International Journal of High Performance Computing Applications, vol. 20, no. 4, Winter 2006, pp. 467-480

The Internet Operating System: Middleware for Adaptive Distributed Computing

Kaoutar El Maghraoui, Travis J. Desell, Boleslaw K. Szymanski, and Carlos A. Varela
Department of Computer Science
Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY 12180-3590, USA
{*elmagk,deselt,szymansk,cvarela*}*@cs.rpi.edu*

Abstract

Large-scale, dynamic, and heterogeneous networks of computational resources (a.k.a. grids*) promise to provide high performance and scalability to computationally intensive applications. To fulfill this promise, grid environments require complex resource management. We propose decentralized middleware-triggered dynamic reconfiguration strategies to enable application adaptation to the constantly changing resource availability of Internet-scale shared computational grids. As a proof of concept, we present a software framework for dynamically reconfigurable distributed applications. The* Internet Operating System (IOS) *is a middleware infrastructure which aims at freeing application developers from dealing with non-functional concerns while seeking to optimize application performance and global resource utilization. IOS consists of distributed middleware agents that are capable of interconnecting themselves in various virtual peer-to-peer topologies. IOS middleware agents: 1) profile application communication patterns, 2) evaluate the dynamics of the underlying physical resources, and 3) reconfigure application components by changing their mappings to physical resources through* migration *and by changing their granularity through a* split and merge *mechanism. A key characteristic of IOS is its decentralized coordination, thereby avoiding the use of global knowledge and thus enabling scalable reconfiguration. The IOS middleware is programming model-independent: we have implemented an actor programming model interface for SALSA programs and also a process programming model interface for MPI programs. Experimental results show that adaptive middleware can be an effective approach to reconfiguring distributed applications with various ratios of communication to computation in order to improve their performance, and more effectively utilize grid resources.*

1 Introduction

A wide variety of computational environments are increasingly available to host the execution of distributed and parallel applications. Examples include shared or dedicated clusters, supercomputers, and large-scale grid and meta-computing environments. Performance variability in such environments is the rule and not the exception. This poses new challenges for application developers that go far beyond those of parallelism and scalability. The complexity lies not only in how to optimize large applications to run on a given set of distributed resources, but also in how to maintain the desired performance when the pool of resources may change anytime, the characteristics of the resources are variable, and the application's demands vary during its life-time. In conventional distributed and parallel systems, achieving high

performance is often accomplished through application-level scheduling or application-specific tuning. Such techniques rely on the following assumptions: 1) having a good knowledge of the application's performance model, 2) executing over a relatively static set of resources, and 3) understanding well the characteristics of these resources.

A successful example of application-level reconfiguration is adaptive mesh refinement in parallel applications [27, 11, 14]. Other approaches have relied on new programming models and abstractions that promote composability, migration, and decentralized coordination [34, 7, 17, 30]. The latter work advocates a separation of concerns between the applications and the middleware. The middleware is responsible for addressing issues such as when and how to reconfigure applications. Applications need to be amenable to reconfiguration by supporting *migration* which involves moving computational units across physical resources, and/or *malleability* which enables splitting and merging computational units to change the application components' granularity.

We present a modular framework that supports application adaptation to both changing computational needs of applications and changing conditions of the execution environment. Reconfiguration is supported at the level of application components—e.g., processes or actors—to allow for flexible adaptation across a large range of distributed applications. The adopted approach tries to minimize the use of a detailed application performance model for the purpose of performance predictions. Component-level profiling is used to learn on the fly the behavior of the applications and to predict, based on the profiled information, how the application can best utilize currently available resources. Resource-level profiling helps match application components to newly available resources or away from resources becoming too slow or unavailable.

In large-scale network systems, it is very expensive, if not impossible, to maintain global and accurate knowledge about the behavior of the available resources. Designing reconfiguration policies that rely on partial and inaccurate knowledge becomes inevitable to achieve fast and scalable decisions. The proposed middleware embodies resource profiling and reconfiguration decisions into software agents. The middleware agents form a *virtual network*. Every agent monitors a network node containing one or more application entities and coordinates with peer agents to learn about available resource characteristics and application communication patterns. We have investigated two virtual topologies for inter-agent coordination: a *peer-to-peer* and a *cluster-to-cluster* virtual topology.

The remainder of the paper is organized as follows. In Section 2, we discuss middleware as an enabling layer capable of reconfiguring and hence adapting applications to dynamic environments. Section 3 outlines some of the key design issues that arise at the middleware level, at the application level and the interface between the middleware and the applications. Section 4 presents the IOS middleware prototype that has been designed and developed with these design concepts and issues in mind. In Section 5, we evaluate IOS performance. Finally, we conclude the paper with a brief discussion and related work.

2 Middleware-Driven Application Reconfiguration

Computational grids are inherently heterogeneous, shared, and dynamic. The efficient deployment of applications on such environments demands complex resource management strategies. The sheer size and dynamic behavior of grid environments makes automatic resource management a necessity. To effectively utilize grid resources, it is imperative to enable applications to dynamically reconfigure to adjust to the dynamics of the underlying resources. Dynamic reconfiguration implies the ability to modify the mapping between application components and physical resources and/or to modify the application components' granularity while the application continues to operate without any disruption of service. Applications should be able to scale up to exploit more resources as they become available or gracefully shrink down as some resources leave or experience failures. Dealing with reconfiguration issues over large-

scale dynamic environments is beyond the scope of application developers. It is therefore imperative to adopt a middleware-driven approach to achieve efficient deployment of distributed applications in a dynamic setting in a portable and transparent manner. One way of achieving this vision is embodying various middleware strategies and services within a virtual network of autonomous agents that coordinate applications and resources.

