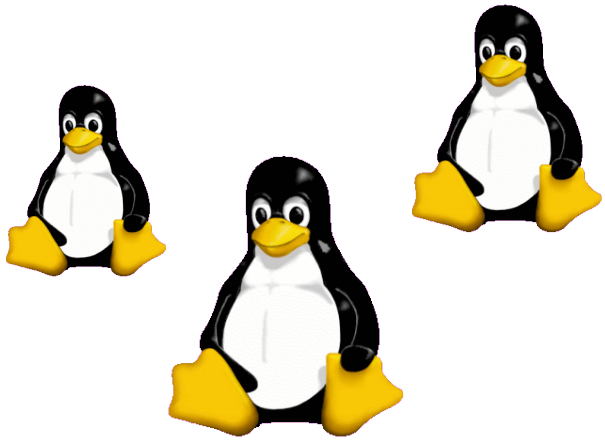
Project Final Report (including no cost extension)
SC Program Announcement Title and Number: Lab 07-23
FAST-OS II: Operating Systems and Runtime Systems at Extreme Scale

HPC-Colony II

Terry Jones, ORNL, Lead Principal Investigator
Laxmikant Kalé, UIUC, Co-Principal Investigator
José Moreira, IBM, Co-Principal Investigator



Program office: Office of Advanced Scientific Computing Research
Program Manager: Dr. Lucy Nowell

Table of Contents

Executive Summary

HPC Colony II has been a 36-month project focused on providing ***portable performance*** for leadership class machines—a task made difficult by the emerging variety of more complex computer architectures. The project attempts to move the burden of portable performance to ***adaptive system software***, thereby allowing domain scientists to concentrate on their field rather than the fine details of a new leadership class machine.

To accomplish our goals, we focused on adding intelligence into the system software stack. Our revised components include: new techniques to address OS jitter; new techniques to dynamically address load imbalances; new techniques to map resources according to architectural subtleties and application dynamic behavior; new techniques to dramatically improve the performance of checkpoint-restart; and new techniques to address membership service issues at scale.

Key Findings

⇒ *The technique of Coordinated Scheduling has been shown to be an effective strategy for removing negative consequences of OS Jitter*. Results from ORNL's Jaguar machine demonstrate that coordinated scheduling can give similar performance to core-specialization while improving overall efficiency through consuming less nodes

⇒ *The study has identified that double in-memory checkpoint restart strategies are able to effectively handle small node-count faults and incur small overhead in the non-fault case.* This approach was scaled up to 32 thousand cores on a BG/P machine with a molecular dynamics benchmark.

⇒ *A dynamic load-balancing framework leverages the ability of a runtime system to adaptively react to changes in the system and keep making progress in the application at a fast rate.* We explored two different scenarios where the conditions of the system suddenly change. The first scenario consists in having node failures. The runtime system can efficiently reconstruct the lost tasks and recover the work lost. However, the load-balancing framework dramatically reduces the memory overhead of fast recovery techniques. The second scenario includes thermal variations in different portions of a cluster. The runtime system reacts to this situation by modifying the frequency of hot cores and moving tasks away from them. The load-balancing framework ensures the work is evenly distributed across the system considering the difference in frequency. With this approach, it is possible to reduce the cooling costs of supercomputing facilities by managing the temperature of individual processors.

⇒ *Task mapping proved to be a fundamental piece in improving the performance of scientific applications.* Using the topology of a cluster, it is possible to have the load balancing framework deciding the map of tasks to nodes to avoid network congestion and to reduce communication costs.

⇒ *The research has revealed that a membership service based on an overlay network peer-to-peer technology can efficiently support 1 million nodes.* This enables the implementation of a new class of resiliency-aware runtime on large scale HPC systems.

⇒ In addition, the research demonstrated that *group communication services, such as publish subscribe, can be made scalable enough to support 1 million nodes*. Scalable publish subscribe provide the flexible and dynamic communication channels that can be used for runtime load balancing and fine grained monitoring.

1. Project Abstract

Motivation & Goals

HPC Colony II seeks to provide ***portable performance*** for leadership class machines. Our strategy is based on ***adaptive system software*** that aims to make the intelligent decisions necessary to allow domain scientists to safely focus on their task at hand and allow the system software stack to adapt their application to the underlying architecture.

Team

Terry Jones[1], Laxmikant Kalé[2], Eliezer Dekel[3], Benjamin Mandler[3], Celso Mendes[2], Xiang Ni[2]
Esteban Meneses[2], Harshitha Menon[2], José Moreira[3], Yoav Tock[3], Lukasz Wesolowski[2], Yanhua Sun[2]

[1]Oak Ridge National Lab
Mailstop 5164
Oak Ridge, TN 37831

[2]University of Illinois
201 N. Goodwin Avenue
Urbana, IL 61801

[3]International Business Machines
1101 Kitchawan Rd
Yorktown Heights NY 10598

Budget

	Year 1	Year 2	Year 3	Total
ORNL	$187,000	$187,000	$187,000	$561,000
IBM	$163,000	$163,000	$163,000	$489,000
UIUC	$250,000	$250,000	$250,000	$750,000
Totals	$600,000	$600,000	$600,000	$1,800,000

Principal Investigators

Lead Principal Investigator
Terry Jones
Oak Ridge National Lab
PO Box 2008 / Mailstop 6164
Oak Ridge, TN 37831
Tel: 865-241-5764
Email: trj@ornl.gov

Co-Principal Investigator
Laxmikant Kalé
Univ. of Illinois at Urbana-Champaign
201 N. Goodwin Avenue
Urbana, IL 61801
Tel: 217-244-0094
Email: kale@cs.uiuc.edu

Co-Principal Investigator
José Moreira
International Business Machines
1101 Kitchawan Rd
Yorktown Heights NY 10598
Tel: 914-945-1709
Email: jmoreira@us.ibm.com

Funding Period

Funding Period: Sept 15, 2009 – Sept 31, 2013 (includes 12 month no-cost extension)

Impact Statistics

Publications: 35 proceeding articles / 5 journal articles / 3 book chapters / 1 dissertation / 5 other

Software Products Impacted: 4 [1]

Academic Support: 4 undergrads / 7 grad students / 1 doctorate awarded / 2 post docs supported

Product Awards: 1 (winner of productivity & performance category, HPC Challenge 2011)

[1] Charm++ 6.5.0, OpenMPI 1.4.4, Linux 2.6.16.60, SpiderCastCPP 1.0

[2] During the course of our project, the Jaguar Cray XT5 machine has been upgraded to the Titan Cray XK6 machine. More information on Jaguar and Titan's architecture is available at the NCCS website: http://www.nccs.gov

[3] T. Hoefler, T. Schneider, and A. Lumsdaine. Characterizing the Influence of System Noise on Large-Scale Applications by

2. Background and Motivation

HPC Colony II is a 36-month project focused on providing ***portable performance*** for leadership class machines—a task made difficult by the emerging variety of more complex computer architectures. The project attempts to move the burden of portable performance to system software, thereby allowing domain scientists to concentrate on their field rather than the fine details of a new leadership class machine. An overview matrix is provided in Table 1.

Collaborators	OAK RIDGE National Laboratory Terry Jones, Project PI	 Laxmikant Kalé, UIUC PI	IBM José Moreira, IBM PI
Objectives	▪ Provide technology to make portable scalability a reality. ▪ Remove the prohibitive performance issues of full POSIX APIs and full-featured operating/ runtime systems. ▪ Enable easier leadership-class level scaling for domain scientists through removing key system software barriers; eliminate the need for "roll-your-own" fault tolerance strategies.		
Approach	▪ Automatic and adaptive load-balancing plus fault tolerance. ▪ High performance peer-to-peer and overlay infrastructure. ▪ Address issues with Linux through coordinated scheduling.		
Challenges	▪ Computational work often includes large amounts of state which places additional demands on successful work migration schemes. ▪ For widespread acceptance from the Linux community, the effort to validate and incorporate HPC originated advancements into the Linux kernel must be minimized.		
Impact	▪ Full-featured environments allow for a full range of programming development tools including debuggers, memory tools, and system monitoring tools that depend on separate threads or other POSIX API. ▪ Automatic load balancing helps correct problems associated with long running dynamic simulations including both performance and fault tolerance issues. ▪ Coordinated scheduling removes the negative impact of OS jitter from full-featured system software		

Table 1: Overview Matrix for Colony II

Performed as a collaboration of three organizations (Oak Ridge National Laboratory, IBM Corporation, and the University of Illinois at Urbana-Champaign), the HPC Colony II Project received primary funding through the DOE Office of Advanced Scientific Computing Research (DOE/ASCR). In addition, 50% matching funds were provided by IBM Corporation for their involvement, and additional support was provided by Oak Ridge National Laboratory and the University of Illinois at Urbana-Champaign.

Our strategy is based on adaptive technology that aims to make the intelligent decisions necessary to allow domain scientists to safely focus on their task at hand and allow the system software stack to adapt their application to the underlying architecture. The growing complexity and diversity of leadership class computer architectures demands a low entry barrier when domain scientists migrate their application codes to new machines. Similarly, efficient performance on diverse machines is an increasingly important issue. The requirement, therefore, is to modify the familiar system software stack to provide an HPC stack that presents a minimal barrier to BOTH portability AND performance.

To help realize our project goal of ***portable performance***, the HPC Colony II project is focused on six interrelated topic areas which utilize adaptive technology to make portable scalability much more feasible. Our six topic areas are:

- Reduce performance consequences from fault tolerance
- Provide scalable membership, monitoring, & communication services
- Investigate innovative ways to provide dynamic load balancing
- Improve the resource management interface between center batch scheduling & on-node system software
- Enable broad application sets on most capable machines
- Enable Linux kernel advances for extreme scale systems

These areas are addressed through a combination of system software strategies. We modify the most familiar and popular operating system in the HPC market space, Linux, to be suitable for extreme node counts. We establish an adaptable runtime system that is able to address the critical issues of fault tolerance and load balancing. Finally, we establish a high-performance open-source membership service that removes the necessity of repeating this critical functionality in multiple places (file systems, job schedulers, sys admin tools, …) as is standard practice today.

The Colony kernel is a modified version of the Linux which runs on Cray's largest machines. Unlike typical Linux, the Colony kernel is able to provide the familiar interfaces of Linux without the problematic scaling issues of OS jitter. [OS Jitter is explained below.]

Charm++ is an adaptive runtime system -- a runtime library to let C++ objects communicate with each other efficiently. Charm++ is a way of writing a program (a programming model). Charm++ is not a programming language in and of itself. Instead, Charm++ uses the C++ programming language as it's base language. Charm++ adds additional functionality and structure on top of C++ that allows the programmer to solve the problem at hand. With processor virtualization, the user divides the problem into a large number of objects without considering the actual number of physical processors. Each object is called a virtual processor. The user views the program as a set of virtual processors that interact with each other. The task of mapping virtual processors to physical processors is left to the Charm++ runtime system. Charm++ organizes the virtual processors as collections of C++ objects that interact via asynchronous method invocations.

Our membership services software is called SpiderCast. Developed by IBM Research, SpiderCast provides best in class performance while permitting the extreme scales envisioned for Exascale machines.

3. Research Review

3.1 Kernel & Kernel Support Advances (Colony lead: ORNL)

Linux is desirable because it is utilized at many universities, it provides a rich set of tools that have been developed over the years, and it offers a great deal of functionality through its expansive API. However, Linux has been shown to introduce performance issues due to OS jitter or noise [Hoefler10, Jones03, Nataraj07].

HPC Colony solves the performance problem by providing effective global time synchronization in software together with an advanced kernel scheduler that is able to optimize global machine performance. This is accomplished by modifying the Linux kernel to have parallel awareness of other nodes in the supercomputer. This allows us to perform *coordinated scheduling*: the machine is managed for parallel throughput with a global perspective instead of the typical local node perspective.

HPC Colony II developed the first co-scheduling Linux kernel designed for High Performance Computing. Results were obtained on ORNL's XT5/XK6 system[2]; first with the normal scheduling Linux 2.6.16.60 operating system, then with the coordinated scheduling Linux 2.6.16.60 operating system described above. Results for normal Linux scheduling showed noticeable variability from test to test at scales of 10K cores. This was expected and coincides well with results obtained from Hoefler et al. [Hoefler10] and Sottile et al. [Sottile04], as well as our previous work. Performance measurements were then obtained using the coordinated-scheduling policy and modified operating system. It was immediately clear for this workload that the coordinated scheduling provided a significant performance improvement, both in terms of average execution time and in terms of variability between runs. Finally, an additional set of performance numbers with the normal operating system were measured. The last set of normal operating system results closely matched the set of results obtained before the co-scheduled kernel results taken in the middle.

Just as the hardware of supercomputers has evolved over time, the applications that use them have also evolved. Today, parallel programs are frequently implemented in the Bulk-Synchronous Single-Program-Multiple-Data (SPMD) programming model. For this programming model, computation consists of one or more *cycles* or *timesteps*. Each cycle may contain one or more *synchronizing collective* operations – an operation in which a set of processes (frequently every process) participates and no single process can continue until every process in the set has participated. Examples of synchronizing collective operations from the Message Passing Interface (MPI) interface are MPI_Barrier, MPI_Allreduce, and MPI_Allgather [MPI-Forum]. For example, a parallel application designed to simulate climate may use the MPI_Allreduce operation to find the maximum pressure present in an array distributed over all nodes; note that the overall maximum pressure cannot be determined until every node has contributed its maximum. Synchronizing collective operations pose serious challenges to scalability since a single instance of a laggard process will block progress for every other process.

Synchronizing collectives are common in parallel applications. Even though today's most prevalent operating systems, including Linux, do not include synchronizing collectives -- operating systems may determine the scalability of a parallel application running in user-space if the parallel application contains synchronizing collectives. The *scalability* of an operating system is referring to the operating system's ability to support a parallel application without introducing scaling issues for the parallel application. Adverse performance associated with synchronizing collectives would seem to restrict their usage, but unfortunately synchronizing collective operations are required for a large class of parallel algorithms. [6]

Operating systems impact synchronizing collectives in the following way. A *cascading effect* results when one laggard process impedes the progress of every other process. The cascading effect has significant operating system implications and proves especially detrimental in an HPC context: while operating systems may be considered very efficient in a serial context, even minimal system and/or daemon activity proves disastrous due to the cascading effect in the large processor count parallel environment common in HPC centers. When interruptions occur on a subset of the computer nodes used for a parallel application during a synchronizing collective (e.g. an interruption for operating system activity such as a file system buffer flush or even a TLB miss), the degree of overlap is a key component in determining the performance impact of the interruption event on the synchronizing collective operation.

[2] During the course of our project, the Jaguar Cray XT5 machine has been upgraded to the Titan Cray XK6 machine. More information on Jaguar and Titan's architecture is available at the NCCS website: http://www.nccs.gov

Figure 1 graphically portrays two separate runs of an eight-way parallel application with time as the x-axis. In the top instance, system activity (denoted as dark-green rectangles) occurs at purely random times. As a result, operations that require all eight processors to make progress are able to go forward only when grey-marble is present across all eight processors vertically (at one point in time). The beige-sand rectangles show those periods in time when the application is running across all 8 processors. In the bottom portrayal of Figure 1, the same amount of system activity occurs (there is the same total amount of dark-green) but it is largely overlapped. This means much more time is available for parallel activities requiring all processors, as shown by the larger green rectangles.

