Experiences Parallelizing, Configuring, Monitoring, and Visualizing Applications for Clusters and Multi-Clusters

Otto Anshus, John Markus Bjørndalen, Lars Ailo Bongo
Department of Computer Science, University of Tromsø, Norway

To make it simpler to experiment with the impact different configurations can have on the performance of a parallel cluter application, we developed the PATHS system. The PATHS system use a "wrapper" to provide a level of indirection to the actual run-time location of data making the data available from wherever threads or processes are located. A wrapper specify where data is located, how to get there, and which protocols to use. Wrappers are also used to add or modify methods accessing data. Wrappers are specified dynamically. A "path" is comprised of one or more wrappers. Sections of a path can be shared among two or more paths.

By reconfiguring the LAM-MPI Allreduce operation we achieved a performance gain of 1.52, 1.79, and 1.98 on respectively two, four and eight-way clusters. We also measured the performance of the unmodified Allreduce operation when using two clusters interconnected by a WAN link with 30-50ms roundtrip latency. Configurations which resulted in multiple messages being sent across the WAN did not add any significant performance penalty to the unmodified Allreduce operation for packet sizes up to 4KB. For larger packet sizes the Allreduce operation rapidly detoriated performancewise.

To log and visualize the performance data we developed EventSpace, a configurable data collecting, management and observation system used for monitoring low-level synchronization and communication behavior of parallel applications on clusters and multi-clusters. *Event collectors* detect events, create *virtual events* by recording timestamped data about the events, and then store the virtual events to a *virtual event space*. *Event scopes* provide different *views* of the application, by combining and pre-processing the extracted virtual events. Online monitors are implemented as consumers using one or more event scopes. Event collectors, event scopes, and the virtual event space can be configured and mapped to the available resources to improve monitoring performance or reduce perturbation. Experiments demonstrate that a wind-tunnel application instrumented with event collectors, has insignificant slowdown due to data collection, and that monitors can reconfigure event scopes to trade-off between monitoring performance and perturbation. The visual views we generated allowed us to detect anomalous communication behavior, and detect load balance problems.

1. Introduction

A current trend in parallel and distributed computing is that compute- and I/O-intensive applications are increasingly run on cluster and multi-cluster architectures. As we add computing resources to a parallel application, one of the fundamental questions is how well the application scales. There are two main ways of scaling an application when processors are added: *speedup*, where the goal is to solve a problem faster, and *scaling up the problem*, where the goal is to solve a larger problem (or get a more fine-grained solution to a given problem) in a fixed time by adding computing resources (see also Amdahl [1] vs. Gustafson [13]).

As the complexity and problem size of parallel applications and the number of nodes in clusters increase, communication performance becomes increasingly important. Of eight scalable scientific applications investigated in [29], most would benefit from improvements to MPI's collective operations, and half would benefit from improvements in point-to-point message overhead and reduced latency.

Scaling an application when it is mapped onto different cluster and multi-cluster architectures involves controlling factors such as balancing the workload between the processes in the system, controlling inter-process communication latency, and managing interaction between the processes. In doing so, one of the main questions is understanding how an application is mapped to the given architecture. This requires an understanding of which computations are done where, where data is located, and when control and data flow through the system.

We show that the performance of collective operations improve by a factor of 1.98 by using better mappings

of computation and data to the clusters [4]. Point-to-point communication performance can also be improved by tuning configurations. In order to tune the performance of collective and point-to-point communication, fine-grained information about the applications communication events is needed to compute how the application behaves.

An example of this is the Message Passing Interface (MPI) [17] collective functions, which have scaling problems if the algorithms of the functions are not mapped properly to the cluster architecture. Understanding why the functions do not scale as well in some configurations as in others is difficult without an exact knowledge of how the functions are implemented and mapped to the cluster architecture.

Even if we discover the reason for the scaling problems, we may not be able to remedy the problem without modifying the source code of the communication library, as mechanisms intended to aid the mapping of the application to the cluster are either insufficient or not implemented [28].