Middleware geared towards applications' reconfigurability should address the following issues:

Architectural Modularity Middleware agents should provide mechanisms that allow profiling meta-information about applications' behavior and resource usage characteristics, propagating this meta information, and deciding when and how to reconfigure applications. It is hard if not impossible to adopt one single strategy that is applicable to different types of execution environments and a large spectrum of applications. For example, it should be possible for the middleware to enable tuning the amount of profiling done depending on the load of the underlying resources. The more accurate the profiling is, the more informed the reconfiguration decisions are. A modular architecture is therefore critical to have extensible and pluggable reconfiguration mechanisms. Different profiling, coordination, and reconfiguration decisions strategies can be used depending on the class of applications and the characteristics of the underlying execution environment.

Generic Interfaces To allow middleware to handle various programming models and technologies, common and generic interfaces need to be designed. The functional details of how to accomplish reconfigurability such as migration, replication, or re-distribution of computation are different among various programming models. However, the strategies embodied in the middleware to achieve application's adaptation such as deciding how to disseminate resource characteristics and how to use meta-level information about applications and resources to effectively reconfigure applications is similar and therefore reusable across various programming paradigms. Generalizing and standardizing the interactions between applications and middleware leads to a more generic and portable approach capable of using new programming models and easily upgrading legacy code that use existing models to enable dynamic reconfiguration.

Decentralized and Scalable Strategies Grids consist of thousands—potentially millions—of globally distributed computational nodes. To address the demands of such large-scale environments with higher degrees of node failures and latency variability, decentralized management becomes a necessity. The sheer size of these computational environments renders using decisions based on global knowledge infeasible and impractical since the overhead of obtaining global information may overwhelm the benefits of using it. This forces any middleware to rely on making decisions based on local, partial, and often inaccurate information. There is an inherent trade-off between the overhead of information dissemination and the maintenance of high quality of information.

Dynamic Resource Availability Large-scale execution environments are expected to be more dynamic. Resource availability is constantly changing. Resources can join anytime during the lifetime of an application, for example, when users voluntarily engage their machines as part of a distributed computation over the Web, or when an over-loaded machine becomes suddenly available upon the completion of some tasks. Resources can also leave at anytime, for example, when a user withdraws her/his machine from a web-based computation or because of a node failure. To improve efficiency, middleware-driven reconfiguration can help applications adjust their resource allocation as the availability of the underlying physical resources changes.

3 Virtual Network Topologies and Models of Reconfiguration

Designing middleware for application reconfiguration over dynamic environments poses a number of challenges in terms of architectural modularity, generic interfaces to distributed applications, decentralized management of information, and efficient and effective decision making in the presence of uncertainty. In this section, we address some research challenges at the middleware level, the application level, and the interface between these layers.

3.1 Middleware and Application Issues

Computational grids consist of diverse topologies of computational nodes. In some cases, nodes are highly heterogeneous and geographically distributed, resembling nodes on the Internet, while in other cases, nodes are tightly clustered, and these clusters may be geographically distributed with high bandwidth connections, resembling hierarchical computational grids, such as the TeraGrid. Therefore, the type of decentralized coordination plays an important role in the efficiency and effectiveness of the middleware. Depending on the computational grid topology, a purely peer-to-peer coordination scheme might be more effective than a hierarchical coordination scheme. In other scenarios a hybrid approach can be more effective.

The connectivity of the middleware agents can be designed to be very high—e.g., a complete graph where every agent knows about every other agent—or very low—e.g., a ring, a tree, or a combination of these two, as frequently used in distributed hash tables. The connectivity level and the frequency and amount of communication between the middleware agents significantly influence the overhead of meta-level information exchange and the quality of information agents can afford. Ultimately, these decisions have an impact on the quality of application reconfiguration decisions that middleware agents take.

The middleware's ability to reconfigure an application dynamically is ultimately bound by the application's reconfiguration capabilities. Different levels of reconfigurability include application stop and restart mechanisms, application component migration and replication, and dynamic granularity of application components. The latter techniques require more complex support from application programming models and tend to be less transparent to application developers. However, they provide the most flexibility to middleware and bring up the best opportunities for adaptability and scalability.

3.2 Application Programming Models

Functional issues of reconfiguration such as how to migrate application entities or how to change the granularity of entities are highly dependent on the programming model and language used to develop a given application. It is imperative to have tools that augment applications with these capabilities. When such tools are available, the middleware can readily use them to automate the process of application adaptation in dynamic environments. The middleware can shield application developers from dealing with complex reconfiguration issues such as when and where to migrate application's entities and which entities need to be reconfigured.

In programming models such as actors [1], application component migration is obtained virtually for free since memory is completely encapsulated and distributed and communication is performed purely by asynchronous message passing. In programming models such as communicating processes (e.g., MPI [26]), additional libraries and application programming interfaces (API) are required to enable process migration. In both the actor and process programming models, dynamic component granularity requires help from the application developers since data repartitioning and process interconnectivity changes are usually highly application-dependent. Nonetheless, well-designed APIs can make the process of extending

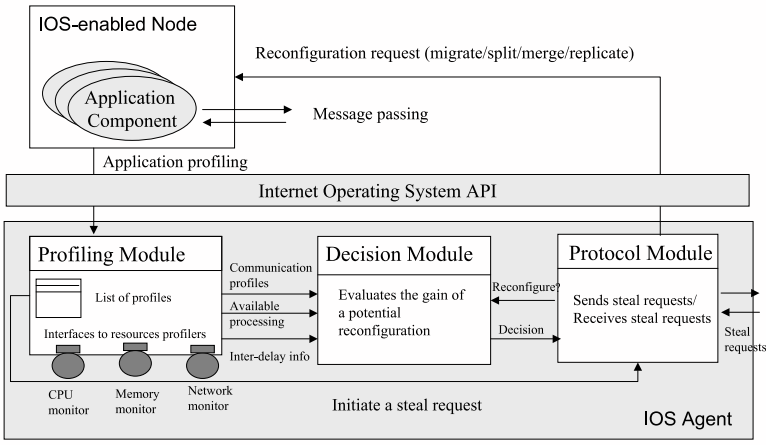

Figure 1. Interactions between a reconfigurable application and the local IOS agent.

applications to support dynamic reconfiguration more manageable. This is particularly true for massively parallel applications composed of relatively independent subtasks.