For clusters comprised of nodes with more than one core, both inter- and intra-node overlap is an issue. Notice that if the eight cores in Figure 1 are spread across two 4-core nodes, it is desirable to ensure overlap between nodes as well as on-node. The bottom run shows very good on-node overlap of operating system interference, but does not fully achieve cross-node overlap of operating system interference.

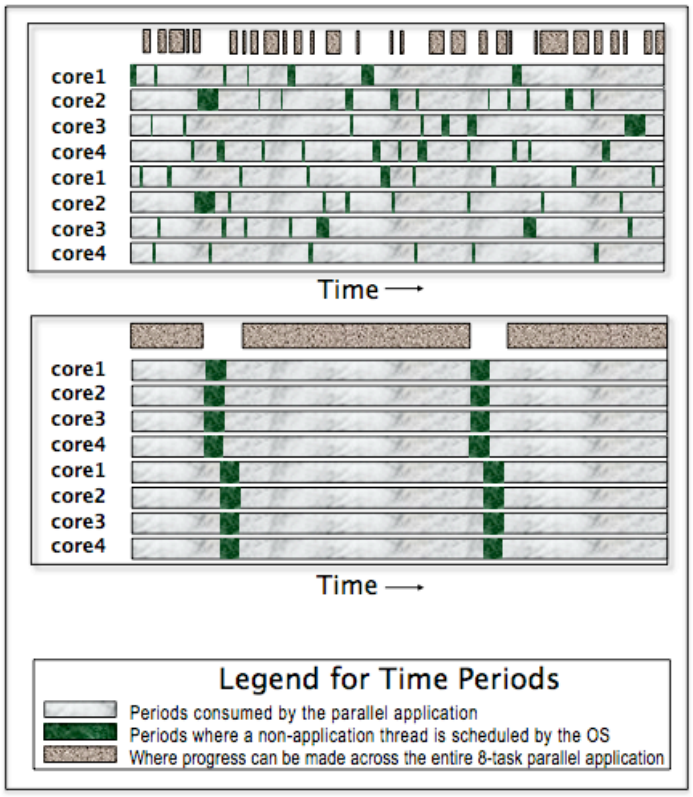


Figure 1: Coordinated and uncoordinated schedulings. The above figure depicts two schedulings of the same eight-way parallel application. In the lower depiction, co-scheduling increases the efficiency of the parallel application as indicated by the larger amount of time periods where progress can be made across the entire 8-task parallel application. The top legend is grey-marble; the middle legend is dark-green, and the bottom legend is beige-sand.

The Colony Linux kernel achieves high scalability through *coordinated scheduling* techniques and other strategies aimed at reducing operating system overhead. Coordinated scheduling (also referred to as *parallel aware scheduling*) seeks to reduce the impact of operating system noise. This is accomplished by increasing the overlap of interruption activity (e.g., increasing the overlap of 'grey-marble activity' in Figure 1).

Colony establishes two alternating intervals for activity across the entire parallel computer. During the longer interval, the parallel application is scheduled (e.g., the 'grey-marble activity' in Figure 1). This is accomplished by modifying the Linux scheduler to favor the parallel application with a high scheduler priority. During a shorter interval, other necessary activities such as health-monitoring daemons, parallel file system daemons, and so on, are scheduled (e.g., the 'dark-green activity' in Figure 1). During this period, the normal Linux algorithms are used allowing delayed operating system activities to make progress. In this way, the federated cores are said to be *co-scheduled* and interfering interruptions from daemon activity are minimized.

Figure 2 at right compares the new technique of coordinated scheduling with a normal (unmodified) environment, an environment that utilizes core-specialization, and an environment that uses core-specialization + processor pinning at 30,000 cores. The Y-axis is time to solution (smaller is better). Coordinated scheduling gave a 2.87x improvement over a normal (unmodified) environment. It also gave better performance than core-specialization despite using much fewer nodes (core specialization reserves one core per node to handle OS jitter). Coordinated scheduling was also able to outperform the alternate technique of core specialization while providing higher machine utilization.

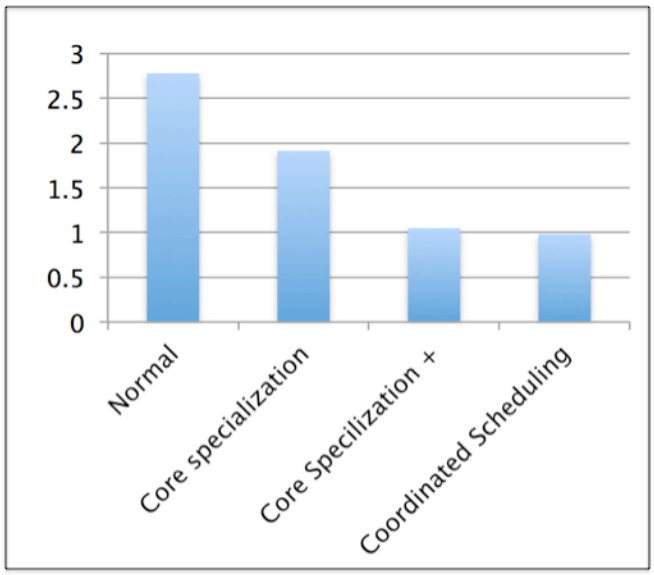


Figure 2: Performance for (a) Normal Linux scheduling; (b) Scheduling with Core-specialization plus additional nodes; (c) Scheduling with Core-specialization and processor pinning plus additional nodes; (d) Coordinated scheduling. Less time is better.

The left half of Figure 3 below depicts the normal Linux results while the right half of Figure 3 depicts the co-scheduled results. As described earlier, the benchmark employed for this testing results in a single number corresponding to a unit of execution time, the lower the better. The graphs indicate shorter durations (better performance) for the co-scheduled kernel.

The best observed time from all experiments was 0.44. The average execution time for the co-scheduled kernel was **0.56**, which compares to **1.60** with the Normal Scheduled kernel. An improvement of 285%. Moreover, the variability was *much improved* with the co-scheduled kernel. The standard deviation for the Normal Scheduled samples was **5.32**; this compares to **0.20** for the co-scheduled kernel.

With a standard deviation larger than the average, it is clear that the samples do not follow a Gaussian distribution. In fact, the distribution of samples for the co-scheduled kernel has a very prominent peak near the average measurement, and a short tail of longer times. However, the distribution for the Normal Scheduled kernel has a much broader peak and a very long tail of outlier samples with much longer times. These results can be seen in Figure 4. In the left histogram, the worse performing outliers are circled in red, and the two most are off the charts at 8.88 and 60.77. This variability is in stark contrast to the co-scheduled results in the right histogram of Figure 4.

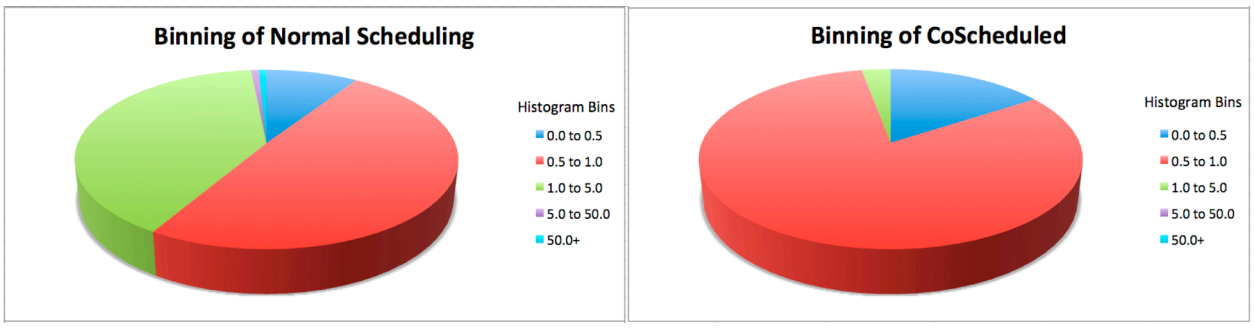


Figure 3. Coordinated and uncoordinated schedulings. The above figure portrays histogram bins in a pie-chart to provide an indication of the relative timing of runs. The top chart gives results without scheduling, and bottom chart gives results for coordinated scheduling.

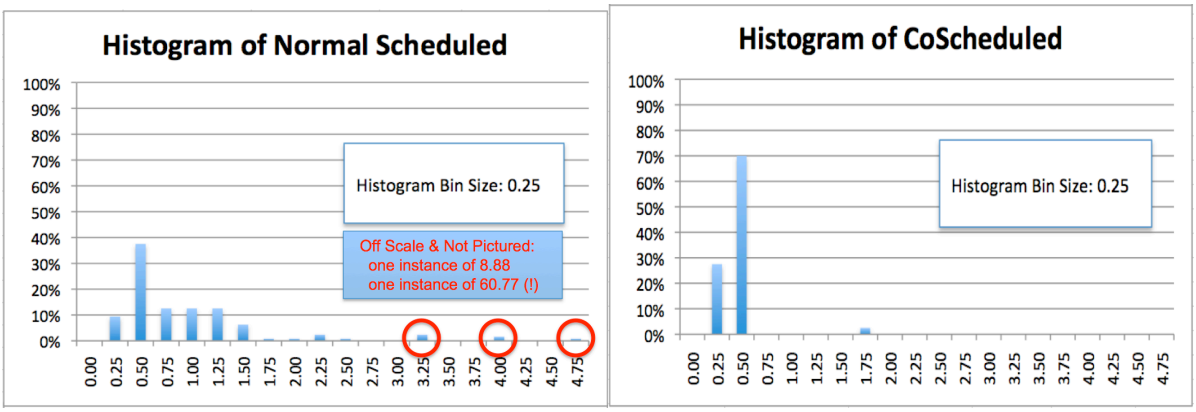


Figure 4. Coordinated and uncoordinated schedulings. The above figure portrays a histogram of runs with and without coordinated scheduling. The lower histogram includes coordinated scheduling.

In context, the 285% speedup is good news for that class of applications impacted by synchronizing collectives, but it should be noted that overall application performance will depend upon many factors beyond synchronizing collective performance. Yet the 30% overall application slowdown reported by Nataraj et al. [Nataraj07] and Ferreira et al. [Ferreira08] indicates a significant amount of speedup may be realized by an entire application when noise effects are minimized.

To achieve this significant performance increases for such bulk-synchronous applications, the coordinated scheduling mechanism in the kernel requires a globally synchronized clock. A second topic area of our work was to investigate coordinated scheduling for machines without hardware support for global synchronized clock (e.g. Cray XT5, XK6, and XK7 architectures). We developed an improved software-based clock synchronization scheme that provides high precision time agreement among distributed memory nodes. The technique is designed to minimize variance from a reference chimer during runtime and with minimal time-request latency. Our scheme permits initial unbounded variations in time and corrects both slow and fast chimers (clock skew). An implementation developed within the context of the MPI message passing interface was designed, and time coordination measurements are presented below. To investigate our design, we began by measuring how nine nodes vary from a reference source when only NTP is employed. Figure 5 presents data for uncorrected variance and variance corrected with a linear fit.

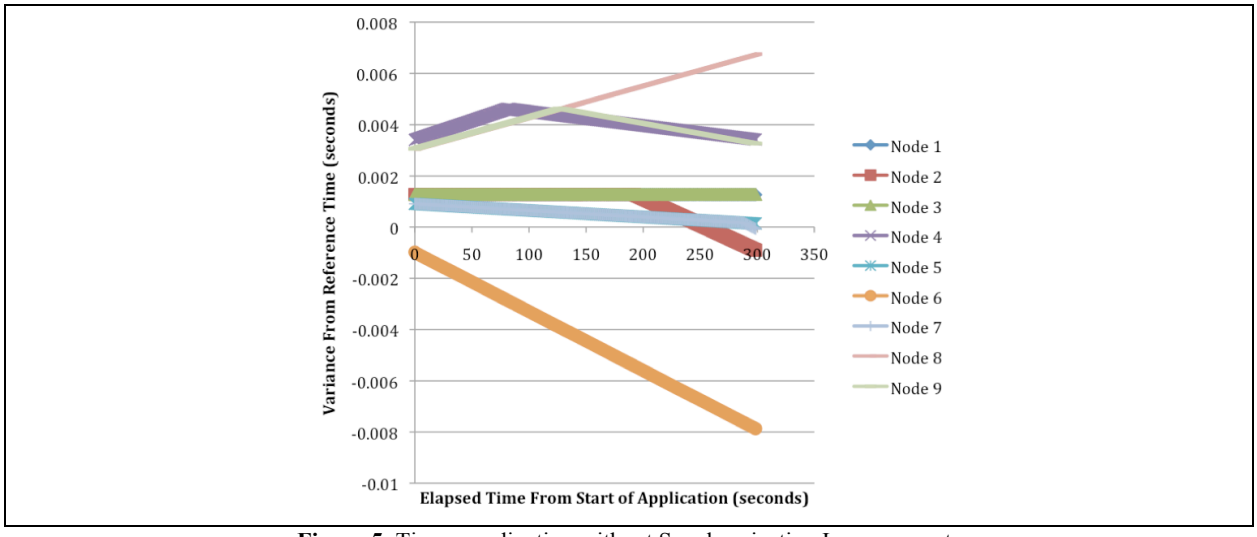


Figure 5: Time coordination without Synchronization Improvements

Given that the mean time variance for even a small set of nodes can reach 20.0 milliseconds under standard Network Time Protocol (NTP), the Colony project designed a new software-based synchronization protocol suitable for high performance computing environments. Several studies have sought to quantify the magnitude of scalability issues associated with operating system noise[3] [4]. For example, the Tau team at the University of Oregon has reported 23% to 32% increase in runtime for parallel applications running at 1024 nodes and 1.6% operating system noise. More recently, Ferreira et al. confirmed that a 1000 Hz 25μs noise interference (an amount measured on a large-scale commodity Linux cluster) can cause a 30% slowdown in application performance on ten thousand nodes. By tightly synchronizing the clocks on the compute nodes, it is possible to extend the system software to support co-scheduling (an effective technique to reduce the effects of noise on a parallel computation).

With our ALCC allocation, the Colony team investigated a new point-to-point synchronization protocol (our previous methods required collective operations). Figure 6 shows a histogram of results for the new synchronization protocol on 20,480 Titan processors.