In other cases, implementations of a communication layer or middleware may not have been tested on a cluster of the same size or configuration as the cluster an application is deployed at. This may expose new problems which also require intimate knowledge of the implementation to find and resolve [2].

2. PATHS

A middleware system or communication library usually provides the user with an API that abstracts away low-level details about how communication is implemented and where objects and shared data is located.

Although this simplifies programming distributed applications, it is difficult for the programmer to determine why some of the functionality provided by the API does not scale well, and where bottlenecks occur. Even if the user discovers a bottleneck or the reason for scaling problems, which may require intimate understanding of the API implementation [2], it may be difficult or impossible to solve the problems without modifying the implementation.

PATHS allows configuring and mapping of an applications *communication paths* to the available resources. PATHS use the concept of *wrappers* to add code along the communication paths, allowing for various kinds of processing of the data along the paths. PATHS use the PastSet [30] distributed shared memory system. In PastSet *tuples* are read from and written to named *elements*.

The PATHS system provides a method of specifying how the application is mapped onto clusters, focusing on the location of computations and data. The specification can be changed by modifying meta-data and meta-code, allowing an applications mapping to be studied, tuned, and remapped to a given cluster without recompiling the application code.

PATHS allows the user to specify what type of instrumentation should be used where. The user can also add new types of instrumentation to the system.

PATHS simplifies studying how different configurations influence the performance of an application when it is mapped onto a cluster or multi-cluster. This simplifies experimenting with multiple factors and configurations.

3. Experiences mapping applications to different clusters and multi-clusters

We have experimented with the reconfiguration of several benchmarks, including global reduction, Monte Carlo Pi, a wind tunnel, video distribution, and the ELCIRC river simulation of the Columbia River. The benchmarks were typicaly mapped onto three different clusters, consisting of 2-, 4- and 8-way SMP nodes. We also ran a multi-cluster configuration where the PATHS system was used to bind the three clusters together, as the nodes in each of the clusters didn't have direct connectivity to nodes in the other clusters. Space limitations prevent us from describing the experiments and their results here in any detail.

Global Reduction and Monte Carlo Pi: We did experiments with two benchmarks: the global reduction benchmark (a benchmark of PastSet's equivalent of MPI Allreduce), and the Monte Carlo Pi benchmark (an emberassingly parallel benchmark).

Experimenting with multiple factors and configurations can help expose the factors that are most important in a particular cluster, and which combination of factors lead to performance bottlenecks that should be avoided. An example is [5], where the sum wrapper was found to scale badly and become a performance bottleneck once it was used by more than 3 or 4 threads. This resulted in more time spent in the sum wrapper than sending messages between the 8-way nodes.

Performance can be improved significantly without removing or modifying components. Instead of rewriting

the implementation of the sum operations, the sum-operators were arranged hierarchically, to allow groups of 3-4 threads to compute a partial sum which was then forwarded down to a sum-operator further down in the hierarchy. This improved the performance of computing partial sums from threads on a node significantly.

For some configurations, sending more messages on the network than the minimum required may improve the performance. One of the reasons for this is that there is sometimes a tradeoff between the number of messages sent and the parallelism in handling these messages. An example is [5], where sending more messages than the minimum required to implement a global reduction sum reduced the latency by nearly a factor 2 (on the 2W cluster) compared to sending the minimum number of messages.

Different clusters may need different configuration strategies. The results in [5] shows that a strategy which performed best in one cluster was not the best strategy for another cluster.

The *Wind tunnel* application is a Lattice Gas Automaton particle simulation. Details of our experiments can be found in [3] and [8, 9]). The application has a linear speedup for each of the 3 clusters, and for a multi-cluster configuration using both Tromsø clusters. Combining the 2W cluster, which is located in Odense, Denmark, with any of the Tromsø clusters gave us sub-linear speedups. We tried a number of experiments and located some of the factors contributing to the bottlenecks, but have yet to identify them all.