4 Generic Modular Middleware Architecture

In this section, we illustrate design and implementation decisions behind the Internet Operating System (IOS) middleware.

4.1 The IOS Architecture

The IOS architecture consists of distributed middleware agents that are capable of interconnecting themselves in various communication topologies. Figure 1 shows the architecture of an IOS agent and how it interacts with the application components that are hosted on a given node. Every IOS agent consists of three pluggable modules that enable evaluation of different types of autonomous reconfiguration, and also allow users to develop their own fine-tuned modules for particular applications. These are the profiling module, the decision module, and the protocol module:

Profiling Module: Resource-level and application-level profiling are used to gather dynamic performance profiles about physical resources and application entities (such as processes, objects, actors, components, or agents). Application entities communicate processing, communication, data access and memory usage profiles to the middleware via a profiling API [25]. Profiling monitors periodically measure and collect information about resources such as CPU power, memory, disk storage, and network bandwidth. The IOS architecture defines interfaces for profiling monitors to allow the use of various profiling technologies. Examples include the Network Weather Service (NWS) [36] and Globus Meta Discovery Service [10].

Decision Module: Using locally and remotely profiled information, the decision module determines how reconfiguration should be accomplished. Different reconfiguration strategies such as random work stealing, application topology sensitive work stealing and network topology sensitive work stealing have been implemented [12].

Protocol Module: The protocol module is responsible for inter-agent communication. It is used to distribute profiled information among the different middleware agents. The agents can connect themselves into various virtual agent topologies forming a *virtual network*. Examples include peer-to-peer and hierarchical interconnections.

4.2 Generic Interface Implementations

The application profiling and reconfiguration request APIs in IOS are generic. We have implemented two different programming models, SALSA and MPI, to illustrate generic middleware-triggered reconfiguration. Our choice of these two programming models has been influenced by the fact that SALSA is a programming language with several embedded features for reconfiguration such as migration and asynchronous message passing while MPI is a standard model that has gained a widespread usage among the parallel processing community. The following sections discuss some implementation details for both programming models.

SALSA/IOS Implementation SALSA (**S**imple **A**ctor **L**anguage **S**ystem and **A**rchitecture) [34] is a programming language for developing actor-oriented distributed applications. SALSA uses actors as a programming abstraction to model a unit of concurrency, mobility and distribution. Unbounded concurrency is supported through actor creation and asynchronous communication through message passing. SALSA has been designed with additional abstractions to facilitate programming reconfigurable distributed applications: universal naming, migration support, and coordination primitives.

Actors are inherently concurrent objects that communicate with each other via asynchronous message passing. An actor is an object because it encapsulates a state and a set of methods. It is autonomous because it is controlled by a single thread of execution. The anatomy of actors simplifies concurrency, synchronization, coordination, namespace management, memory management, scheduling, and mobility in distributed systems. SALSA and the actor model are extremely good candidates for use in developing autonomously reconfigurable applications.

Migrating SALSA Actors: Actors in SALSA can be migrated transparently to the application developer, as references to other actors are location independent, and the language run-time guarantees message delivery whether communication is remote or local. The anatomy of an actor simplifies migration since every actor encapsulates cleanly its state and since it is based on asynchronous message passing. Message forwarding protocols are used to ensure that no messages in transit are lost while an actor is migrating to a new location.

Autonomous SALSA Actors: To enable profiling and middleware-driven migration of SALSA actors, an actor behavior extends the `AutonomousActor` class. *Autonomous actors* automatically profile communication, memory and CPU usage. This information is directly communicated to the IOS profiling module through the IOS profiling API.

MPI/IOS Implementation MPI [26] is a widely used standard for developing parallel applications. However, the issues of scalability, adaptability and load balancing are not directly addressed by MPI. To maintain a good performance, MPI applications need to be able to scale up to accommodate new resources or shrink to accommodate leaving or slow resources. Most existing MPI implementations assume a static network environment. MPI implementations that support the MPI-2 Standard [19] provide some support for dynamic process management by allowing running processes to spawn new processes and communicate with them. However, developers still need to handle explicitly issues such as resource discovery, resource allocation, scheduling, profiling, and load balancing. Additional middleware support is therefore needed to relieve application developers from non-functional concerns while allowing high performance. In what follows we describe mechanisms adopted to achieve migration of MPI processes at the user-level and how iterative MPI applications have been integrated with IOS.

Migrating MPI Processes: We have achieved MPI process migration at the user level by rearranging MPI communicators. Migration is performed through a collaboration of all the participating MPI pro-

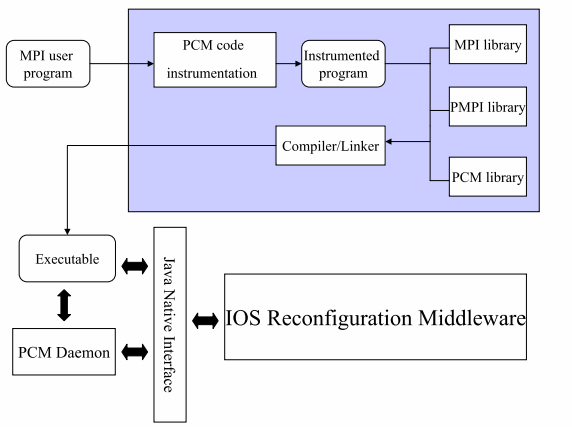

Figure 2. Library and executable structure of an MPI/IOS application.

cesses. It has to be done at a point where there are no pending communications. Process migration requires an update of any communicator that involves the migrating process. The class of iterative applications have natural barrier points at the beginning of each iteration when there is no pending communication. When necessary, we perform all reconfiguration at these times. A migration request forces all running MPI processes to enter a reconfiguration phase. The migrating process spawns a new process in the target location and sends it its locally checkpointed data.