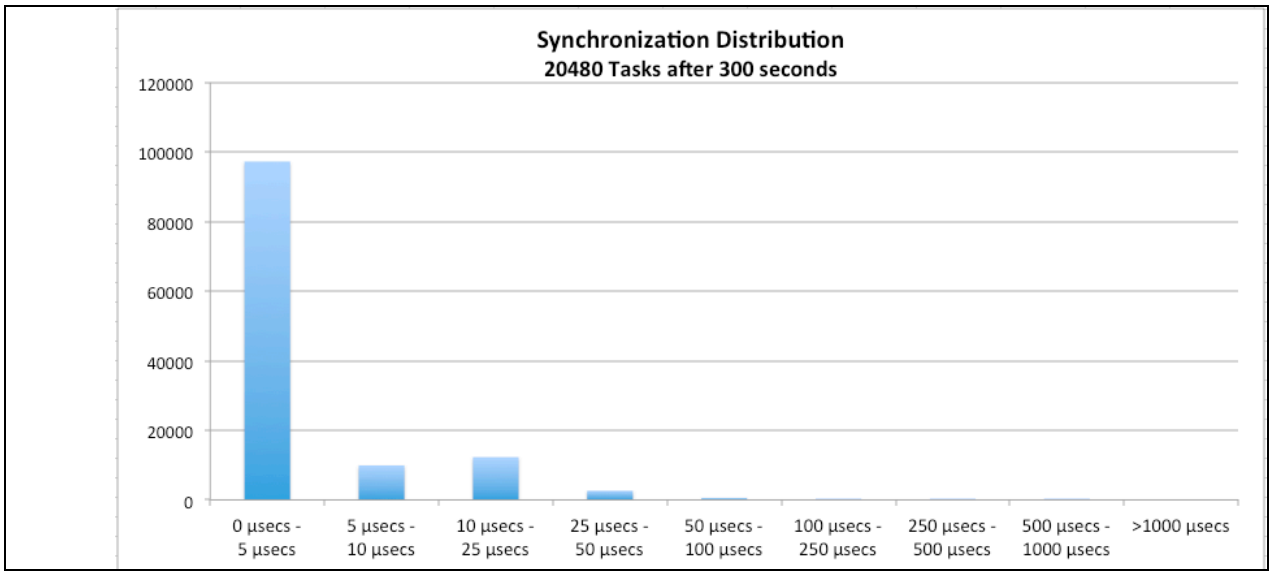


Figure 6: Time synchronization with Coordination improvements after 5 minutes. These figures represent the variance for synchronizing up to 20,480 nodes.

3.2 Fault Tolerance Advances (Colony lead: Univ. of Illinois)

During this period, we have improved the various fault tolerance mechanism in Charm++ runtime system. Both schemes, checkpoint/restart and message logging, are strong candidates to provide resilience at exascale. As such, our main efforts were invested in evaluating how well each scheme would tackle the challenges of large-scale systems.

Our double in-memory checkpoint/restart mechanism was tested on larger systems. Checkpoint based fault tolerance methods are effective approaches at dealing with faults. With these methods, the state of the entire parallel application is checkpointed to reliable storage. When a fault occurs, the application is restarted from a recent checkpoint. Leveraging Charm++'s parallel objects for checkpointing, two variations of checkpointing schemes, a disk-based and a double in-memory checkpointing schemes, are incorporated in the production distribution of Charm++. One of the unique features of both schemes is

[3] T. Hoefler, T. Schneider, and A. Lumsdaine. Characterizing the Influence of System Noise on Large-Scale Applications by Simulation. *In International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*, Nov. 2010.

[4] Terry Jones, Shawn Dawson, Rob Neely, William Tuel, Larry Brenner, Jeff Fier, Robert Blackmore, Pat Caffrey, Brian Maskell, Paul Tomlinson, and Mark Roberts, *Improving the Scalability of Parallel Jobs by adding Parallel Awareness to the Operating System*. Proceedings of Supercomputing 2003, Phoenix, AZ, November 2003.

that the program can be restarted on smaller number of available processors as a result of failure. Furthermore, the application will continue to execute on the remaining processors without much performance penalty after automatic load balancing. In particular, compared to the disk-based method, the double in-memory checkpointing scheme takes advantage of the fast memory access for checkpointing to both local memory and remote memory through high speed interconnect.

Recently, we worked on further minimizing the checkpoint and restart overhead by applying more efficient collectives for barriers. We demonstrated the in-memory checkpointing scheme using MPI on very large scale supercomputers. One obstacle for demonstrating fault tolerance on MPI applications is that the queueing system on supercomputers will kill a job when a process fails. Without the support of the queueing system, we developed a scheme that mimics a failure of a process without actually killing it. This is implemented as a DieNow() function, which users insert at any place in their program to trigger a failure. When DieNow() function is called by the program, the process will hang and stop responding to any communication as if it had died. Charm++ will pick up a spare processor from a pool and restart the application from the recent checkpoint in memory.

To demonstrate the performance and scalability of the newly optimized double in-memory checkpointing scheme, we used two benchmarks, which are leanMD (a molecular dynamicbenchmark) and Jacobi (a 7-point stencil benchmark). In the experiments, we measured the overhead of checkpoint and restart of these two benchmarks on a BlueGene/L machine. The results are shown in figures 7 and 8. We could see the checkpoint time scales well in both applications with small and large memory footprint. In particular, the restart time of leanMD simulating an 1-million atom system only increases from 0.08 seconds on 2K cores to 0.38 seconds on 32K cores.

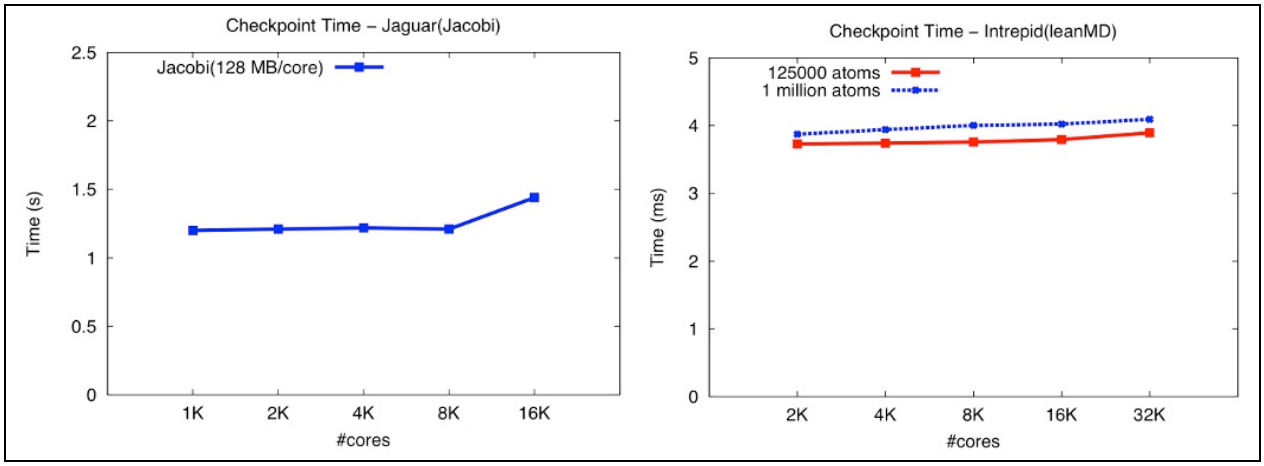


Figure 7. Time to checkpoint in different applications. Increasing the size of the system does not severely impact the ability to quickly store the checkpoint in local storage.

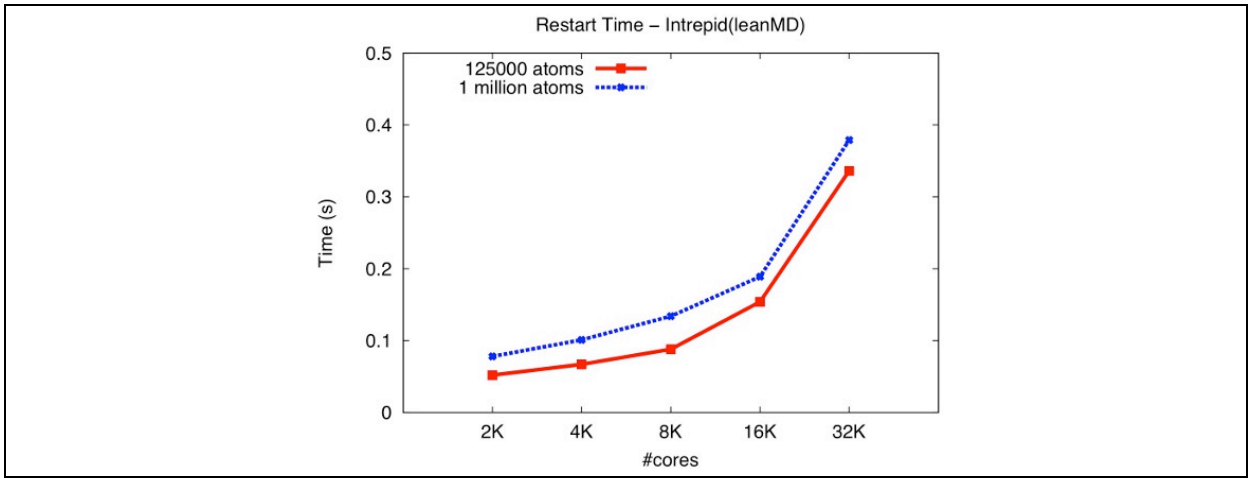


Figure 8. The time to restart after a failure is very short. However, the scale of the system poses a challenge.

On another front, we studied a novel mechanism to adapt our message logging scheme to executions where the communication load per node changes. Recently, we have been investigating another way in which message logging can be improved by allowing a load balancer to provide crucial information about the application. With this information, we vastly reduce the memory overhead associated with storing the messages. We call this approach Dynamic Team-based Message Logging. Nodes are divided into teams and only messages crossing team boundaries are logged. Teams are, however, dynamic. Depending on the runtime conditions, the load balancer may change the teams to reflect the change in the load of different nodes. Figure 9 shows the results of the Team Load Balancer (TeamLB) that attains two goals: i) provides a good load balance across the computation nodes and, ii) reduces the message logging overhead by grouping highly connected objects in the same team. Although this load balancer has a small overhead penalty, it drastically reduces the memory overhead of message logging.

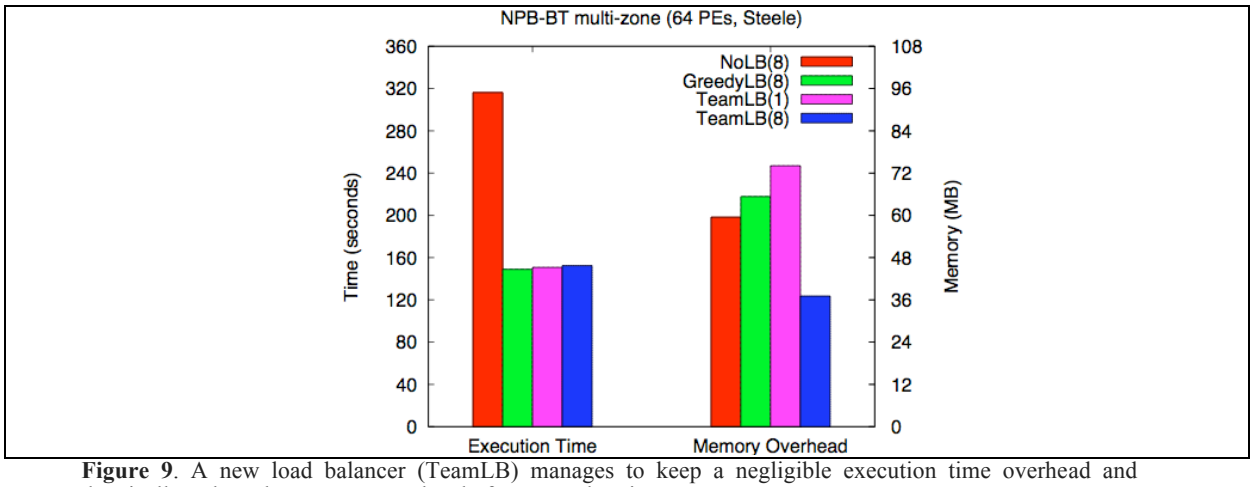


Figure 9. A new load balancer (TeamLB) manages to keep a negligible execution time overhead and drastically reduce the memory overhead of message logging.

Additionally, we have been exploring another way to scale our fault tolerance approaches. By using the SMP build of Charm++, where there is a single heavyweight process per node (and multiple threads), we managed to obtain a scheme that better matches the type of failures in big systems. By inspecting failure logs that are publicly available, we built a profile of the frequency of failures and the number of nodes in each failure. Our findings can be summarized in that the probability of more than one node failing concurrently is very low. Most of the time a failure will only include a single node.

We redesigned both approaches, checkpoint/restart and message logging, to address this characteristic of failures in supercomputers. Our implementations generated promising preliminary results, but we are in the process of extending them to include more experiments with more applications.

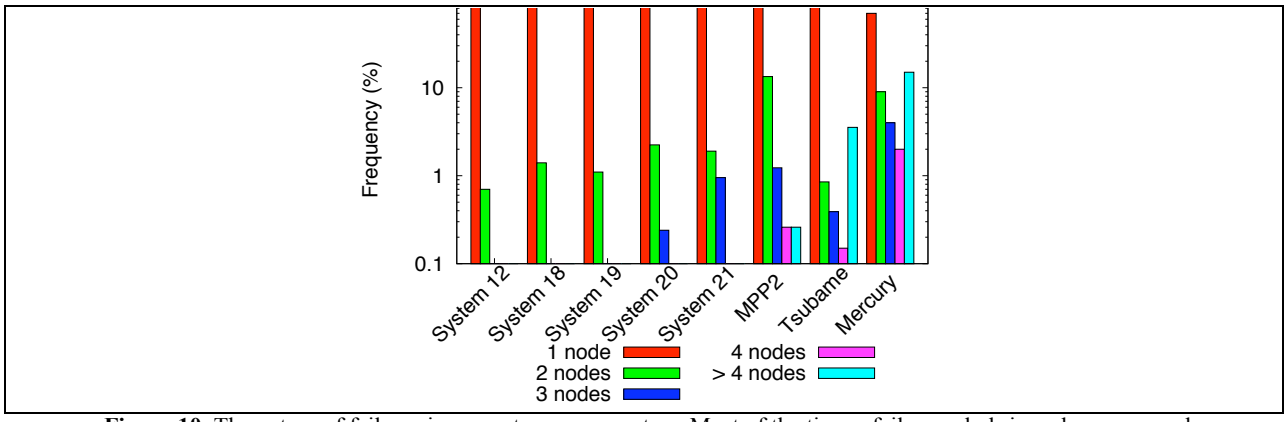


Figure 10. The nature of failures in current supercomputers. Most of the time a failure only brings down one node.

Also during this period, we continued our fault tolerance research by investigating a more advanced form of the message-logging scheme that we had studied initially. The new technique is based on a variant of the *causal* message logging protocol that seems to be a promising alternative to provide fault tolerance in large supercomputers. This study was conducted over three phases: first, we analyzed various scenarios that make pessimistic (i.e. conservative) message logging compromise the performance in order to keep consistency in an execution; next, we did a performance comparison of pessimistic and causal approaches for message logging with different applications; then we conducted a performance evaluation of the simple causal message logging protocol for applications that scale up to 1024 processors.

In contrast to pessimistic message logging, this new causal approach has low latency overhead, especially in collective communication operations. Besides, it reduces the number of messages when more than one thread is running per processor. In our tests, we demonstrated that a simple causal message logging protocol has a faster recovery and a low performance penalty when compared to checkpoint/restart.

A major source of performance penalization of most message-logging protocols is the use of determinants. These bits of information are necessary to provide a correct recovery from a failure. During normal execution, the message-logging protocol creates, stores and sends determinants. The combine cost of all those operations varies from application to application, but it may be as high as 20% in some situations. Therefore, a strategy that avoids determinants is desirable to keep message-logging as an alternative to provide fault tolerance in the future. A new strategy that avoids the use of determinants uses high-level information from the programming language. In some cases, it is possible to avoid the creation of determinants altogether, removing a high percentage of the performance penalization of message-logging.

