoring servers will offer several views of the collected data – data can be read or written in a composite form (either s-expressions or binary blobs) to a single file, or individual elements can be collected from a synthetic file hierarchy which presents subsets of the data or even individual data elements. Here we can see the power of using the 9P protocol: there is one server-client pipe, but the pipe has “texture”, i.e., each message has a descriptor of which resource that message relates to. We can multiplex many kinds of operations to different objects over a single connection, using the same protocol we use for file distribution and command execution. Data is typically transferred up the tree in a consolidated form to maintain correlation and maximize efficiency.

The hiearchical nature of the file system model provides an extensible, dynamic framework. Various subsystems and applications can create nodes within the synthetic monitoring file system tree to collect and propagate data. We are currently exploring techniques for providing modular filtering and coalescing mechanisms for the data represented in these extensions.

Use of the synthetic file server model allows us to explore both pull and push models of data collection. In the push model, children write to a parent's data file and once it receives telemetry from all its children the parent will in turn propagate its consolidated view of the data. In the pull model, when a parent receives a request for data it will query its children and will only return a result when all children have responded. Timeout thresholds prevent node failure from bringing the monitoring network to a grinding halt.

This design is based in part on our previous experience with implementing the supermon [21] system. It aggregated individual nodes, each running mon servers into supermon servers, which themselves could be aggregated. It was possible and rather easy to build a multi-level tree of supermon servers, but the specification of the connections was determined by a command-line configuration. In order to scale our monitoring system to millions of cores, we'll use the same aggregation libraries discussed earlier to allow a much more fluid allocation of monitoring agents. Additionally, by retaining local views of the state of their children, monitoring agents can enable better local decision making by other aspects of the system such as the distributed workload manager.

4.3. Workload management

We are interested in exploring alternative workload management models as an alternative to the top-down model typified by both the MPI and HTC application communities. While such an approach works for a certain class of application, we wish to support emerging approaches such as processor oblivious work stealing [3]. In such models we can make workload management decisions in a distributed fashion, obviating the need to contact central workload managers or function in a strictly top-down approach. We plan to explore different models of a system primitive which can be parameterized to request application execution on a remote node. Parameters will provide hints as to the relationship with the current node, request near nodes for tightly coupled tasks and far nodes for tasks which won't interact much with the current node.

We have some experience in implementing such a file system for top-down approaches. Xcpu [13] is a 9P server which exposes a file system interface for process creation and management. It provides files which expose information about the node such as machine architecture which is important in heterogenous cluster environments. It also provides control files which the control system writes to set up execution, providing separate files which set arguments, name space configuration, and environment variables. It provides a set of files which can be read/written to provide access to standard input and output of the application. It also provides files which provide information about the state of the session as well as allow direct control of execution.

The Xcpu infrastructure then communicates with these servers in a top-down fashion to distribute exectuables and workload to a cluster of machines. As mentioned previously, we intend to construct a much more dynamic system – but we believe we can use the same core file-server based interface for process execution and control. By using our aggregation libraries to implement this file server we can obtain the same sort of scalability benefits and adaptivity we have discussed previously in the file service and monitoring examples.

We plan on exploring multiple models of node allocation for workload distribution in order to play with alternatives and evaluate their effectiveness in different topologies. Similar to the other examples, every node will run a workload service, with key nodes providing aggregation points to distribute load to leafs. We can support a top-down workload distrbution approach where aggregation trees grow from the root node, but it may make more sense to spray initial workloads to the leaves of the interconnect, preserving nodes at key aggregation points for dedicated system services. Sparse distribution of unrealted tasks within a toroidal topology will reserve neighbor nodes for related tasks. We also plan to experiment with more ad-hoc aggregation approaches which attempt to leverage the cross-product of the tree and torus network topologies. Aggregates of nodes will self-organize in a structure corresponding to network proximity.

5. Status

We have ported Plan 9 kernels to both the Blue Gene/L and the Blue Gene/P hardware and implemented native drivers for the various high performance interconnects. At the moment we are running Plan 9 on both the I/O nodes and the compute nodes, as well as running a hosted version of Inferno [6] on the Front-End Node to provide command and control access from Linux. As a point of comparison, we are also working on porting our kernel to the Kittyhawk [2] infrastructure including a port of the Kittyhawk interconnect driver which abstracts the Tree and Torus interconnects as an Ethernet. Our current effort is focused on optimizing our kernel port, designing and implementing protocols tailored to the properties of the HPC interconnects, and providing SMP support for the Blue Gene platform.

Our next step is implementing and evaluating the designs

discussed in this paper, starting with the file cache, monitoring, and workload management examples. We are in the process of porting several existing applications to our framework, as well as implementing some emerging applications such as unstructured search directly on our infrastructure.