Process migration and checkpointing support have been implemented as part of a user-level library. This approach allows portability across several vendor MPI implementations that support MPI-2 process spawning since the library is implemented entirely in the user space and does not require any infrastructural changes. The library is called PCM (Process Checkpointing and Migration) [24, 25].

Profiling MPI Processes: To accomplish application behavior profiling of MPI processes, their communication patterns must be periodically sent to their corresponding IOS profiling agents. This is achieved through a profiling library based on the MPI profiling interface (PMPI). The MPI specification provides a general mechanism for intercepting calls to MPI functions. This allows the development of portable performance analyzers and other tools without access to the MPI implementation source code. The only requirement is that every MPI function be callable by an alternate name (`PMPI_Xxxx` instead of the usual `MPI_Xxxx`). This profiling library intercepts all communication methods of MPI using a local PCM Daemon (PCMD) that sends periodically communication summaries to the local IOS profiling module.

4.3 IOS Module Implementations

The modularity of IOS allows for different implementations of the profiling, decision and protocol modules. This section presents sample implementations of IOS modules.

Virtual Network Implementations Different protocols have been implemented to organize the middleware agents into different decentralized *virtual networks*, which can adjust themselves according to the network topology of the execution environment. Two types of representative virtual networks have been investigated: *peer-to-peer* (p2p) and *cluster-to-cluster* (c2c). The p2p virtual network provides strictly peer-to-peer communication patterns between the middleware agents. The c2c virtual network provides a hierarchical arrangement of agents, by electing middleware agents to act as managers for groups of agents. Both virtual networks are sensitive to the fluctuations of the network load, and dynamically change themselves in response to changes in the network.

A Network Sensitive Peer-to-Peer Topology (NSp2p): Agents initially connect to the IOS virtual

network either through other known agents or through a *peer server*. Peer servers act as registries for agent discovery. Upon contacting a peer server, an agent registers itself and receives a list of other agents (peers) in the virtual network. Peer servers are only used for discovering peers in a virtual network. Being part of the virtual network, an agent can discover new peers as information gets propagated across peers. Agents can also dynamically leave the virtual network. Each agent will keep a list of peers at various distances (determined by latency), and will request reconfiguration with low latency peers before high latency peers [12].

A Network Sensitive Cluster-to-Cluster Topology (NSc2c): In NSc2c, agents are organized into groups of virtual clusters (VCs) which have collectively low latency communication. Each VC elects one agent to act as the cluster manager. Cluster managers view each other as peers and organize themselves in the same manner as the NSp2p virtual network topology.

Reconfiguration Decision Strategies Current implementations of decision strategies involve a *decision function* which can utilize profiled information to determine if it is worthy to migrate an actor or a process from one node to the other. Two strategies have been evaluated: random stealing (RS) and application-sensitive random stealing (ARS) [12].

Random Work Stealing (RS): Random work stealing inherits ideas from previous work stealing approaches [8]. Nodes with light computation and communication loads attempt to steal work from heavily loaded nodes. This reconfiguration strategy is quite simple and has low overhead as only communication and computation load at each node must be profiled. No information about applications characteristics is required. This strategy is stable since it does not cause additional overhead when the system is already over-loaded.

Application-sensitive Random Stealing (ARS): Application-sensitive random stealing is an extension of RS by taking into account the behavior of the application. The rationale behind ARS is collocating highly synchronized application's entities within nodes with low latency interconnections. ARS provides significant improvement over RS for a large class of applications (see Section 5).

5 Profiling and Reconfiguration Performance Results

This section discusses some key experimental results pertaining to the IOS middleware. We show the effect of using knowledge about the application's topology in the reconfiguration decisions and the effect of certain virtual topologies on the quality of reconfiguration. We also evaluate the effect of reconfiguration using both SALSA and MPI.

5.1 Application-Sensitive Reconfiguration Results

In previous work [12], we have shown that using application behavior information to accomplish autonomous reconfiguration can be highly effective. Additionally, in both programming models used, the overhead of IOS is very low and in some cases negligible. For the experimental testbed we used a four-dual node cluster of SUN Blade 1000 machines. Each node has a processing speed of 750M cycles per second and 2 GB of memory.

Autonomous Reconfiguration using SALSA/IOS SALSA benchmarks were developed to model various communication topologies with an emphasis on the computation-to-communication ratio of the applications. Four different application topologies are represented, each pertaining to different levels of inter-actor synchronization. The unconnected topology represents massively parallel applications. The

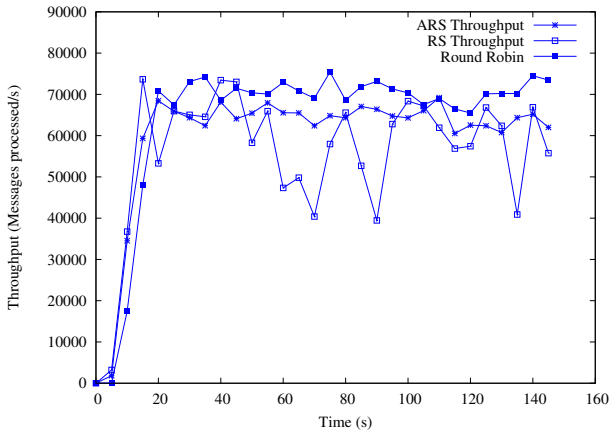

Figure 3. Performance of the massively parallel unconnected benchmark.