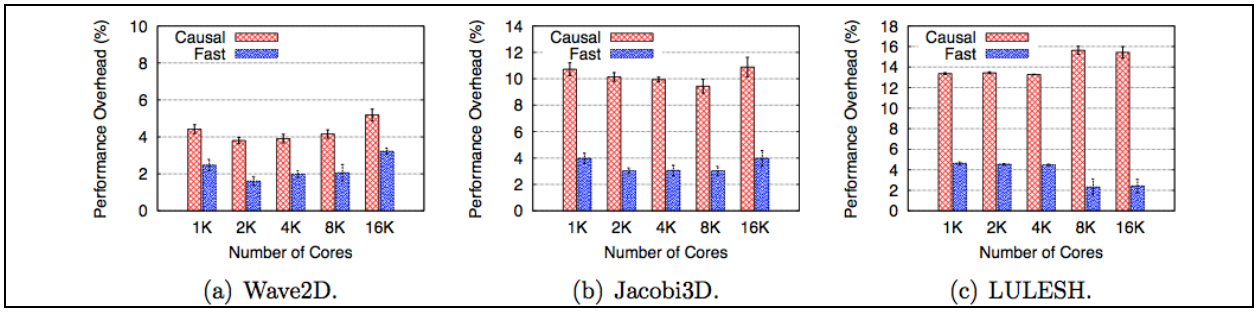


Figure 11: Causal versus Fast message-logging strategies for three applications.

The experiments run on Intrepid revealed that the new protocol reduces the performance overhead by a big margin, compared to a traditional message-logging protocol. A collection of three iterative stencil programs were used in the experiments (Jacobi3D, Wave2D and LULESH).

These programs differ in the amount of computation per iteration and the communication pattern. The optimized protocol reduced the performance overhead in these applications more than 50%, 66%, 75%, respectively, compared to a traditional message-logging protocol. After that reduction, the performance overhead of the protocol is lower than 4% for all the applications examined.

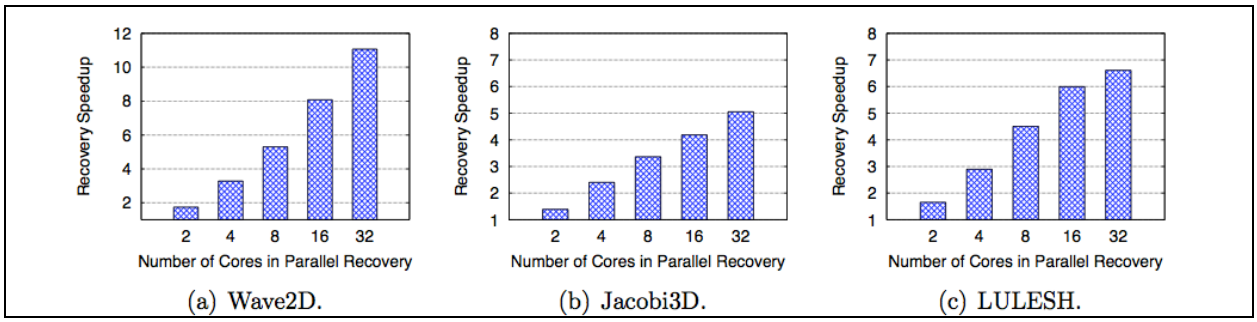


Figure 12: Recovery speedups made possible by parallel recovery

Additionally, this new protocol was extended with the parallel recovery strategy of Charm++ to accelerate recovery by a factor of 10, 5, and 6 in Jacobi3D, Wave2D, and LULESH, respectively. Overall, the new strategy has a low overhead and provides a competitive strategy to provide resilience at exascale.

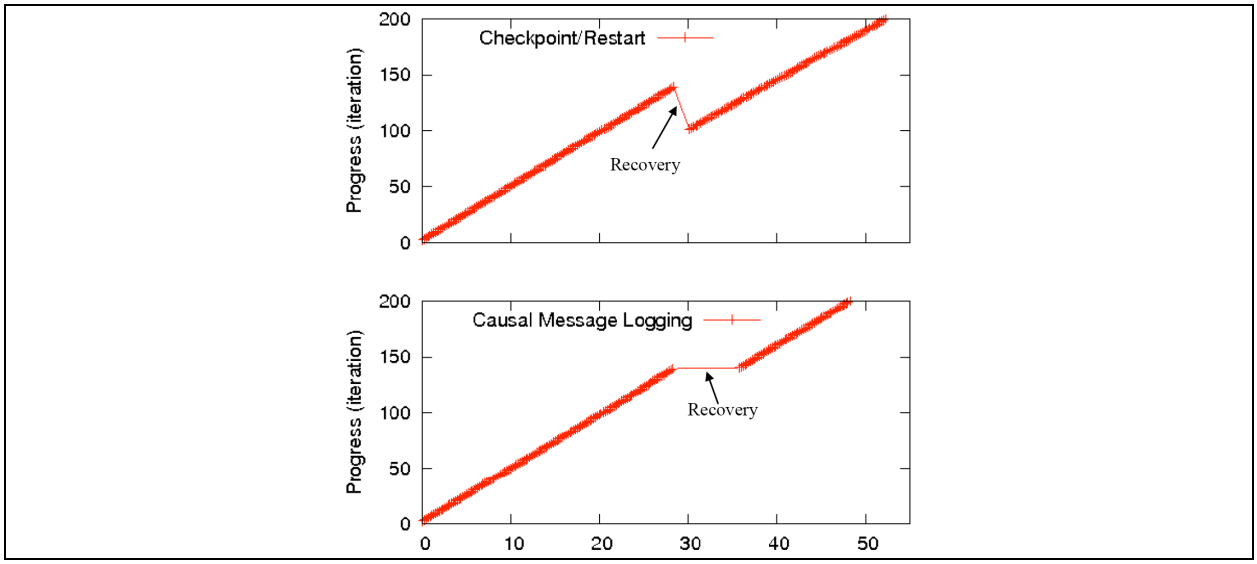


Figure 13. Effect of a failure on execution of a 7-point stencil.

To better evaluate the new approach when failures occur, we employed a 7-point stencil, and forced the recovery to happen after an external failure was introduced in the execution. The code executes 200 iterations, and we introduced a failure at iteration 140. Figure 13 shows the performance under the causal message logging protocol and under checkpoint-restart; a checkpoint was taken at iteration 100. The figure plots the application progress, in terms of completed iterations, as a function of elapsed time. In the checkpoint-restart case, the work of a few iterations (i.e. 100 to 140) needs to be redone when the failure occurs; meanwhile, with causal message-logging, only the failing processor requires its work to be repeated, and other processors that do not depend on it can proceed. Hence, the interruption is less severe, and the overall execution is allowed to complete faster than in the checkpoint-restart case. Notice also the significant energy savings of the causal message logging protocol over checkpoint-restart, as only a few processors are affected by the occurrence of the failure and its recovery.

In summary, our evaluations so far identified multiple performance problems of pessimistic message logging and showed that causal message logging has better performance and scalability for all the programs we ran in our experiments. Full results of these studies were reported in [Meneses2011]. There are, however, remaining challenges for causal message logging. Specifically, it imposes a higher latency on communication, which can be a problem for strong scaling and collective operations, and it requires a modest amount of additional memory to store determinants, when compared to executions without any fault tolerance provision. As we proceed in our research, we are addressing these issues and exploiting ways to alleviate them on large scale systems.

3.3 Scalable Load Balancing (Colony lead: Univ. of Illinois)

Meeting power requirements of the huge exascale machines of the future is one major challenge facing the HPC community. As power consumption and power costs rise, the bottom line impact is felt by everyone involved in parallel research. Members of the Parallel Programming Laboratory (PPL) are focusing on ways to minimize cooling power for these machines. We propose a technique that uses a combination of DVFS and temperature aware load balancing to constrain core temperatures as well as save cooling energy. Our scheme is specifically designed to suit parallel applications that are typically

tightly coupled. Currently, the temperature control comes at the cost of execution time and we are working to minimize the timing penalty. We have run experiments with three parallel applications, each with a different power utilization profiles, run on a 128-core (32-node) cluster with a dedicated air conditioning unit. As the experiment is running, we calibrate the efficacy of our scheme based on three metrics: ability to control average core temperatures thereby avoiding hot spot occurrence, timing penalty minimization, and cooling energy savings. Our preliminary results show cooling energy savings of up to 57% with timing penalty mostly in the range of 2 to 20%.

To demonstrate the effectiveness of our scheme, we use three applications having different utilization and power profiles. The first is a canonical benchmark, Jacobi2D, that uses 5 point stencil to average values in a 2D grid using 2D decomposition. The second application, Wave2D, uses a finite differencing scheme to calculate pressure information over a discretized 2D grid. The third application, Mol3D, is from molecular dynamics and is a real world application to simulate large bio-molecular systems. The experiment shows that we were able to reduce the timing penalty associated with DVFS by a great margin. Using our load balancing strategy, we were able to reduce the timing penalty to 27% for Jacobi2D. Other than that we also used performance counters in order to relate application characteristics to temperature control. Looking towards the future, we plan to take the DAG of the application into account in order to reduce the timing penalty even further. We are also looking for ways to save machine energy consumption based on the application characteristics.

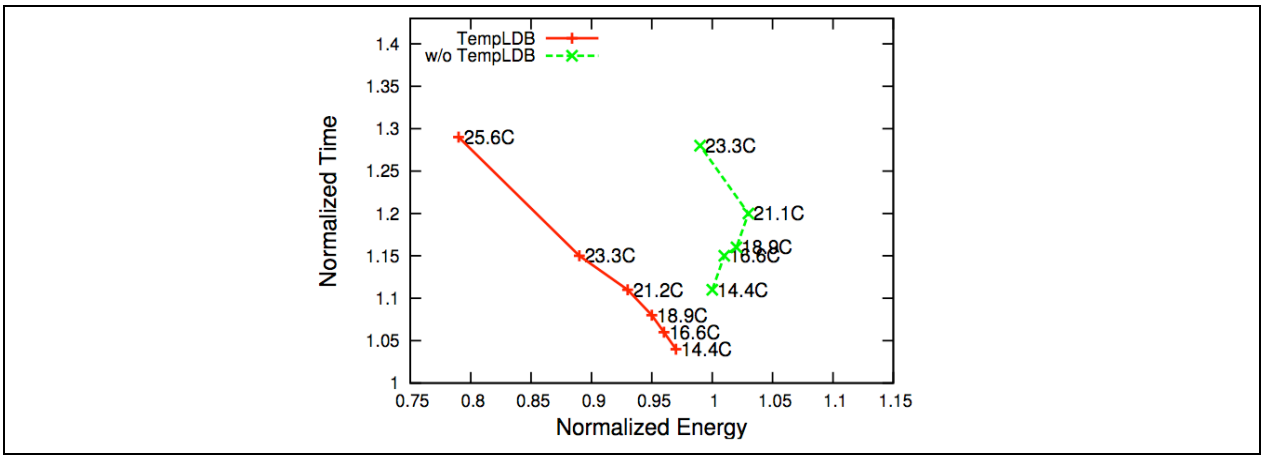


Figure 14. Reduction in energy consumption by controlling the temperature of different cores in the machine.

If the cost of load balancing is more than the benefit obtained from load balancing, it degrades the performance further. Meta-Balancer framework, implemented in Charm++, automatically identifies a load balancing period based on the application characteristics and cost of load balancing. Meta-Balancer has been previously shown to perform well up to 4096 cores on Jaguar. In this project we show the benefits of Meta-Balancer on up to 131,072 cores of Intrepid, BlueGene/P.

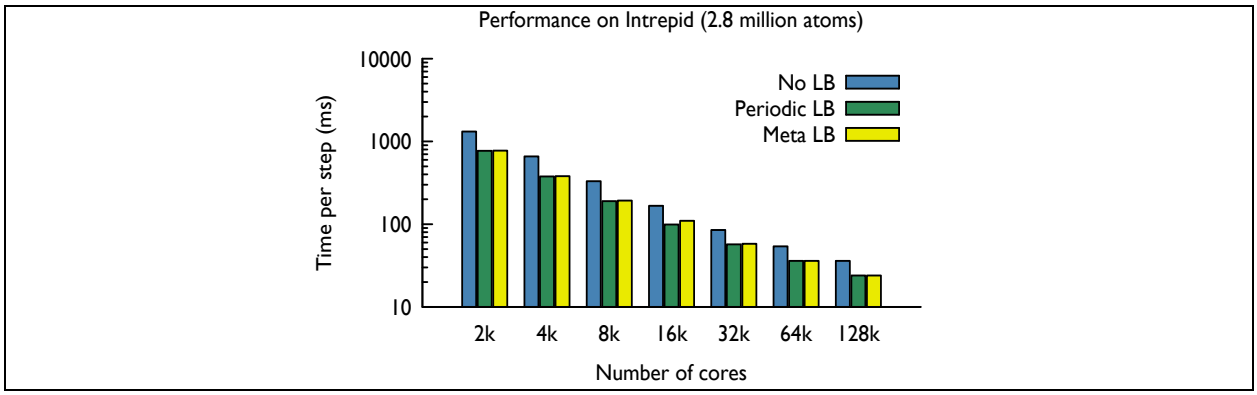


Figure 15: New Smart Runtime System Load Balancer

The performance of Meta-Balancer was evaluated on LeanMD benchmark and the results are plotted in Figure 1 above. LeanMD is a molecular dynamics simulation program written in Charm++. It simulates the behavior of atoms based on the Lennard-Jones potential. Load imbalance in LeanMD is due to the variation in number of atoms per cell as well as the slow migration of atoms. LeanMD was run for a system of 2.8 million atoms for 2000 iterations from 2048 to 131072 cores. The load balancing strategies used were GreedyLB for runs till 8k cores and HybridLB for larger runs. The HybridLB is a hierarchical strategy which uses GreedyLB and RefineLB strategies. We experimented with a range of hand tuned load balancing period. We find that if load balancing is done frequently, then the overhead of load balancing is more than the benefit but if load balancing is performed infrequently, then the performance is affected by load imbalance. Meta-Balancer is able to automatically identify optimal load balancing period without any input from the user. It is also able to scale to 131072 cores without any performance bottleneck.

HPC-Colony yielded significant research results on dynamic load-balancing techniques. First, we consolidated our studies of applying a hierarchical load-balancing scheme that we had developed in the previous year; those studies were reported in [Zheng2010]. Using this hierarchical load balancer more recently, combined with optimizations added to the SMP version of Charm++, we were able to scale the NAMD molecular simulator to the entire extent of Jaguar, a Cray XT5 at ORNL, running on 224,000 processors. Part of the obtained results, which we reported in [Mei2011], is shown in Figure 16, corresponding to NAMD's performance under different configurations on Jaguar with a 100 million-atom data-set. As shown, scaling is excellent for the no-cutoff case.