Video distribution By careful configuration we managed to support 2016 clients without dropping frames [3].

LAM-MPI Configuration: For efficient support of synchronization and communication in parallel systems, these systems require fast collective communication support from the underlying communication subsystem. One approach is defined by the Message Passing Interface (MPI) Standard [17]. Among the set of collective communication operations broadcast is fundamental and is used in several other operations such as barrier synchronization and reduction [19]. Thus, it is advantageous to reduce the latency of broadcast operations on these systems.

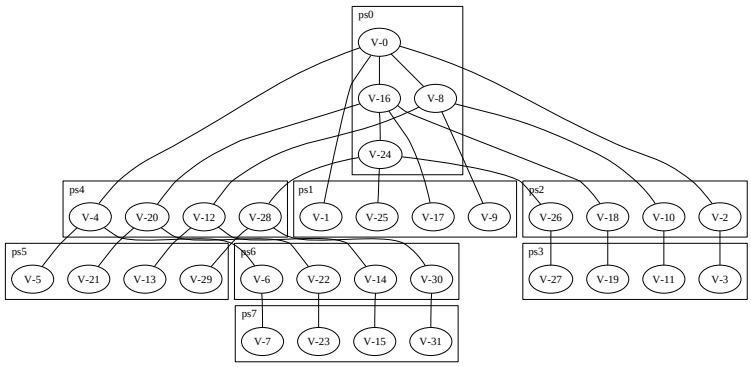


Figure 1: Log-reduce tree for 32 processes mapped onto 8 nodes.

In one of the experiments we did [31], we used PATHS to improve the execution times of the collective "AllReduce" operation in the PastSet tuple space system. To get a baseline, we compared this performance with the equivalent operation performance in LAM-MPI (Allreduce). We found that we could configure the PastSet Allreduce to be 1.83 times faster than LAM-MPI [14, 24]. The reason for this is that MPI uses static trees to reduce and broadcast the values. If these trees are not well matched to the cluster topologies, the performance will suffer as can be seen in figure 1 where many messages are sent across nodes in the system. The broadcast operation has a better mapping for this cluster though.

By adding an experimental configuration system to LAM-MPI we were able to first replicate the performance of LAM-MPI, showing that adding our configuration system did not impact the performance of LAM-MPI when running identical configurations. Then, experiments were conducted to try to find configurations which improved the performance compared to the original mappings [4]. Simple changes to the configuration resulted in a performance matching the PastSet Allreduce.

4. Monitoring

In this section we describe EventSpace [9, 7], an approach and a system for online monitoring the communication behavior of parallel applications.

For low-level performance analysis [8, 27] and prediction [32, 11], large amounts of data may be needed. For some purposes the data must be consumed at a high rate [25]. When the data is used to improve the performance of an application at run-time, low perturbation is important [22, 26]. To meet the needs of different monitors the system should be flexible, and extensible. Also the sample rate, latency, perturbation,

and resource usage should be configurable [22, 11, 27]. Finally, the complexity of specifying the properties of such a system must be handled.

The EventSpace system is designed to scale with regards to the number of nodes monitored, the amount of data collected, the data observing rate, and the introduced perturbation. Complexity is handled by separating instrumentation, configuration, data collection, data storage, and data observing.

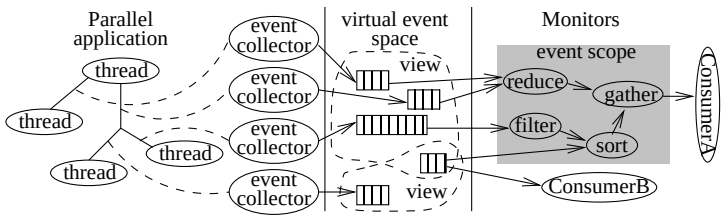


Figure 2: EventSpace overview.