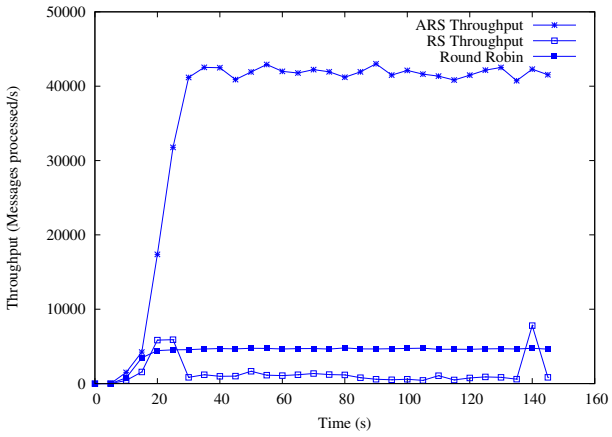

Figure 4. Performance of the massively parallel sparse benchmark.

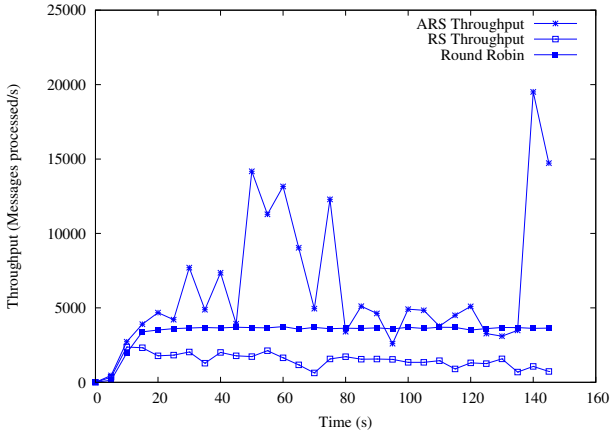

Figure 5. Performance of the highly synchronized tree benchmark.

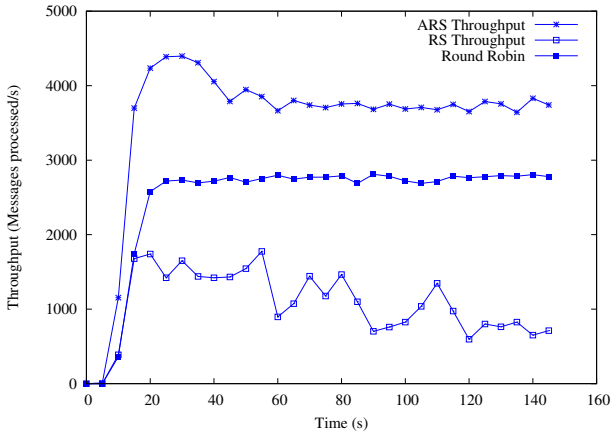

Figure 6. Performance of the highly synchronized hypercube benchmark.

sparse topology models applications with a low communication to computation ratio. Both the tree and hypercube topologies model higher communication to computation ratios.

Figures 3, 4, 5, and 6 show that a reconfiguration strategy that takes application behavior into consideration (such as the application's communication topology), results in largely improved performance on applications with varying levels of synchronization, over strategies that do not, such as RS. Figure 3 also shows that the overhead of the SALSA/IOS middleware is not significant—compare ARS to a round-robin strategy with no reconfiguration overhead. We used the number of messages processed per second (throughput) as a performance metric in these experiments. The benchmarks studied are characterized by having uniformly distributed communication throughout execution. They have also been designed to be long-running applications. Therefore, the number of processed messages per second gives a good measure of how well the application is performing.

Autonomous Reconfiguration using MPI/IOS An implementation of a two-dimensional heat diffusion application in MPI was used to evaluate the reconfiguration capabilities of MPI using IOS. For comparative purposes, we used MPICH2 [6], a free implementation of the MPI-2 standard. We emulated a shared and dynamic environment with varying load conditions by introducing artificial load in some of the cluster nodes and varying it periodically.

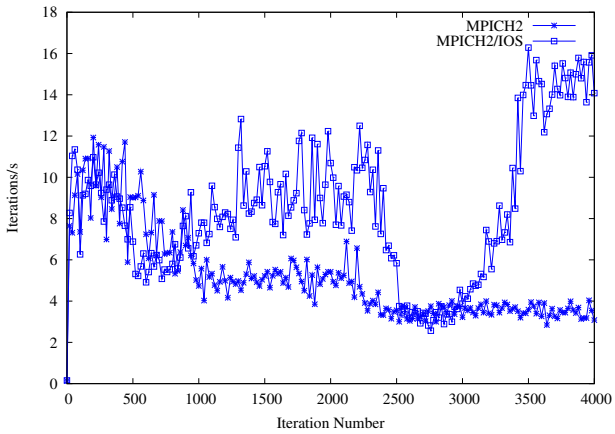

Figure 7. Performance of the heat simulation application using MPICH2 and MPICH2/IOS.

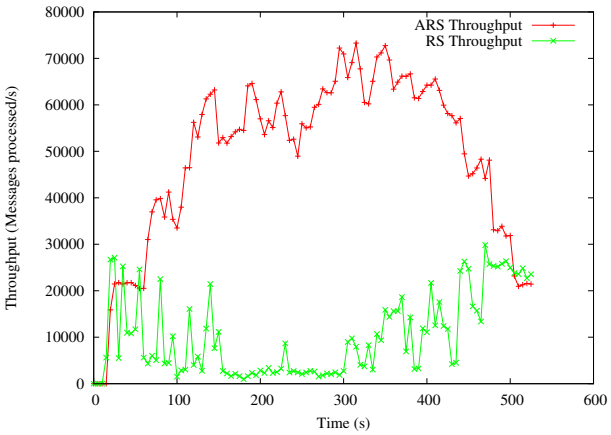

Figure 8. The tree topology on a dynamic environment using ARS and RS.