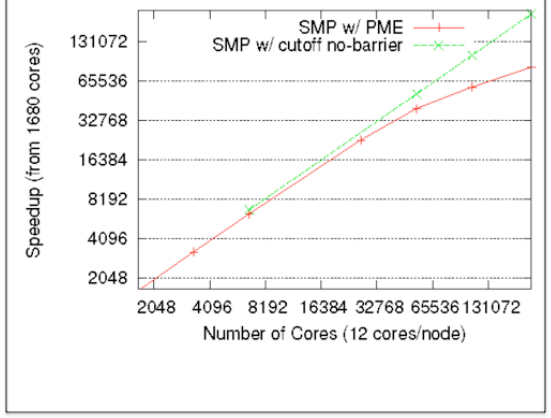


Figure 16: Scaling of NAMD on ORNL's Jaguar under different configurations

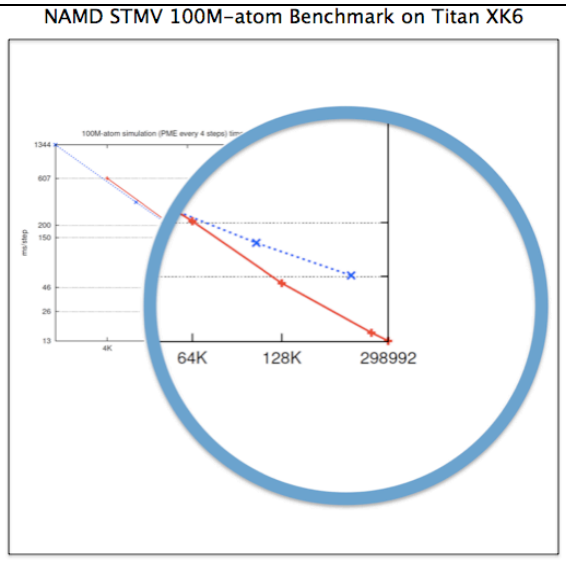


Figure 17: Further performance improvements achieved in NAMD scaling through Charm++ runtime optimizations. Full Titan machine performance improved from 26ms to 13ms.

More recently, further improvements were obtained through novel techniques exploiting new features in the Cray Gemini interconnect available with XK6 version of Titan. The Jaguar machine was upgraded from an XT5 system has 2.6 Ghz six-core AMD Opteron nodes (total 224,256 cores) to an XK6 system with 2.1Ghz sixteen-core interlagos nodes (total 298,992 cores) and GPUs. As portrayed in Figure 17 and reported in a pending paper, our Titan optimized runtime system was able Performance for a 100M atom NAMD run (PME every 4 steps) improved from a 26 milliseconds per step runtime over last year's MPI

over SeaStar+ numbers, to a 13 milliseconds per step runtime with the new software and hardware [Yanhua12].

The areas of weather and climate prediction pose a hard challenge for the efficient use of large systems. One of the major factors limiting performance of forecasting models in current machines is load imbalance. Besides the static causes of such imbalance, such as topography, there are dynamic factors that may affect a weather simulation, like the movement of clouds and of thunderstorms. Due to that imbalance, scalability of those models suffers when they are executed on a large number of processors.

We investigated the use of our Adaptive MPI (AMPI), an implementation of the MPI standard based on Charm++, on BRAMS, an existing production-level weather forecasting model. BRAMS is written in Fortran90 and uses MPI for parallelization. As an example, Figure 14 shows the result of a real BRAMS forecast and the corresponding load observed on the 64 processors executing that forecast. The color coding scheme represents rain intensity, in the forecast, and processor load, in the grid of processors. As one can see, there is a big and clear correlation between more rain and higher computational load.

We conducted several tests with BRAMS on Kraken, a Cray XT5 at ORNL. In those tests, we assessed the effects of virtualization and of load balancing on BRAMS executions. Our first observation was that simple AMPI virtualization already improved BRAMS performance. This was due to a combination of (a) better overlap between computation and communication, and (b) better cache utilization, since the over-decomposition of AMPI produces sub-domains that more naturally fit the sizes of the machine's caches (we measured such cache improvements and reported results in [Rodrigues2010]).

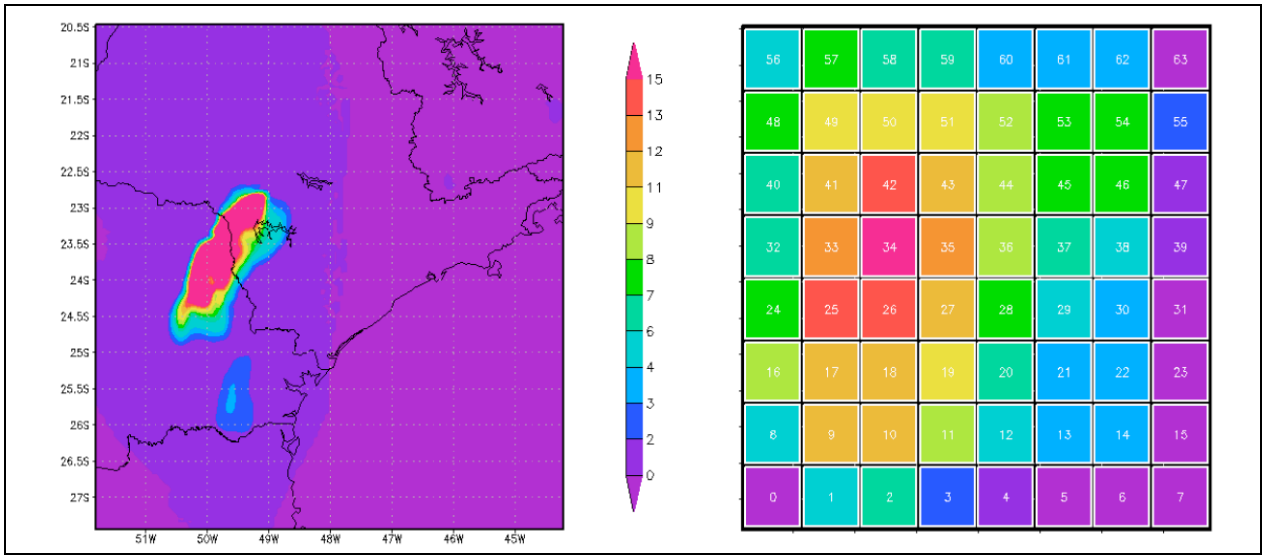


Figure 18. Results of a BRAMS weather forecasting and corresponding load on the 64 used processors.

Next, we applied several load balancers to BRAMS. Besides testing various load balancers already available in Charm++, we also developed a new balancer based on the distribution of sub-domains to processors according to a space-filling curve defined by to a Hilbert function. This distribution seems to be very appropriate for the two-dimensional domain decomposition employed by BRAMS, and preserves some of the locality of communication across sub-domains, even when some of those sub-domains migrate across processors due to load balancing. As a brief sample of our obtained results, Figure 19 shows the original processor utilization in BRAMS before any virtualization was applied, and the resulting utilization obtained with a virtualization factor of eight (i.e. AMPI divides each original domain into eight sub-domains) and the Hilbert load balancer. There is a much higher utilization, and we observed a reduction of more than 30% in the total execution time.

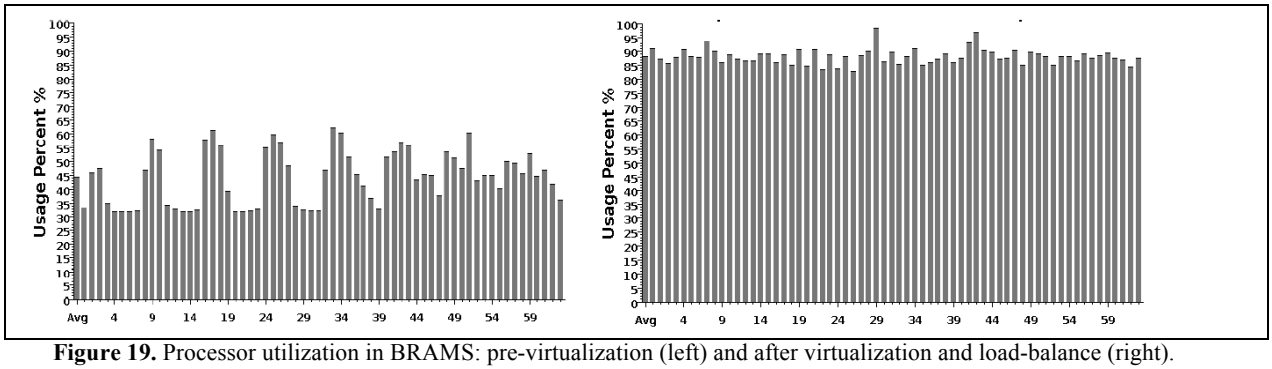


Figure 19. Processor utilization in BRAMS: pre-virtualization (left) and after virtualization and load-balance (right).

The more recent work in this area has focused on attempts to provide more automation to the entire optimization process via load balancing, such as finding automatically the best load balancing period, based on balancing costs and imbalance penalties. Our studies indicated that, for codes with a large memory footprint such as BRAMS, assessing the degree of imbalance is relatively cheap compared to actually migrating sub-domains across processors. Hence, one can exploit techniques that monitor the degree of imbalance closely, and only allow migrations that would produce performance gains higher than the penalties associated to the current imbalance.

The present grant also partly funded preliminary work on power-aware load-balancing techniques. It is now well known that increasing the number of cores and clock speeds on a smaller chip area implies more heat dissipation and an increased heat density. This increased heat, in turn, leads to higher cooling costs and the possible occurrence of hot spots. Effective use of dynamic voltage and frequency scaling (DVFS) can help to alleviate this problem. However, there is an associated execution time penalty, which can get amplified in parallel applications. In high performance computing, applications are typically tightly coupled and even a single overloaded core can adversely affect the execution time of the entire application. We have started to investigate a temperature-aware load-balancing scheme that uses DVFS to keep core temperatures below a user-defined threshold, with minimal timing penalties. While doing so, it also reduces the possibility of hot spots. We tested our scheme with three parallel applications having different energy consumption profiles.

Results from our initial experiments show that it is possible to save up to 14% in execution time and 12% in machine energy consumption as compared to frequency scaling without using load balancing. As an example, Figure 16 shows the measured execution time of a Jacobi-2D code on 128 processors, as a function of the temperature set for the machine room's air conditioner. When our temperature-aware load balancer (TempLDB) is used, the effects of a slowdown due to pure DFVS are not as strong, resulting in better overall performance. In other tests, we are also able to bound the average temperature of all the cores and reduce the temperature deviation amongst the cores by a factor of three. A full description of our initial results in this area was reported in [Sarood2011].

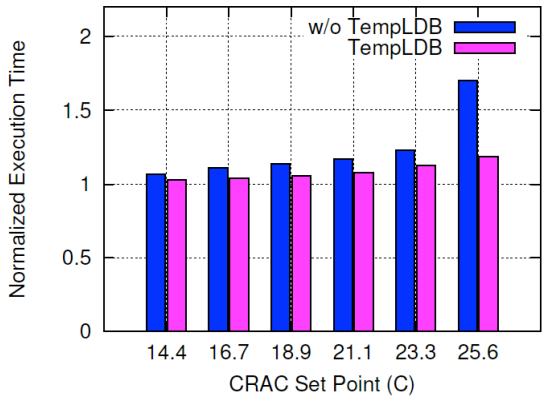


Figure 20. Effects of temperature-aware load balancing on Jacobi-2D execution with 128 cores under DVFS

3.4 Task Mapping (Colony lead: Univ. of Illinois)

The third focus of our adaptive runtime system work was the problem of task mapping on large parallel machines. Network contention has a significantly adverse effect on the performance of parallel applications with increasing size of parallel machines. Machines of the current petascale era are forcing application developers to map tasks intelligently to job partitions to achieve the best performance

possible. We have developed a framework for automated mapping of parallel applications with regular communication graphs to two and three dimensional mesh and torus networks. This framework can save much effort on the part of application developers to generate mappings for their individual applications.

One component of our framework is a process topology analyzer to find regular patterns and, when found, to determine the dimensions of the communication graphs of applications. The other component is a suite of heuristic techniques for mapping 2D object grids to 2D and 3D processor meshes. The framework chooses the best heuristic from the suite for a given object grid and processor mesh pair based on the *hop-bytes* metric. We obtained performance improvements using the framework, for a 2D Stencil benchmark in MPI and for the Weather Research and Forecasting model (WRF) running on the IBM Blue Gene/P.

For WRF, some of our results are shown in Figure 21; on 1,024 nodes, the average hops per byte reduced by 63% and the communication time (not shown in the figure) reduced by 11%. We measured an overall performance improvement of 17% for the application. At 4,096 nodes, there is a reduction in total execution time by 5%. Such performance improvements can be quite significant for the overall completion time of long running simulations. We also compared our algorithms with others discussed in the literature, as described with the full results of this study in [Bhatele2010].

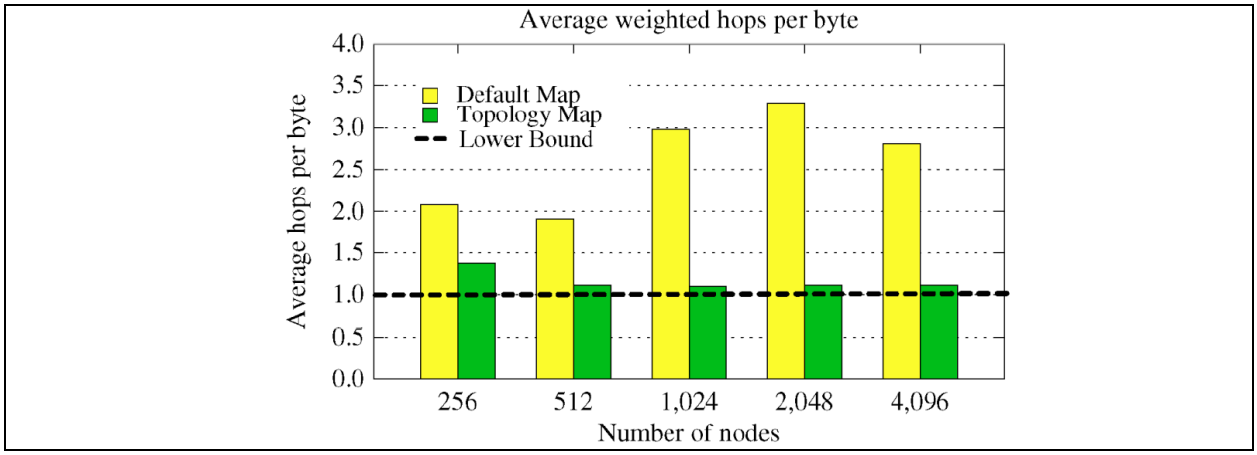


Figure 21. Results from topology-aware mapping of the WRF model on Blue Gene/P

3.5 Scalable membership, monitoring, & communication services (Colony lead: IBM Research)

Membership services enable the discovery of active groups of processes as well as failed nodes and processes, thus facilitating fault tolerant implementations [Renesse98, Ganesh03, Allavena05, Varma06]. Attribute replication services allow each node to declare runtime attributes on itself, which facilitates easy integration of cluster services and a distributed mechanism for service location and discovery. Monitoring services enable the collection and aggregation of statistics from nodes and processes thus supporting the implementation of dynamic load balancing schemes [Renesse03]. Group communication services provide groups of processes with the means to efficiently communicate using topic-based publish/subscribe, which greatly helps developing clustered applications [Chockler01, Eugster03, Chockler07]. Finally, a DHT (distributed hash table) provides services for storing and looking up key-value pairs in a distributed manner [Stoica01, Cass10].