Our approach, illustrated in figure 2, is to have a *virtual event space*, that contains traces of an applications communication (including communication used for synchronization). *Event scopes* are used by consumers to extract and combine virtual events from the virtual event space, providing different *views* of an applications behavior. The output from event scopes can be used in applications and tools for adaptive applications, resource performance predictors, and run-time performance analysis and visualization tools. When triggered by communication events, *event collectors* create a virtual event, and store it in the virtual event space. A virtual event comprises timestamped data about the event.

In EventSpace, an application is instrumented when the configurable communication paths are specified. Event collectors are implemented as PATHS wrappers integrated in the communication system. They are triggered by PastSet operations invoked through the wrapper. The virtual event space is implemented by using PastSet. There is one *trace element* per event collector.

We have also experimented with 3D visualizations in VRML. Initial experiments suggest that this may be a useful method of visualizing an application's behaviour. Extensive 3D visualizations can be found in the Avatar system [20][21].

5. EventSpace Monitoring Experiments

To demonstrate the feasibility of the approach and the performance of the EventSpace prototype, we monitored a wind tunnel application running on the 4W and 8W clusters. We used eight matrices, each split into slices. The slices were split between 140 threads. Using PastSet, each thread exchanged the border entries of its slices with threads processing neighbor slices. We scaled the problem size to fill the 128MB memory on each of the 4W nodes. Slices were exchanged approximately every 300 ms (and thus virtual events are produced at about 3.3 Hz).

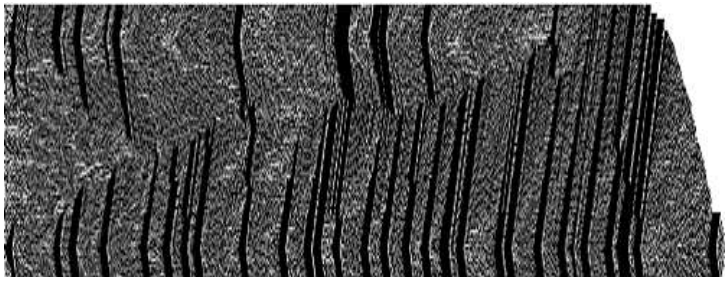
Event Collecting Overhead: The overhead introduced to the communication path by a single event collector wrapper is measured by adding event collector wrappers before and after it. The average overhead, calculated using recorded timestamps in these wrappers, is 1.1 μs on a 1.8GHz Pentium 4 and 6.1 μs on a 200 MHz 8W node. This is comparable to overheads reported for similar systems [22, 27]. The slowdown due to data collection is insignificant.

Event Scope Overhead: We measured how fast virtual events can be observed from the virtual event space, and the slowdown the wind-tunnel application will experience as a result of this.

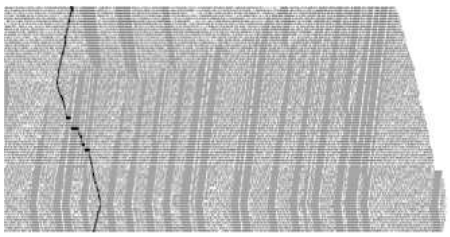
The rate at which virtual events can be extracted from the virtual event space decreases from 2000Hz to 2Hz as the number of concurrent extractions increases from one to around 2000. The amount of data extracted goes from 36 bytes to about 79KB. The slowdown of the wind tunnel application is always insignificant. For all experiments the average execution time for the wind tunnel is 612 seconds $\pm$ 14 seconds.

We concurrently consumed events located on different nodes. When consuming sequentially from all nodes, there is no difference in sample rates and slowdown. However, when more processing are added to the communication paths the concurrent version is faster. This is because we now have better overlap of communication and computation.

The event scopes used to monitor the collective operation resulted in a slowdown of 1.17. We discovered that an event scope actually perturbed the wind-tunnel more than the compute intensive monitoring threads running on each node of the clusters. By reconfiguring the event scopes to use less resources the slowdown was reduced to 1.08. But the trade-off was a 50% reduction in the observation rate.