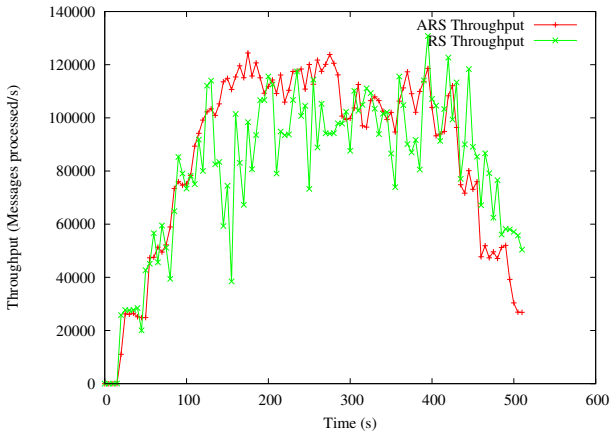

Figure 9. The unconnected topology on a dynamic environment using ARS and RS.

In both experiments, the cluster's load was varied by increasing or decreasing the processors loads at various intervals. Figure 7 shows that IOS allows the MPI application to adapt by trying to get rid of slow processors anytime an expected increase in performance is predicted through migration. This evaluation occurs when the work stealing mechanism gets triggered by a processor becoming lightly-loaded or a new processor becoming available. The figure also shows the performance of the same application under the same dynamic environment with no reconfiguration capabilities. With no ability to adapt, the application used the initial configuration throughout the course of the experiment and experienced a constant degradation in its performance. This degradation was significant as the highly synchronized nature of this application causes it to run as fast as the slowest processor.

5.2 Throughput and Scalability Improvements

Migration has been used to allow applications to utilize dynamic environments, by allowing the application to use new nodes as they become available, and to withdraw from nodes that become unavailable or over-loaded. Figures 8 and 9 show two different applications, one tightly coupled and the other massively parallel, executing on a dynamic environment using two different reconfiguration strategies. In both tests, the applications were run on a single node, and every 30 seconds an additional node was made available. After allowing the application to execute on 8 nodes for three minutes, one by one, nodes were

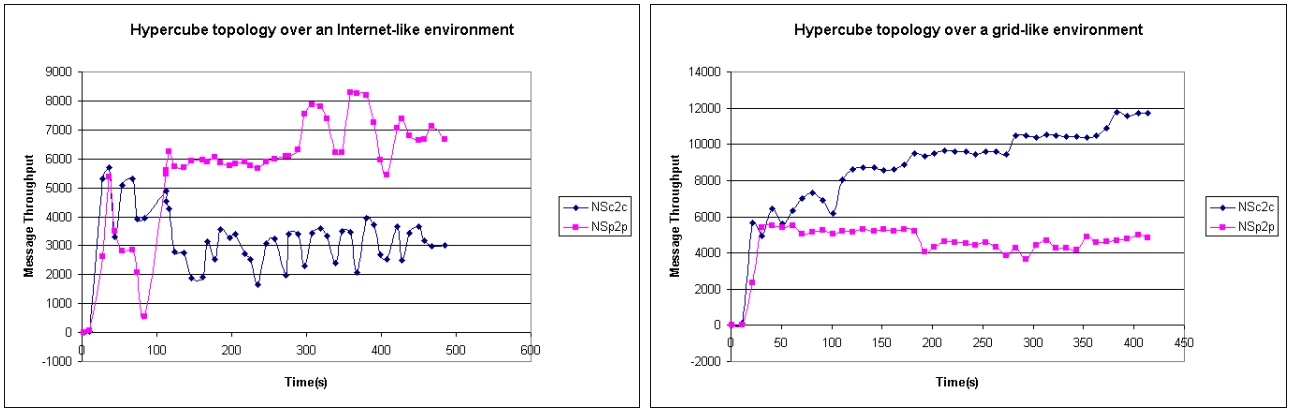

Figure 10. Message throughput for the hypercube application topology on Internet- and Grid-like environments.

becoming unavailable every thirty seconds until only one node was remaining. These results show that the middleware can effectively reconfigure the application to use the available resources as they change over time. While the massively parallel application was able to utilize the available resources using both an application topology sensitive reconfiguration strategy (ARS) and a strategy based solely on resource availability (RS), the tightly coupled application showed improvement only when ARS was used.

5.3 Virtual Network Evaluation

We have evaluated different arrangements of middleware agents using NSp2p and NSc2c to propagate profiled information for use in different reconfiguration strategies. Two different physical environments were used to model Internet-like networks and Grid-like networks. The first physical network consisted of 20 machines running Solaris and Windows operating systems with different processing power and different latencies to model the heterogeneity of Internet computing environments. The second physical network consisted of 5 clusters with different inter-cluster network latencies. Each cluster consists of 5 homogeneous SUN Solaris machines. Machines in different clusters have different processing power.

Figures 10 and 11 show not only that decentralized middleware management can accomplish intelligent application reconfiguration, but also that the virtual network used plays a role in the effectiveness of the reconfiguration. The p2p topology performs better in Internet-like environments that lack structure for highly synchronized parallel and distributed applications, while the c2c topology is more suitable for grid-like environments that have a rather hierarchical structure.

5.4 Overhead Evaluation

The overhead of using IOS with SALSA was evaluated using two applications, a massively parallel astronomic application and a tightly coupled two-dimensional heat diffusion application. Both applications are iterative. Therefore the average time spent of executing every iteration is a good measure of how well the application is performing. Figures 12 and 13 show the the time spent by each iteration using SALSA and SALSA/IOS. In both cases the overhead was minimal, around 0.5% for the astronomic application and 2% for the two-dimensional heat diffusion application.