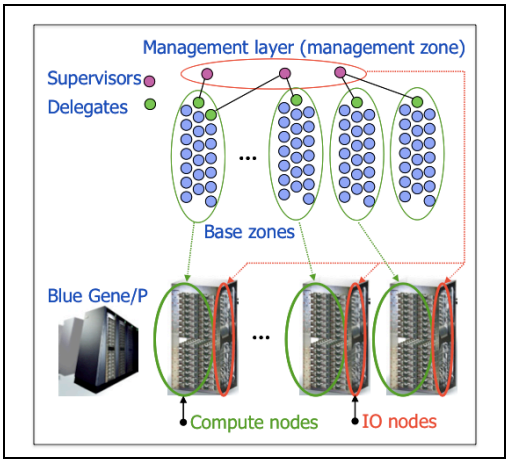


Figure 22: The SpiderCast hierarchical topology and its mappings to Blue Gene/P.

The hierarchical membership service we designed and implemented in SpiderCast is targeted to support a million nodes. This is the expected number of sockets in future large scale systems, and is more than enough to support current large scale systems. We employ a flexible two-layer hierarchical topology, comprised of base zones federated by a management zone, which forms a zone of its own. Each zone efficiently supports approximately 1000-2000 members (see Fig. 22). Our design aspires to quickly identify membership changes in a scalable environment with minimal overall system disruption, thus enabling efficient exascale size deployments. Furthermore, our design maps nicely to the Blue Gene/P architecture.

In order to test our implementation, we conducted large scale testing of the membership and hierarchical membership components on the Blue Gene/P platform in the IBM Watson Research Center. This set of experiments was presented in a poster at SC'11 [Tock11]. The tests emulated the workload of up to 2M nodes (as the Watson-based system has only 4 racks). We demonstrated that the relevant performance metrics of the membership service permit efficient support for 1M nodes, as planned.

Figure 23 presents the boot time of a single base zone as a function of the number of nodes. results indicate that a 2048 base zone boots in ~5.5 seconds. Figure 23 also presents the boot time of a hierarchical system with the full number of management nodes (management layer), connected to stub (small) base-zones. The stub base-zone simulate the same load exhibited by a full (2048) zone. The results indicated that a management zone with 1024 base-zones boots in ~9.7 seconds. Using these two measurements we can project the worse case boot time of a full system, shown in Figure 20. Results indicate that a 1M node system (512 base zones of 2048 nodes each) would have a stable view in ~10 seconds from boot.

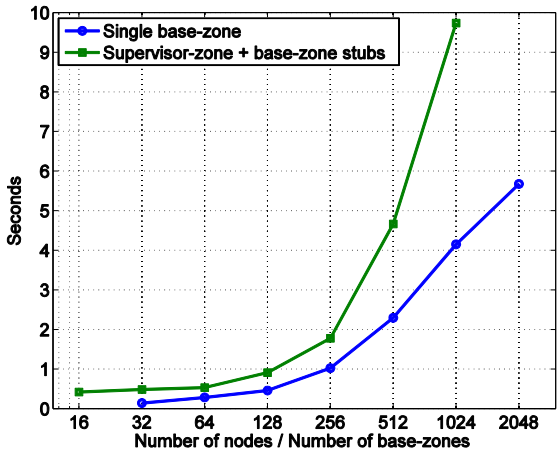


Figure 23. The boot time of a single base zone, as a function the number of nodes (blue); and the boot time of a hierarchical system with a variable number of stub base zones (green).

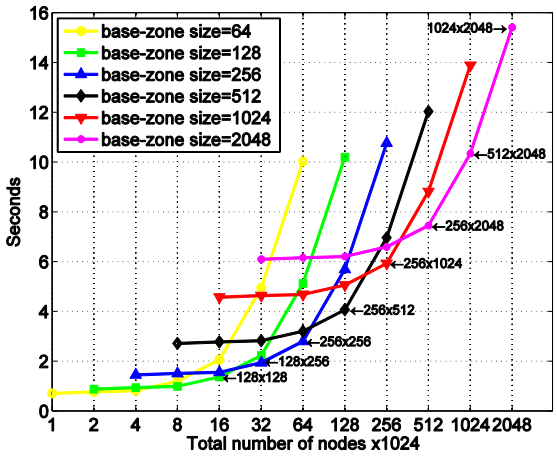


Figure 24. The projected boot time of a full system, as a function of total size, up to 2M nodes. Each curve represents a given base zone size. The arrows indicate the optimal setup in terms of management-zone and base-zone sizes.

Figure 24 presents the time it takes for a node to join and leave the overlay. Leave time include failure detection. The upper 3 curves present the join time of a variable number of concurrent processes. The middle 3 curves present the leave time of a variable number of concurrent processes. The bottom 3 curves present the leave time of a variable number of concurrent processes, as measured by a High Priority Monitor (HPM). These results indicate that a failure in a 1M node system would be detected and communicated to all nodes in about 800ms (400ms x2), and to the HPM in ~60ms (30ms x2).

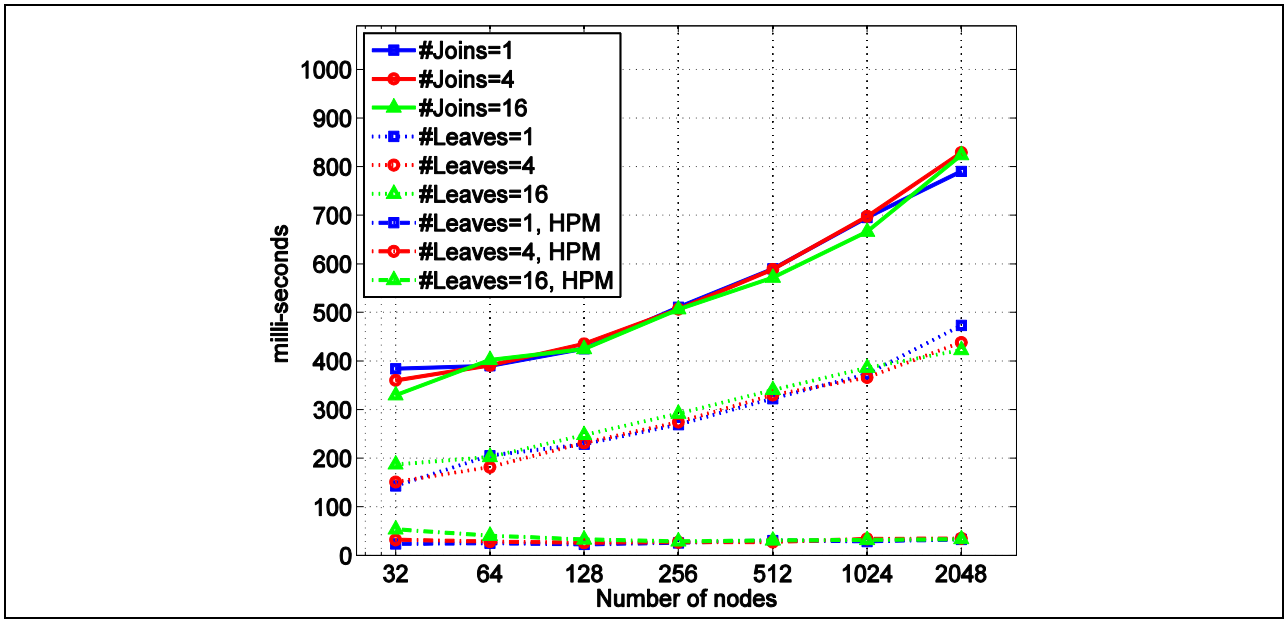


Figure 25. Join and leave time, as a function of zone size.

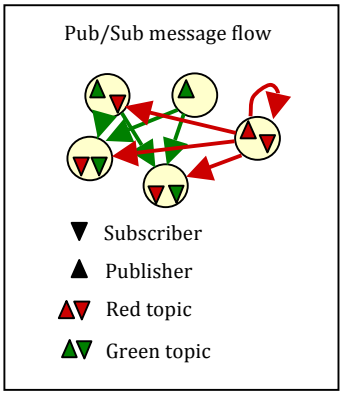
These results indicate that the discovery functions (Fig.23-24) and failure detection functions (Fig. 25) of the hierarchical membership service work according to plan. We are currently working towards a more detailed publication [Tock12].

Publish-Subscribe group communication

The light-weight topic-based publish/subscribe messaging service provides dynamic groups of processes with the means to communicate with each other in order to achieve a common goal. Example use cases are runtime control over processes groups; load balancing forward path; replication, and so on.

In order to implement the pub/sub service we implemented a randomized structured overlay topology, based on the Symphony protocol [Manku03]. This topology provides an overlay which supports O(LogN) routing from every node to very node like Chord [Stoica01], but has a simpler, more robust protocol, with better resistance to node churn.

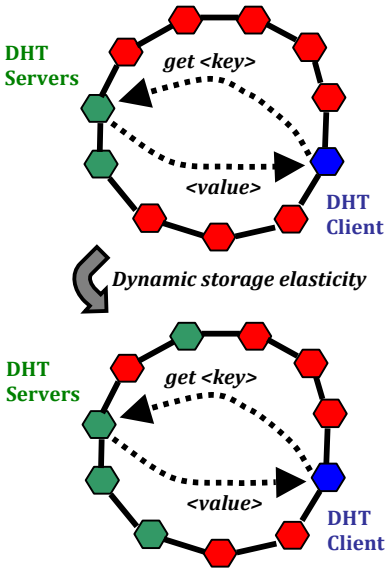
On top of the structured overlay topology we implemented two routing protocols. First is an efficient O(N) broadcast algorithm based on [ElAnsary03]; and second is a novel pub/sub routing algorithm that we developed. Our novel algorithm routes messages to nodes interested in a certain topic, while minimizing the number of non-interested nodes that are required to perform routing.



A building block in the pub/sub routing scheme is "interest aware membership", a mechanism that maintains the topic subscriptions on each node. Interest aware membership was implemented on top of the attribute service [TR1]. The last tier implements the pub/sub API and provides end-to-end quality of service and reliability.

Distributed hash table (DHT) implementation

In the current funding period we implemented a simplified form of a Distributed Hash Table (DHT), a key-value in-memory storage system. Our Pub/Sub design [TR2] calls for an internal use of the DHT, in order to facilitate the maintenance of large scale topic membership. We implemented a one-hop DHT in each zone that supports the full scale of get/put/delete variations with versioning. Our implementation does not currently provide replication, since in our internal use we do not need such s service. The DHT API is also exposed to the users of each zone.



The DHT implementation allows each node in a zone to be defined as a DHT server (that stores data), a DHT client (that operates on data), or both. This allows the user to dynamically change the amount of in-memory storage allocated in each zone. The DHT protocol supports the orderly addition and removal of DHT servers without the loss of data.

4. Funds / Costs Review

ORNL: Burn rates proceeded as planned; ORNL continued their involvement during the UIUC no cost extension.

IBM: Burn rates proceeded as planned; IBM continued their involvement during the UIUC no cost extension.

UIUC: UIUC requested and received approval for a no-cost extension. The Parallel Programming Laboratory (which is conducting the HPC Colony II research at UIUC) has been impacted by the recent Blue Waters announcement at UIUC/NCSA. The extension allows us to complete the planned Colony work and assume new work associated with Blue Waters with available staffing. While the UIUC work is being extended, there is benefit from Blue Waters in that an additional environment will be utilized for research with ideas developed under Colony.

5. Quantitative Impact & Achievements

The aforementioned loss of UIUC personnel to the Blue Waters project was the lone deviation to Colony II's plans. This resulted in a no-cost extension and ***all*** original investigation areas were explored. We maintained bi-weekly teleconferences with our project collaborators from UIUC and IBM, and had face-to-face team meetings at the Supercomputing conferences in November, and at the annual Charm++ Workshop in Urbana, IL, in April. Finally, a project website is maintained at http://www.hpc-colony.org

5.1 Awards

a. Winner of HPC Challenge Award at SC' 2011
b. IEEE Computer Society Sidney Fernbach Award: Laxmikant Kale

5.2 Selected Overall Project Highlights

a. ORNL is undertaking a post-project activity to pursue getting coordinated-scheduling advances adopted by HPC vendors. In addition, IBM is evaluating SpiderCast advances for possible HPC products.

b. ORNL is undertaking a post-project activity to pursue getting coordinated-scheduling advances adopted by HPC vendors. In addition, IBM is evaluating SpiderCast advances for possible HPC products.

c. The Parallel Programming Laboratory won the first place in the 2011 HPC Challenge contest at Supercomputing Conference for its parallel programming framework Charm++. The award recognizes Charm++ as the best performing system in the class 2 (productivity and performance) category of the contest.

b. Coordinated scheduling has produced compelling results. At 30,000 cores, coordinated scheduling achieved a 2.83x speedup over the normal Linux parallel-oblivious scheduling baseline. Moreover, coordinated scheduling was also able to outperform the alternate technique of core specialization (which reserves one core on each node to deal with OS interfering activities) while providing higher machine utilization. A bulk-synchronous-parallel benchmark improved 285% in execution time performance under the new kernel.

c. Recent developments in Charm++ fault tolerance infrastructure permits to run Charm++ programs on top of an MPI library and simulate rank failures. This mechanism exports a function to the user to kill a rank. Using this technique, new fault tolerance methods and algorithms can be developed on top of Charm++. The approach scales up to 32K cores on BG/P and provides an almost-negligible restart time.

d. A recent (April 2012) full-machine Jaguar test of a new Charm++ implementation provided impressive performance gains over an earlier version. The new version features a new network layer implementation designed for Cray's Gemini interconnect. Performance for a 100M atom NAMD run (PME every 4 steps) improved from a 26 milliseconds per step runtime over last year's MPI over SeaStar+ numbers, to a 13 milliseconds per step runtime with the new software and hardware.

e. We conducted research on scalable membership, attribute, monitoring and communication services that will enable sophisticated applications and general purpose cluster computing on high-performance computing systems with a very large numbers of processors. The SpiderCast system, that provides these services, is based on overlay and peer-to-peer technologies. SpiderCast will, on the one hand, utilize the unique architecture and networking features of Blue Gene/P [BGP08] to achieve top performance, and on the other hand, will develop broad scalable technologies for systems with hundreds of thousands of processors, which can be deployed on general cluster systems. Large scale experiments were conducted on the Blue Gene/P platform in the IBM Watson Research Center.

f. Developed a new power aware load balancing strategy which has shown improvements for both execution time and power consumption. The new scheme takes advantage of dynamic voltage and frequency scaling (DVFS) hardware capabilities.

g. We completed the initial implementation of a multi-zone scalable membership service as well as the low level design of the new Distributed Hash Table to be used for key-value pairs within SpiderCast.

h. Our new adaptive task mapping strategies show improvements for the Weather Research and Forecasting (WRF) model. For 1,024 nodes, the average hops per byte reduced by 63% and the communication time reduced by 11%..

i. Developed new causal-based message logging scheme with improved performance and scalability.

j. We also completed the design and implementation of a new dynamic load-balancing technique. Results for the BRAMS weather forecasting model show much higher machine utilization and reduction of more than 30% in execution time.