(a) Communication and computation times for the 'four threads per CPU' configuration (in total 96 threads)



(b) Part of figure 3(a) with with 250th step highlighted.

Figure 3. Communication and computation times

When running four monitors concurrently, the slowdown was about the same as the largest slowdown caused by a single monitor. However, for the consumers, the observe rates were reduced by 10-50%.

6. Visualization Experiments

In this section we provide examples on how the data in a virtual event space can be used for analyzing the communication behavior of the wind tunnel parallel application. We used the same clusters as before, two in Tromso (4W, 8W) and one (2W) in Odense in Denmark. Communication between Tromsø and Odense is the departments Internet backbone with about 30-40ms latency, and a widely varying bandwidth (well below 34Mbits).

The wind-tunnel had linear scalability when run on the 4W cluster, and the perturbation due to monitoring could not be measured. We observed that when 200 steps were executed, having 4 threads per CPU[1] was about 5% faster than having 1 thread per CPU (the same problem size was used for both configurations). When the number of steps was increased to 500, they were equally fast.

In figure 3(a), there is one horizontal bar for each thread, indicating when it is using the communication system (black) and when it is is computing (light gray). On the horizontal-axis elapsed time is shown. We can see thick black stripes starting at the lower threads going upward. By using the step information displayed when pointing at a bar, we can see that most threads are one step ahead of the thread below (neighbors can only be one step apart).

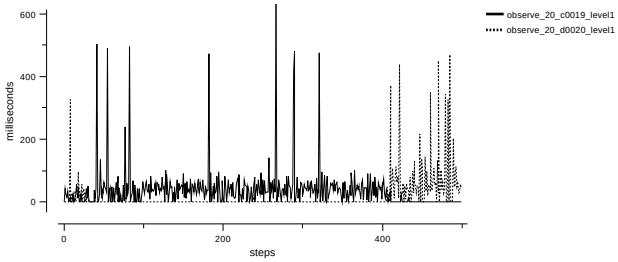


Figure 4: Read operation times for worker thread 20, on elements from thread above (solid), and from the thread below (dotted).

By looking at the completion times (where the bars end) we can see a *wavefront*, where the threads higher up finishes earlier than the threads further down. By highlighting some steps we found that after 100 steps the threads had gotten roughly equally far, after 200 steps we could see the wavefront shape starting to emerge, and after 400 steps it was clearly visible. An emerging wavefront is shown in figure 3(b), where the 250th step is highlighted. In the background the same black stripes as seen in figure 3(a) are shown. The threads above the strip, are not affected by the wavefront.

Figure 4 shows how thread 20 changes, at around the 400th step, from spending most of its time waiting for data from the thread above (solid line), to waiting for data from the thread below (dotted line). By highlighting

[1] Each thread did an equal amount of work

the 400th step in the communication-computation view, we can see that this is where the wavefront hits worker thread 20. Also the time per step is slightly increased after the 400th step.

The four thread per CPU configuration slows down as the computing proceeds, due to the communication behavior of the wind-tunnel application.

7. Related Work

There are many approaches and tools to parallelizing, configuring, monitoring, and visualizing the behavior of parallel applications. We do not have the space here to go into details, and to contrast existing work with ours. We have given several references earlier in the paper.

Other interesting approaches are: performance analysis tools [16, 27], NetLogger [27], firmware based distributed shared virtual memory (DSVM) system monitor for the SHRIMP multicomputer [15], network performance monitoring tools [11, 18], Infopipes [6], Gscope [12], Prism debugger for MPI [23], JAMM monitoring sensor management system [26], Network Weather Service [32], other monitoring systems include [11, 22, 27, 32], configurable monitoring systems include [11, 22, 27, 32].

8. Conclusions

Small and simple changes to a configuration influence scaling and latency. It is hard to find good configurations analytically or by computation. Instead, it is demonstrated that by starting with what is believed to be a good configuration, one can run a number of experiments using the PATHS system to identofy configurations with better performance.