To evaluate the cost of reconfiguration of the PCM library, we varied the problem data size of the heat diffusion application and measured the overhead of reconfiguration in each case. In the conducted experiments, we started the application on a local cluster. We then introduced artificial load in one

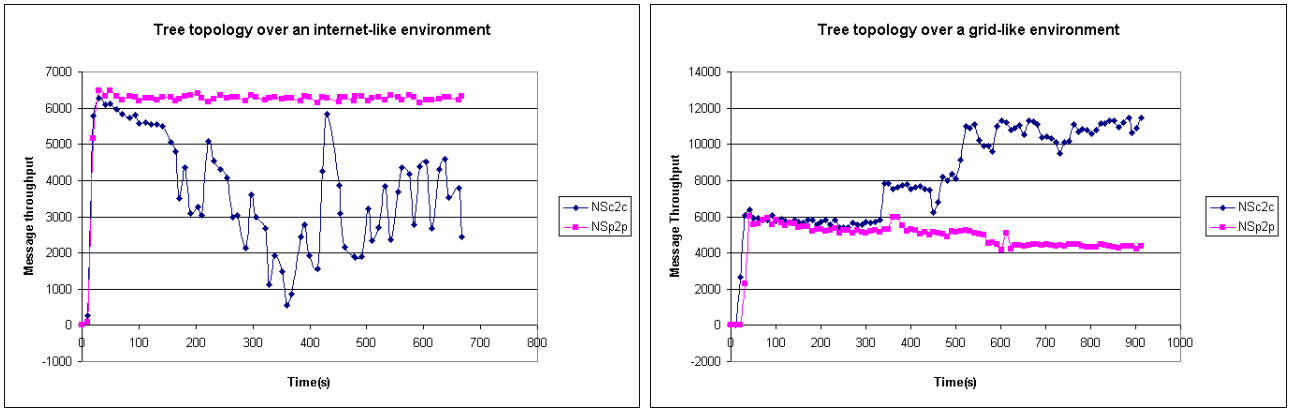

Figure 11. Message throughput for the tree application topology on Internet- and Grid-like environments.

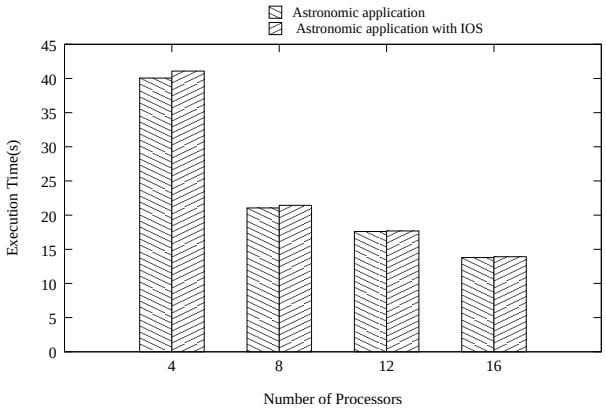

Figure 12. Overhead of using SALSA/IOS on a massively parallel astronomic data-modeling application with various degrees of parallelism.

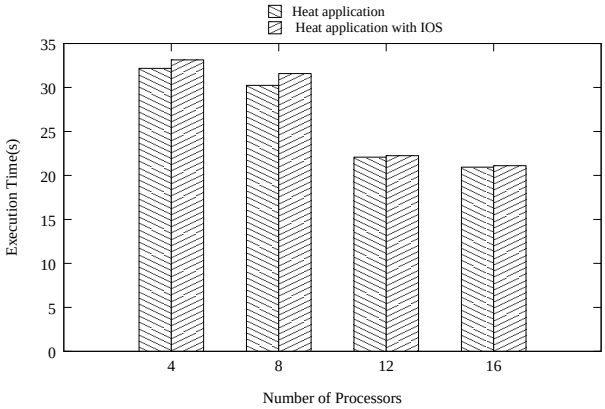

Figure 13. Overhead of using SALSA/IOS on a tightly synchronized two-dimensional heat diffusion application with various degrees of parallelism.

of the participating machines. One execution was allowed to reconfigure by migrating the suffering process to an available node that belongs to a different cluster, while the second execution was not allowed to reconfigure itself. The experiments in Figures 14 and 15 show that in the studied cases, reconfiguration overhead was negligible. In all cases, it accounted for less than 1% of the total execution time. The application studied is not data-intensive. We also used an experimental testbed that consisted of 2 clusters that belong to the same institution. So the network latencies were not significant. The reconfiguration overhead is expected to increase with larger latencies and larger data sizes. However, reconfiguration will still be beneficial in the case of large-scale long-running applications. Figure 15 shows the breakdown of the reconfiguration cost. The overhead measured consisted mainly of the costs of checkpointing, migration, and the synchronizations involved in re-arranging the MPI communicators. Due to the highly synchronous nature of this application, communication profiling was not used because a simple decision function that takes into account the profiling of the CPU usage was enough to yield good reconfiguration decisions.

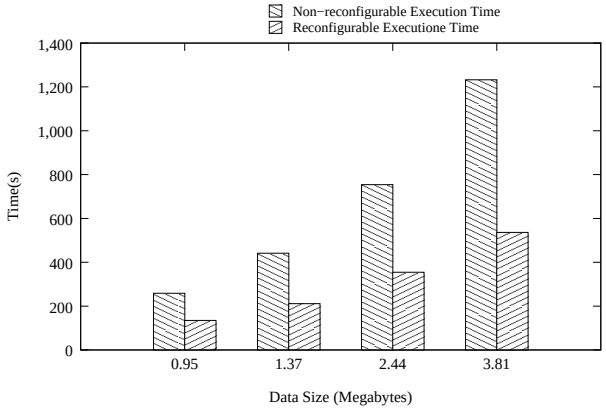

Figure 14. Execution time of reconfigurable and non-reconfigurable execution scenarios for different problem data sizes for the heat diffusion application.