5.3 Publications

- Aaron Becker, Gengbin Zheng, and Laxmikant Kale. *Distributed Memory Load Balancing*. Encyclopedia of Parallel Computing
- Aaron Becker, Gengbin Zheng, and Laxmikant Kale. *Load Balancing, Distributed Memory*. Encyclopedia of Parallel Computing. 2011. Springer. New York, NY. pp 1043-1051.
- Abhinav Bhatele, Eric Bohm, Laxmikant V. Kale. *Optimizing communication for Charm++ applications by reducing network contention*. Concurrency and Computation: Practice and Experience, Volume 23, Issue 2. Feb. 2011. pp. 211-222.
- Abhinav Bhatele, Gagan Gupta, Laxmikant V. Kale and I-Hsin Chung. *Automated Mapping of Regular Communication Graphs on Mesh Interconnects*. In Proceedings of International Conference on High Performance Computing (HiPC), 2010. Dec. 2010. pp. 1-10
- Celso L. Mendes, Laxmikant V. Kale, Eduardo R.Rodrigues, Jairo Panetta. *Adaptive and Dynamic Load Balancing for Weather Forecasting Models*. In Annual meeting of the Cray Users Group (CUG) 2012. May 2012. Stuttgart, Germany.
- Chao Mei, Yanhua Sun, Gengbin Zheng, Eric J. Bohm, Laxmikant V. Kale, James C. Phillips and Chris Harrison. *Enabling and Scaling Biomolecular Simulations of 100 Million Atoms on Petascale Machines with a Multicore-optimized Message-driven Runtime*. Proceedings of the ACM/IEEE Supercomputing Conference 2011 (SC'11). Nov. 2011. Seattle, WA. pp. 1-11
- Danny Bickson, Ezra N. Hoch, Nir Naaman, Yoav Tock. *A Hybrid Multicast-Unicast Infrastructure for Efficient Publish-Subscribe in Enterprise Networks*. In SYSTOR 2010, The 3rd Annual Haifa Experimental Systems Conference. Haifa, Israel.
- Eduardo R. Rodrigues, Philippe O. A. Navaux, Jairo Panetta, Celso L. Mendes and Laxmikant V. Kale. *Optimizing an MPI Weather Forecasting Model via Processor Virtualization*. In Proceedings of International Conference on High Performance Computing (HiPC). Dec-2010
- Eduardo R. Rodrigues, Philippe O. A. Navaux, Jairo Panetta, Alvaro Fazenda, Celso L. Mendes and Laxmikant V. Kale. *A Comparative Analysis of Load Balancing Algorithms Applied to a Weather Forecast Model*. In Proceedings of 22nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). Oct. 2010. Itaipava, Brazil. pp. 71 - 78

- Esteban Menses. *Scalable Message-Logging Techniques for Effective Fault Tolerance in HPC Applications.* PhD diss., University of Illinois at Urbana-Champaign, 2013.
- Esteban Meneses, Celso L. Mendes and Laxmikant V. Kale. *Team-based Message Logging: Preliminary Results proceeding article*. In 3rd Workshop on Resiliency in High Performance Computing (Resilience) in Clusters, Clouds, and Grids (CCGRID 2010). May-2010. Melbourne, Victoria, Australia. pp. 697-702
- Esteban Meneses, Greg Bronevetsky and Laxmikant V. Kale. *Dynamic Load Balance for Optimized Message Logging in Fault Tolerant HPC Applications.* In International Conference on Cluster Computing (Cluster10). Sep-2010. Austin, TX. pp. 281-289
- Esteban Meneses, Greg Bronevetsky and Laxmikant V. Kale. *Evaluation of Simple Causal Message Logging for Large-Scale Fault Tolerant HPC Systems*. 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2011). May-2011. Anchorage, AK. pp. 1533-1540
- Esteban Meneses, Osman Sarood, and Laxmikant Kale. Accessing Energy Efficiency of Fault Tolerance Protocols for HPC Systems. In Proceedings of 24nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). Oct. 2012. New York City, NY.
- Laxmikant Kale. Esteban Meneses, Xiang Ni and L. V. Kale. *A Message-Logging Protocol for Multicore Systems*. In Proceedings of the 2nd Workshop on Fault-Tolerance for HPC at Extreme Scale (FTXS 2012). Jun-2012. Boston, MA.
- Esteban Meneses, Xiang Ni and Laxmikant V. Kale. *Design and Analysis of a Message Logging Protocol for Fault Tolerant Multicore Systems*. UIUC Technical Report PPL-11-30. Jul-2010. Champaign, IL.
- Gengbin Zheng, Abhinav Bhatele, Esteban Meneses and Laxmikant V. Kale. *Periodic Hierarchical Load Balancing for Large Supercomputers*. In International Journal of High Performance Computing Applications (IJHPCA). Nov-2011. pp. 371-385
- Gengbin Zheng, Esteban Meneses, Abhinav Bhatele and Laxmikant V. Kale. *Hierarchical Load Balancing for Charm++ Applications on Large Supercomputers.* In Proceedings of the Third International Workshop on Parallel Programming Models and Systems Software for High-End Computing (P2S2). Sept. 2010. San Diego, CA. pp. 436-444
- Gengbin Zheng, Abhinav Bhatele, Esteban Meneses and Laxmikant V. Kale. *Periodic Hierarchical Load Balancing for Large Supercomputers.* International Journal for High Performance Computing Applications (IJHPCA).
- Gengbin Zheng, Xiang Ni and Laxmikant V. Kale. *A Scalable Double In-memory Checkpoint and Restart Scheme towards Exascale.* In Proceedings of the 2nd Workshop on Fault-Tolerance for HPC at Extreme Scale (FTXS 2012). June-2012. Boston, MA.
- Gregory Chockler, Sarunas Girdzijauskas, Roie Melamed, Yoav Tock, Ymir Vigfusson. *Magnet: Practical Subscription Clustering for Internet-Scale Publish/Subscribe.* In DEBS 2010, 4th ACM International Conference on Distributed Event Based Systems.
- Harshitha Menon, Bilge Acun, Simon Garcia, Osman Sarood, and Laxmikant Kale. *Thermal Aware Automated Load Balancing for HPC Applications.* In Cluster Computing (CLUSTER), 2013 IEEE International Conference on. IEEE, 2013.
- Harshitha Menon, and Laxmikant Kale. *A Distributed and Dynamic Load Balancer for Iterative Applications.* In Proceedings of 2013 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'13). Nov. 2013. Denver, CO.

- Harshitha Menon, Nikhil Jan, Gengbin Zheng, and Laxmikant Kale. *Automated Load Balancing Invocation based on Application Characteristics.* In Cluster Computing (CLUSTER), 2012 IEEE International Conference on. IEEE, 2012.
- Jonathan Lifflander, Phil Miller, Ramprasad Venkataraman, Anshu Arya, Terry Jones and Laxmikant Kale. *Exploring Partial Synchrony in an Asynchronous Environment Using Dense {LU}*. UIUC Technical Report PPL-11-34. Aug-2010. Champaign, IL.
- Jonathan Lifflander, Phil Miller, Ramprasad Venkataraman, Anshu Arya, Terry Jones and Laxmikant Kale. *Mapping Dense LU Factorization on Multicore Supercomputer Nodes.* In Proceedings of 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS). May-2012. Shanghai, China.
- Laxmikant Kale. *Charm++*. Encyclopedia of Parallel Computing. 2011. Springer. New York,NY. pp. 256-264
- Li Yu, Ziming Zheng, Zhiling Lan, Terry Jones, Jim Brandt, and Ann Gentile. *Filtering log data: finding the needles in the haystack.* In 42nd IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012). Jun-2012. Boston, MA.
- Osman Sarood and Abishek Gupta and Laxmikant V. Kale. *Temperature Aware Load Balancing for Parallel Applications: Preliminary Work.* Proceedings of High Performance Power Aware Computing (HPPAC), in IEEE International Parallel and Distributed Processing Symposium 2011. May-2011. pp. 796 - 803
- Osman Sarood, Esteban Meneses and Laxmikant V. Kale. *A 'Cool' Way of Improving the Reliability of HPC Machines.* In Proceedings of 2013 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'13). Nov. 2013. Denver, CO.
- Osman Sarood and Laxmikant V. Kale. *A 'Cool' Load Balancer for Parallel Applications.* In Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11). Nov. 2011. Seattle, WA.
- Terry Jones. *Linux Kernel Co-Scheduling and Bulk Synchronous Parallelism.* International Journal of High Performance Computing Applications (IJHPCA). Vol. 26, Issue 2, May 2012.
- Terry Jones. *Linux Kernel Co-Scheduling For Bulk Synchronous Parallel Applications.* In 1st International Workshop on Runtime and Operating Systems for Supercomputers (ROSS 2011). May-2011. Tucson, AZ.
- Terry Jones, Andrew Tauferner, and Todd Inglett. *Linux OS Jitter Measurements at Large Node Counts using a BlueGene/L* technical report ORNL-TM2009-303. Jan-2010. Oak Ridge, TN.
- Terry Jones and Gregory Koenig. *A Clock Synchronization Strategy for Minimizing Clock Variance at Runtime in High-end Computing Environments.* In Proceedings of 22nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). Itaipava, Brazil.
- Terry Jones and Gregory Koenig. *Clock Synchronization in High-end Computing Environments: A Strategy for Minimizing Clock Variance at Runtime*. Concurrency and Computation: Practice and Experience. Journal of Concurrency and Computation: Practice and Experience (J CCPE), Vol. 25, Issue 6, DOI: 10.1002/cpe.2868, pages 881-897, April 25, 2013.
- Terry Jones and Gregory Koenig. *Providing Runtime Clock Synchronization With Minimal Node-to-Node Time Deviation on XT4s and XT5s*. CUG 2011. May-2011. Fairbanks, AK.

- Terry Jones, Michael Kirby, Joshua Ladd, David Dreisigmeyer, and Joshua Thompson. Accurate *Fault Prediction of BlueGene/P RAS Logs Via Geometric Reduction*. In 1st International Workshop on Fault-Tolerance for HPC at Extreme Scale (FTXS 2010). Jun-2010. Chicago, IL.
- Xiang Ni, Esteban Meneses, and Laxmikant Kale. *ACR: Automatic Checkpoint/Restart for Soft and Hard Error Protection.* In Proceedings of 2013 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'13). Nov. 2013. Denver, CO.
- Xiang Ni, Esteban Meneses, and Laxmikant Kale. *Hiding Checkpoint Overhead in HPC Applications with a Semi-Blocking Algorithm.* In Cluster Computing (CLUSTER), 2012 IEEE International Conference on, pp. 364-372. IEEE, 2012.
- Y. Tock, B. Mandler, J. Moreira, and T. Jones. *Poster: Scalable Infrastructure to Support Supercomputer Resiliency-Aware Applications and Load Balancing*. In companion to International Conference for High Performance Computing, Networking, Storage and Analysis (Poster, SC'11). Nov-2011. Seattle, WA.
- Yanhua Sun, Gengbin Zheng, Chao Mei, Eric J. Bohm, Laxmikant Kale, James C. Phillips, and Terry Jones. Optimizing Fine-grained Communication in a Biomolecular Simulation Application On Hybrid Cray XK6 System. In Proceedings of 2012 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'12). Nov. 2012. Salt Lake City, UT.
- Yanhua Sun, Gengbin Zheng, L. V. Kale, Terry R. Jones and Ryan Olson. *A uGNI-based Asynchronous Message-driven Runtime System for Cray Supercomputers with Gemini Interconnect*. In Proceedings of 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2012). May-2012. Shanghai, China.
- Ymir Vigfusson, Hussam Abu-Libdeh, Mahesh Balakrishnan, Ken Birman, Robert Burgess, Gregory Chockler, Haoyuan Li, and Yoav Tock. *Dr. Multicast: Rx for Data Center Communication Scalability*. EuroSys 2010.
- Yoav Tock and Benjamin Mandler. *SpiderCast: Distributed Membership and Messaging for HPC Platforms: An Architectural Overview and High Level Design*. IBM Technical Report IBM-IL-YT2010-1. Jan-2010. Haifa, Israel.
- Yoav Tock, Benjamin Mandler, and Gennady Laventman. *SpiderCast: Distributed Membership and Messaging for HPC Platforms: Publish-Subscribe and DHT Services High Level Design*. IBM Technical Report IBM-IL-YT2010-2. May-2010. Haifa, Israel.
- Yoav Tock, Benjamin Mandler, Jose Moreira, and Terry Jones. Design and Implementation of a Scalable Membership Service for Supercomputer Resiliency-Aware Applications. Lecture Notes in Computer Science Volume 8097, 2013, pp. 354-366. DOI http://10.1007/978-3-642-40047-6 37. Print ISBN 978-3-642-40046-9.
- Yoav Tock, Benjamin Mandler, Jose Moreira, and Terry Jones. Design and Implementation of a Scalable Membership Service for Supercomputer Resiliency-Aware Applications. Euro-Par 2013 Parallel Processing: 19th International Euro-Par Conference, Aachen, Germany, August 26-30, 2013: Proceedings. Springer, 2013.
- Ziming Zheng, Zhiling Lan, Li Yu and Terry Jones. *3-Dimensional Root Cause Diagnosis via Co-analysis*. In Proceedings of 9th International Conference on Autonomic Computing (ICAC). Sep-2012. San Jose, CA.