The LAM-MPI implementation [10] of the MPI standard is documented to, out of the box, use configurations which do not scale well. A configuration mechanism has been added to LAM-MPI and used to double the performance of the MPI Allreduce function.

Thus, the ability to tune the configuration with knowledge about the application and the cluster topology, as opposed to relying on the implementation to do important optimization choices, is found to be important.

To get information about the behavior of an application we developed EventSpace on top of PATHS. Using data from EventSpace we have visualized the behavior of several applications, including a wind tunnel, and used it to find communication and load bottlenecks.

We find that the ability to configure, monitor and visualize the behavior of parallel applications are very usefull when experimenting to achieve better performance.

REFERENCES

[1] AMDAHL, G. Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. In *Proceedings of the AFIPS Spring Joint Computer Conference, Atlantic City, New Jersey, USA* (1967), AFIPS Press, Reston, Virginia, USA, pp. 483–485.

[2] ARPACI-DUSSEAU, A. C., ARPACI-DUSSEAU, R. H., CULLER, D. E., HELLERSTEIN, J. M., AND PATTERSON, D. A. Searching for the Sorting Record: Experiences in Tuning NOW-Sort. In *Proceedings of the SIGMETRICS symposium on Parallel and distributed tools (SPDT 98), USA* (1998), pp. 124–133.

[3] BJØRNDALEN, J. M., ANSHUS, O., LARSEN, T., BONGO, L. A., AND VINTER, B. Scalable Processing and Communication Performance in a Multi-Media Related Context. *Euromicro 2002, Dortmund, Germany* (September 2002).

[4] BJØRNDALEN, J. M., ANSHUS, O., VINTER, B., AND LARSEN, T. Configurable Collective Communication in LAM-MPI. *Proceedings of Communicating Process Architectures 2002, Reading, UK* (September 2002).

[5] BJØRNDALEN, J. M., ANSHUS, O., LARSEN, T., AND VINTER, B. PATHS - Integrating the Principles of Method-Combination and Remote Procedure Calls for Run-Time Configuration and Tuning of High-Performance Distributed Application. In *Norsk Informatikk Konferanse* (Nov. 2001), pp. 164–175.

[6] BLACK, A. P., HUANG, J., KOSTER, R., WALPOLE, J., AND PU, C. Infopipes: An abstraction for multimedia streaming. *Multimedia Systems, special issue on multimedia middleware* (2002). Volume 8 Issue 5 pp 406-419, Springer Verlag.

[7] BONGO, L. A. EventScope: Configurable On-line Monitoring of Parallel and Distributed Applications. Master's thesis, Department of Computer Science, University of Tromsø, Dec. 2002.

[8] BONGO, L. A., ANSHUS, O., AND BJØRNDALEN, J. M. Using a virtual event space to understand parallel application communication behavior, 2003. Technical Report 2003-44, Department of Computer Science, University of Tromsø.

[9] BONGO, L. A., ANSHUS, O. J., AND BJØRNDALEN, J. M. Eventspace - exposing and observing communication behavior of parallel cluster applications. Euro-Par 2003, August 2003, Klagenfurt, Austria.

[10]BURNS, G., DAOUD, R., , AND VAIGL, J. LAM: An Open Cluster Environment for MPI. www.lam-mpi.org, 1994.

[11]DINDA, P., GROSS, T., KARRER, R., LOWEKAMP, B., MILLER, N., STEENKISTE, P., AND SUTHERLAND, D. The architecture of the Remos system. In *Proc. 10th IEEE Symp. on High Performance Distributed Computing* (2001).

[12]GOEL, A., AND WALPOLE, J. Gscope: A visualization tool for time-sensitive software. In *In Proceedings of the Freenix Track of the 2002 USENIX Annual Technical Conference* (June 2002).

[13]GUSTAFSON, J. L. Reevaluating Amdahl's law. *Communications of the ACM 31*, 5 (1988), pp. 532–533.