It is already clear to us that a major research challenge will be finding the right balance between compute performance, determinism, reliability, and I/O throughput as we choose how to deploy distributed components throughout the system. Another major challenge will be finding ways to effectively leverage this new model with existing and emerging applications. While most applications should benefit from increased scalability of underlying system services, we would also like to explore ways for applications to exploit a similar self-organizing distribution strategy during their execution. It is our belief that this will broaden the applications base for existing machines as well as help pave the way for exascale systems.

6. Acknowledgements

The ideas presented in this paper were developed in collaboration with David A. Eckhardt at Carnegie Mellon University, Charles Forsyth at Vitanuova Ltd, and Jim McKie of Bell Labs. This material is based upon work supported by the Department of Energy under Award Number DE-FG02-08ER25851. We have also benefited from access to the Argonne National Lab Blue Gene/P as part of the Department of Energy INCITE program. The Argonne support team has been particularly patient and helpful through the bootstrapping process which often has played havoc with the hardware and their control system. The port of Plan 9 to the Blue Gene hardware would not have been possible without the support of IBM's Blue Gene research team as well as contributions and support from Volker Strumpen, Amos Waterland, Volkmar Uhlig, and Jonathan Appavoo.

References

[1] J. Appavoo. Clustered objects. Technical report, Department of Computer Science, University of Toronto, 2005.

[2] J. Appavoo, V. Uhlig, and A. Waterland. Project kittyhawk: building a global-scale computer: Blue Gene/P as a generic computing platform. In *ACM SIGOPS Operating System Review*, 2008.

[3] J. Bernard, J.-L. Roch, and D. Traore. Processor-oblivious parallel stream computations. In *Proceedings of Parallel, Distributed and Network-Based Processing*, 2008.

[4] T. Budnik, B. Knudson, M. Megerian, and S. Miller. High throughput computing on IBM's Blue Gene/P.

[5] C. Clos. A study of non-blocking switching networks. In *Bell Systems Technical Journal*, volume 32, pages 406–424.

[6] S. Dorward, R. Pike, D. Presotto, D. Ritchie, H. Trickey, and P. Winterbottom. The Inferno Operating System. *Bell Labs Technical Journal*, 2(1):5–18, Winter 1997.

[7] C. Forsyth. The ubiquitous file server in Plan 9. *http://plan9.bell-labs.com/sources/contrib/uriel/doc/charles/servers.pdf*, February 2005.

[8] A. Gara and et.al. Overview of the IBM Blue Gene/L system architecture. In *IBM Journal of Research and Development*, volume 49, 2005.

[9] IBM Blue Gene team. Overview of the IBM Blue Gene/P project. In *IBM Journal of Research and Development*, volume 52, 2008.

[10] Infiscale. Perceus cluster management system. *http://www.perceus.org*.

[11] M. Leininger. Application and algorithm challenges for future petascale to exascale computing architectures. In *OASCR AMR Future Architectures Panel*, 2007.

[12] R. Minnich and L. Ionkov. Xget. *http://www.xcpu.org/xget*.

[13] R. Minnich, A. Mirtchovski, and L. Ionkov. XCPU: a new, 9P-based, process management system for clusters and grids. In *Proceedings of IEEE International Conference on Cluster Computing*, 2006.

[14] R. Minnich, M. J. Sottile, S.-E. Choi, E. Hendriks, and J. McKie. Right-weight kernels: an off-the-shelf alternative to custom light-weight kernels. In *ACM SIGOPS Operating Systems Review*, volume 40, 2006.

[15] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, and P. Winterbottom. Plan 9 from Bell Labs. *Computing Systems*, 8(3):221–254, Summer 1995.

[16] I. Raicu, Z. Zhang, M. Wilde, and I. Foster. Enabling loosely-coupled serial job execution on the IBM BlueGene/P supercomputer and the SiCortex SC5832.

[17] I. Raicu, Z. Zhang, M. Wilde, I. Foster, P. Beckman, K. Iskra, and B. Clifford. Toward loosely coupled programming on petascale systems. In *IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SuperComputing/SC)*, 2008.

[18] R. G. Ross. Cluster storage for commodity computation. Technical Report 690, University of Cambridge Computer Laboratory, 2007.

[19] P. C. Roth, D. C. Arnold, and B. P. Miller. Mrnet: A software-based multicast/reduction network for scalable tools. In *Proceedings of the International Conference on SuperComputing*, 2003.

[20] B. Smith. The quest for general purpose parallel computing, 1994.

[21] M. J. Sottile and R. G. Minnich. Supermon: A high-speed cluster monitoring system. In *CLUSTER '02: Proceedings of the IEEE International Conference on Cluster Computing*, page 39, Washington, DC, USA, 2002. IEEE Computer Society.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. pages 149–160, 2001.

[23] ZeptoOS Team. ZeptoOS: The small Linux for big computers. *http://www-unix.mcs.anl.gov/zeptoos*, 2008.