5.4 Talks

- Celso L. Mendes and Laxmikant V. Kale, "Adaptive MPI", Blue Waters PRAC Fall Workshop, Urbana, October 2010.
- Abhinav Bhatele, "Mapping your Application on Interconnect Topologies: Effort versus Benefits", George Michael HPC Fellow Presentation at Supercomputing'10, New Orleans, November 2010.
- Esteban Meneses, "Clustering Parallel Applications to Enhance Message-Logging Protocols", 4th Workshop INRIA-Illinois Joint Laboratory on Petascale Computing, Urbana, November 2010.
- Eric Bohm, "Scaling NAMD into the Petascale and Beyond", 4th Workshop INRIA-Illinois Joint Laboratory on Petascale Computing, Urbana, November 2010.
- Eric Bohm, Chao Mei, Yanhua Sun and Gengbin Zheng, "Charm++ Tutorial", Chinese Academy of Sciences, Beijing, China, December 2010.
- Abhinav Bhatele, "Topology Aware Mapping", University of Illinois (presented by telecom to the Chinese Academy of Sciences", December 2010.
- Laxmikant V. Kale, "State of Charm++", Charm++ Workshop, Urbana, April 2011.
- Osman Sarood, "Temperature-Aware Load Balancing for Parallel Applications", Charm++ Workshop, Urbana, April 2011.
- Abhinav Bhatele, "New Developments in the Charm++ Load Balancing Framework and its Applications", Charm++ Workshop, Urbana, April 2011.
- Esteban Meneses and Xiang Ni, "Fault Tolerance Support for Supercomputers with Multicore Nodes", Charm++ Workshop, Urbana, April 2011.
- Eric Bohm, "Charm++ Tutorial", Charm++ Workshop, Urbana, April 2011.
- Abhinav Bhatele, Gagan Gupta, Laxmikant V. Kale and I-Hsin Chung. *Automated Mapping of Regular Communication Graphs on Mesh Interconnects*. In Proceedings of International Conference on High Performance Computing (HiPC), 2010. Dec. 2010. pp. 1-10
- Celso L. Mendes, Laxmikant V. Kale, Eduardo R.Rodrigues, Jairo Panetta. *Adaptive and Dynamic Load Balancing for Weather Forecasting Models*. In Annual meeting of the Cray Users Group (CUG) 2012. May 2012. Stuttgart, Germany.
- Chao Mei, Yanhua Sun, Gengbin Zheng, Eric J. Bohm, Laxmikant V. Kale, James C. Phillips and Chris Harrison. *Enabling and Scaling Biomolecular Simulations of 100 Million Atoms on Petascale Machines with a Multicore-optimized Message-driven Runtime*. Proceedings of the ACM/IEEE Supercomputing Conference 2011 (SC'11). Nov. 2011. Seattle, WA. pp. 1-11
- Danny Bickson, Ezra N. Hoch, Nir Naaman, Yoav Tock. *A Hybrid Multicast-Unicast Infrastructure for Efficient Publish-Subscribe in Enterprise Networks*. In SYSTOR 2010, The 3rd Annual Haifa Experimental Systems Conference. Haifa, Israel.
- Eduardo R. Rodrigues, Philippe O. A. Navaux, Jairo Panetta, Celso L. Mendes and Laxmikant V. Kale. *Optimizing an MPI Weather Forecasting Model via Processor Virtualization*. In Proceedings of International Conference on High Performance Computing (HiPC). Dec-2010
- Eduardo R. Rodrigues, Philippe O. A. Navaux, Jairo Panetta, Alvaro Fazenda, Celso L. Mendes and Laxmikant V. Kale. *A Comparative Analysis of Load Balancing Algorithms Applied to a Weather Forecast Model*. In Proceedings of 22nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). Oct. 2010. Itaipava, Brazil. pp. 71 - 78

◊ Esteban Meneses, Celso L. Mendes and Laxmikant V. Kale. *Team-based Message Logging: Preliminary Results proceeding article*. In 3rd Workshop on Resiliency in High Performance Computing (Resilience) in Clusters, Clouds, and Grids (CCGRID 2010). May-2010. Melbourne, Victoria, Australia. pp. 697-702

◊ Esteban Meneses, Greg Bronevetsky and Laxmikant V. Kale. *Dynamic Load Balance for Optimized Message Logging in Fault Tolerant HPC Applications.* In International Conference on Cluster Computing (Cluster10). Sep-2010. Austin, TX. pp. 281-289

◊ Esteban Meneses, Greg Bronevetsky and Laxmikant V. Kale. *Evaluation of Simple Causal Message Logging for Large-Scale Fault Tolerant HPC Systems*. 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2011). May-2011. Anchorage, AK. pp. 1533-1540

◊ Esteban Meneses, Xiang Ni and L. V. Kale. *A Message-Logging Protocol for Multicore Systems*. In Proceedings of the 2nd Workshop on Fault-Tolerance for HPC at Extreme Scale (FTXS 2012). Jun-2012. Boston, MA.

◊ Gengbin Zheng, Abhinav Bhatele, Esteban Meneses and Laxmikant V. Kale. *Periodic Hierarchical Load Balancing for Large Supercomputers*. In International Journal of High Performance Computing Applications (IJHPCA). Nov-2011. pp. 371-385

◊ Gengbin Zheng, Esteban Meneses, Abhinav Bhatele and Laxmikant V. Kale. *Hierarchical Load Balancing for Charm++ Applications on Large Supercomputers.* In Proceedings of the Third International Workshop on Parallel Programming Models and Systems Software for High-End Computing (P2S2). Sept. 2010. San Diego, CA. pp. 436-444

◊ Gengbin Zheng, Abhinav Bhatele, Esteban Meneses and Laxmikant V. Kale. *Periodic Hierarchical Load Balancing for Large Supercomputers.* International Journal for High Performance Computing Applications (IJHPCA).

◊ Gengbin Zheng, Xiang Ni and Laxmikant V. Kale. *A Scalable Double In-memory Checkpoint and Restart Scheme towards Exascale.* In Proceedings of the 2nd Workshop on Fault-Tolerance for HPC at Extreme Scale (FTXS 2012). June-2012. Boston, MA.

◊ Gregory Chockler, Sarunas Girdzijauskas, Roie Melamed, Yoav Tock, Ymir Vigfusson. *Magnet: Practical Subscription Clustering for Internet-Scale Publish/Subscribe.* In DEBS 2010, 4th ACM International Conference on Distributed Event Based Systems.

◊ Jonathan Lifflander, Phil Miller, Ramprasad Venkataraman, Anshu Arya, Terry Jones and Laxmikant Kale. *Mapping Dense LU Factorization on Multicore Supercomputer Nodes.* In Proceedings of 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS). May-2012. Shanghai, China.

◊ Li Yu, Ziming Zheng, Zhiling Lan, Terry Jones, Jim Brandt, and Ann Gentile. *Filtering log data: finding the needles in the haystack.* In 42nd IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012). Jun-2012. Boston, MA.

◊ Osman Sarood and Abishek Gupta and Laxmikant V. Kale. *Temperature Aware Load Balancing for Parallel Applications: Preliminary Work.* Proceedings of High Performance Power Aware Computing (HPPAC), in IEEE International Parallel and Distributed Processing Symposium 2011. May-2011. pp. 796 - 803

◊ Osman Sarood and Laxmikant V. Kale. *A 'Cool' Load Balancer for Parallel Applications.* In Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11). Nov. 2011. Seattle, WA.

◊ Terry Jones. *Linux Kernel Co-Scheduling For Bulk Synchronous Parallel Applications.* In 1st International Workshop on Runtime and Operating Systems for Supercomputers (ROSS 2011). May-2011. Tucson, AZ.

◊ Terry Jones and Gregory Koenig. *A Clock Synchronization Strategy for Minimizing Clock Variance at Runtime in High-end Computing Environments.* In Proceedings of 22nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). Itaipava, Brazil.

◊ Terry Jones and Gregory Koenig. *Providing Runtime Clock Synchronization With Minimal Node-to-Node Time Deviation on XT4s and XT5s*. CUG 2011. May-2011. Fairbanks, AK.

◊ Terry Jones, Michael Kirby, Joshua Ladd, David Dreisigmeyer, and Joshua Thompson. Accurate *Fault Prediction of BlueGene/P RAS Logs Via Geometric Reduction*. In 1st International Workshop on Fault-Tolerance for HPC at Extreme Scale (FTXS 2010). Jun-2010. Chicago, IL.

◊ Yanhua Sun, Gengbin Zheng, L. V. Kale, Terry R. Jones and Ryan Olson. *A uGNI-based Asynchronous Message-driven Runtime System for Cray Supercomputers with Gemini Interconnect*. In Proceedings of 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2012). May-2012. Shanghai, China.

◊ Ymir Vigfusson, Hussam Abu-Libdeh, Mahesh Balakrishnan, Ken Birman, Robert Burgess, Gregory Chockler, Haoyuan Li, and Yoav Tock. *Dr. Multicast: Rx for Data Center Communication Scalability*. EuroSys 2010.

◊ Yoav Tock and Benjamin Mandler. *SpiderCast: Distributed Membership and Messaging for HPC Platforms: An Architectural Overview and High Level Design*. IBM Technical Report IBM-IL-YT2010-1. Jan-2010. Haifa, Israel.

◊ Yoav Tock, Benjamin Mandler, and Gennady Laventman. *SpiderCast: Distributed Membership and Messaging for HPC Platforms: Publish-Subscribe and DHT Services High Level Design*. IBM Technical Report IBM-IL-YT2010-2. May-2010. Haifa, Israel.

◊ Ziming Zheng, Zhiling Lan, Li Yu and Terry Jones. *3-Dimensional Root Cause Diagnosis via Co-analysis*. In Proceedings of 9th International Conference on Autonomic Computing (ICAC). Sep-2012. San Jose, CA.

5.5 Software Products

Software title	Current Version	Brief Description	Date of Last Release
Hierarchical load balancer module	Charm++ 6.5.0	plug-in	3/13/13
In-memory checkpoint-auto-restart module	Charm++ 6.5.0	enhanced features	3/13/13
Team based load balancer	Charm++ 6.5.0		3/13/13
Causal message-logging module	Charm++ 6.5.0	newly created	3/13/13
Hi-Precision Synchronized Global Clocks	OpenMPI 1.4.4	high precision global synchronized clocks	5/20/12
Parallel Coordinated Scheduling	Linux 2.6.32.59	gives Linux kernel parallel awareness for coordinated scheduling	10/1/13
Spider Cast	SpiderCastCPP 1.0	A C++ implementation of a scalable infrastructure that provides a membership service and group communication services for HPC environments.	5/1/12

Table 2: HPC Colony II Software Products

Notes:

- In addition to our on-team involvement with IBM, we are working with Cray to ensure our work on ORNL's Titan machine results in a commercially available technology. This work is being funded by ORNL and includes close involvement with both HPC vendors. A pathforward plan has been developed to release coordinated scheduling technology for future machines.
- Some parts of this research have been incorporated to the public distribution of the Charm++ software infrastructure, which is available in both source and binary formats. In particular, a new release of Charm++ (v.6.4.0) was made available recently, through the Charm++ download website: http://charm.cs.uiuc.edu/software/
- SpiderCast is currently identified by IBM as an internal asset. As such, it is a candidate for inclusion in some IBM products and/or continued development of advanced features.

6. Feedback, Recommendations, and Project Experiences

a) To Future Projects: Work closely with program manager to pave the way for allocations such as INCITE and ALCC.

b) To Future Projects: The process for acquiring a patent waiver for an industrial partner may require as much as 6 months. If industrial partners are potentially interested in pursuing a patent, any *legwork* involved in pursuing the patent is best started quite early.

c) To Future Projects: Highlights may be requested at any time. Maintain an ongoing activity to produce viewgraphs in the requested template.

d) To Headquarters: Any reduction in the time or effort involved in the approval-cycle for patent waivers would be helpful.

e) To headquarters: The earlier potential responders are made aware about upcoming FOAs, the better. A surprising amount of time is needed for team-building and deciding upon topic niches.

7. References

[Allavena05] A. Allavena, A. Demers, and J. E. Hopcroft, "Correctness of a gossip based membership protocol," in *PODC '05: Proceedings of the twenty-fourth annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*. New York, NY, USA: ACM Press, 2005, pp. 292-301.

[BGP08] IBM-Blue-Gene-Team, "Overview of the IBM Blue Gene/P project," *IBM Journal of Research and Development*, vol. 52, no. 1/2, pp. 199-220, 2008.

[Chockler01] G. Chockler, I. Keidar, and R. Vitenberg, "Group communication specifications: a comprehensive study," *ACM Computing Surveys*, vol. 33, no. 4, pp. 427-469, 2001.

[Cass10] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35-40, Apr. 2010.

[Chockler07] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, "Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication," in *DEBS '07: Proceedings of the 2007 inaugural international conference on Distributed event-based systems*. New York, NY, USA: ACM, 2007, pp. 14-25.

[Dusseau96] Andrea C. Dusseau, Remzi H. Arpaci, and David E. Culler. Efffective distributed scheduling on parallel workloads. In ACM SIGMETRICS '96 Conference on the Measurement and Modeling of Computer Systems, 1996.

[ElAnsary03] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi, "Efficient broadcast in structured P2P networks," in LNCS (The 2nd International Workshop On Peer-To-Peer Systems), F. Kaashoek and I. Stoica, Eds., vol. 2735, 2003, pp. 304-314.

[Eugster03] P. TH. Eugster, P. A. Felber, R. Guerraoui, and A. M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114-131, June 2003.

[Ferreira08] Kurt Ferreira, Ron Brightwell, Patrick Bridges**.** Characterizing Application Sensitivity to OS Interference Using Kernel-Level Noise Injection. *International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'08),* Austin, TX, November 2008.

[Ganesh03] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulie. Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, 52(2), February 2003.

[Hoefler10] T. Hoefler, T. Schneider, and A. Lumsdaine. Characterizing the Influence of System Noise on Large-Scale Applications by Simulation. In International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10*)*, Nov. 2010.

[Howland04] P. Howland and H. Park. Generalizing discriminant analysis using the generalized singular value decomposition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(8):995–1006, 2004.

[Jones03] Terry Jones, Shawn Dawson, Rob Neely, William Tuel, Larry Brenner, Jeff Fier, Robert Blackmore, Pat Caffrey, Brian Maskell, Paul Tomlinson, and Mark Roberts**,** Improving the Scalability of Parallel Jobs by adding Parallel Awareness to the Operating System**.** In International Conference for High Performance Computing, Networking, Storage and Analysis (SC'03*)*, Phoenix, AZ, November 2003.

[Lakshman10] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35-40, Apr. 2010.

[Lee99] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. Nature, 401(6755):788–791, 1999.

[Manku03] G. S. Manku, M. Bawa, and P. Raghavan, "Symphony: distributed hashing in a small world," in *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, ser. USITS'03, vol. 4. Berkeley, CA, USA: USENIX Association, 2003, pp. 10-23.

[MPI-Forum] MPI Forum. MPI: A Message-Passing Interface Standard. Version 2.2. September 4th 2009.

[Nataraj07] A. Nataraj, A. Morris, A. D. Malony, M. Sottile, and P. Beckman. The ghost in the machine: Observing the effects of kernel operation on parallel application performance. *In Proceedings of SC'07*, 2007.

[Oliker07] L. Oliker, A. Canning, J. Carter et al., "Scientific Application Performance on Candidate PetaScale Platforms," *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*:1-12, 2007.

[Renesse98] R. V. Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Proc. Middleware 98*, 1998.

[Renesse03] R. Van Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM Trans. Comput. Syst.*, vol. 21, no. 2, pp. 164-206, May 2003.

[Sottile04] M. Sottile and R. Minnich. Analysis of microbenchmarks for performance tuning of clusters. In *Proceedings of IEEE Cluster2004 International Conference on Cluster Computing*, pages 371–377, 2004.

[Stoica01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 149-160, October 2001.

[Tock10a] Yoav Tock, Benjamin Mandler, "SpiderCast: Distributed Membership and Messaging for HPC Platforms: An Architectural Overview and High Level Design". *Colony-II technical report*, January 2010.

[Tock10b] Yoav Tock, Benjamin Mandler, Gennady Laventman, "SpiderCast: Distributed Membership and Messaging for HPC Platforms: Publish-Subscribe and DHT Services High Level Design". *Colony-II technical report*, May 2010.

[Tock11] Y. Tock, B. Mandler, J. Moreira, T. Jones. Scalable Infrastructure to Support Supercomputer Resiliency-Aware Applications and Load Balancing. *SC'11 Poster, International Conference for High Performance Computing, Networking, Storage and Analysis*, November 2011.

[Tock12] Y. Tock, B. Mandler, J. Moreira, T. Jones. Scalable Infrastructure to Support Supercomputer Resiliency-Aware Applications and Load Balancing. HiPC'12, *Submitted.*

[Varma06] J. Varma, C. Wang, F. Mueller, C. Engelmann, and S. L. Scott, "Scalable, fault tolerant membership for mpi tasks on hpc systems," in *ICS '06: Proceedings of the 20th annual international conference on Supercomputing*. New York, NY, USA: ACM, 2006, pp. 219-228.