[14]LAM-MPI homepage. http://www.lam-mpi.org/.

[15]LIAO, C., JIANG, D., IFTODE, L., MARTONOSI, M., AND CLARK, D. W. Monitoring shared virtual memory performance on a myrinet-based PC cluster. In *International Conference on Supercomputing* (1998), pp. 251–258.

[16]MOORE, S., D.CRONK, LONDON, K., AND J.DONGARRA. Review of performance analysis tools for MPI parallel programs. In *8th European PVM/MPI Users' Group Meeting, Lecture Notes in Computer Science 2131* (2001), Y. Cotronis and J. Dongarra, Eds., Springer Verlag.

[17]MPI: A Message-Passing Interface Standard. *Message Passing Interface Forum* (Mar. 1994).

[18]http://www.caida.org/tools/taxonomy/.

[19]PANDA, D. Issues in Designing Efficient and Practical Algorithms for Collective Communication in Wormhole-Routed Systems. *Proc. ICPP Workshop Challenges for Parallel processing* (1995), pp. 8–15.

[20]REED, D. A., MADHYASTHA, T. M., AYDT, R. A., ELFORD, C. L., SCULLIN, W. H., AND SMIRNI, E. I/O, Performance Analysis, and Performance Data Immersion. In *MASCOTS* (1996), pp. 5–15.

[21]REED, D. A., SHIELDS, K. A., SCULLIN, W. H., TAWERA, L. F., AND ELFORD, C. L. Virtual Reality and Parallel Systems Performance Analysis. *IEEE Computer 28*, 11 (1995), 57–67.

[22]RIBLER, R. L., VETTER, J. S., SIMITCI, H., AND REED, D. A. Autopilot: Adaptive control of distributed applications. In *Proc. of the 7th IEEE International Symposium on High Performance Distributed Computing* (1998).

[23]SISTARE, S., DORENKAMP, E., NEVIN, N., AND LOH, E. MPI support in the Prism programming environment. In *13th ACM International Conference on Supercomputing* (1999).

[24]SQUYRES, J. M., LUMSDAINE, A., GEORGE, W. L., HAGEDORN, J. G., AND DEVANEY, J. E. The Interoperable Message Passing Interface (IMPI) Extensions to LAM/MPI. In *Proceedings, MPIDC'2000* (March 2000).

[25]TIERNEY, B., AYDT, R., GUNTER, D., SMITH, W., TAYLOR, V., WOLSKI, R., AND SWANY, M. A grid monitoring architecture. *Tech. Rep. GWD-PERF-16-2, Global Grid Forum, January 2002.* (2002).

[26]TIERNEY, B., CROWLEY, B., GUNTER, D., HOLDING, M., LEE, J., AND THOMPSON, M. A monitoring sensor management system for Grid environments. In *Proc. 9th IEEE Symp. On High Performance Distributed Computing* (2000).

[27]TIERNEY, B., JOHNSTON, W. E., CROWLEY, B., HOO, G., BROOKS, C., AND GUNTER, D. The NetLogger methodology for high performance distributed systems performance analysis. In *Proc. 7th IEEE Symp. On High Performance Distributed Computing* (1998).

[28]TRAFF, J. L. Implementing the MPI Process Topology Mechanism. *Supercomputing 2002* (2002).

[29]VETTER, J. S., AND YOO, A. An empirical performance evaluation of scalable scientific applications. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing* (2002).

[30]VINTER, B. *PastSet a Structured Distributed Shared Memory System*. PhD thesis, Tromsø University, 1999.

[31]VINTER, B., ANSHUS, O. J., LARSEN, T., AND BJØRNDALEN, J. M. Extending the Applicability of Software DSM by Adding User Redefinable Memory Semantics. *Parallel Computing (ParCo) 2001, Naples, Italy* (Sept. 2001).

[32]WOLSKI, R., SPRING, N. T., AND HAYES, J. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems 15*, 5–6 (1999).