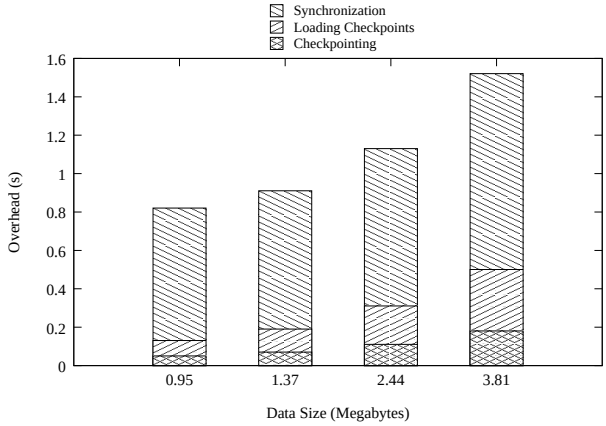

Figure 15. Breakdown of the reconfiguration overhead for the experiment of Figure 14.

6 Discussion and Related Work

Several research groups have been trying to achieve distributed computing on a large scale. Wisconsin's Condor project studies high throughput computing by "hunting" for idle workstation cycles [31]. The Globus project seeks to enable the construction of larger *computational grids* [15]. Virginia's Legion meta-system integrates research in object-oriented parallel processing, distributed computing, and security to form a programmable world-wide virtual computer [18]. WebOS seeks to provide operating system services, such as client authentication, naming, and persistent storage, to wide area applications [33]. UIUC's 2K is an integrated operating system architecture addressing the problems of resource management in heterogeneous networks, dynamic adaptability, and configuration of component-based distributed applications [22].

Research on process and data migration for grid applications includes [32, 28]. A key idea behind these approaches is to detect service degradations and react to such events by dynamically reconfiguring application processes to effectively use available computational resources. An adaptive middleware layer is needed, capable of migrating and replicating data and processes proactively based on the dynamically changing availability of resources on the grid [12, 2, 23]. Peer-to-peer concepts have also been used in the JUXMEM [5] project to enable data migration and sharing in dynamic grid environments.

While adaptive process and data migration and replication can have a large impact on the performance of grid computing applications, they both assume a reasonable underlying model of resource usage and expected future performance and availability of grid resources. Two mechanisms to predict performance based on profiling resource usage are the Network Weather Service (NWS) [36] and the Globus Meta Discovery Service (MDS) [10]. Recent research has devised and evaluated different mechanisms for resource management in dynamic heterogeneous grid environments—e.g., see [16, 4, 20, 21, 3].

An important consideration when adapting applications to dynamic grid environments through proactive data and process migration and replication is that the failure semantics of applications changes considerably. Research on fault detection and recovery through checkpointing and replication includes [13, 29, 9]. Notice that an application checkpointing mechanism is necessary for adaptive application migration and can readily be used for fault tolerance as well. More fine-grained process-level rather than application-level checkpointing and migration requires logging messages in transit to properly restore a distributed application state upon failure [13].

There is a large body of research into computational grids and grid-based middleware, hence this section only attempts to discuss a selection of this research area (see [35] for a more complete survey). This work demonstrates that middleware can provide effective application reconfiguration using different adaptation tools, based on profiling of application behavior and the execution environment, thus making it generically applicable to different applications and programming models. Additionally, such types of middleware and reconfiguration tools can enable applications to efficiently utilize dynamically changing environments, which could lead to more efficient utilization of grid resources. Finally, for future grid environments, decentralized management is possible and worthy of further research.

Acknowledgments

This work has been partially supported by the following grants: IBM SUR Award 2003, IBM SUR Award 2004, NSF CAREER Award No. CNS-0448407, and NSF INT No. 0334667.

References

[1] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1986.

[2] G. Agha and C. Varela. Worldwide computing middleware. In M. Singh, editor, *Practical Handbook on Internet Computing*, pages 38.1–21. CRC Press, 2004.

[3] G. Allen, D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, E. Seidel, and J. Shalf. The Cactus Worm: Experiments with dynamic resource selection and allocation in a grid environment. *International Journal of High-Performance Computing Applications*, 15(4):345–358, 2001.

[4] D. Angulo, I. Foster, C. Liu, and L. Yang. Design and evaluation of a resource selection framework for grid applications. In *IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, Edinburgh, Scotland, July 2002.

[5] G. Antoniu, J.-F. Deverge, and S. Monnet. How to bring together fault tolerance and data consistency to enable grid data sharing. *Concurrency and Computation: Practice and Experience*, (17), 2006. To appear.

[6] Argone National Laboratory. MPICH2, `http://www-unix.mcs.anl.gov/mpi/mpich2`.

[7] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski. The GrADS project: Software support for high-level grid application development. *International Journal of High-Performance Computing Applications*, 15(4):327–344, 2002.

[8] R. D. Blumofe and C. E. Leiserson. Scheduling Multithreaded Computations by Work Stealing. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS '94)*, pages 356–368, Santa Fe, New Mexico, November 1994.

[9] A. Bouteiller, F. Cappello, T. Hrault, G. Krawezik, P. Lemarinier, and F. Magniette. MPICH-V2: A fault tolerant MPI for volatile nodes based on the pessimistic sender based message logging. In *Supercomputing 2003*, Phoenix, USA, November 2003.

[10] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *10th IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, August 2001.

[11] H. L. de Cougny, K. D. Devine, J. E. Flaherty, R. M. Loy, C. Özturan, and M. S. Shephard. Load balancing for the parallel adaptive solution of partial differential equations. *Appl. Numer. Math.*, 16:157–182, 1994.

[12] T. Desell, K. E. Maghraoui, and C. Varela. Load balancing of autonomous actors over dynamic networks. In *Hawaii International Conference on System Sciences, HICSS-37 Software Technology Track*, Hawaii, January 2004.

[13] J. Field and C. Varela. Transactors: A programming model for maintaining globally consistent distributed state in unreliable environments. In *Conference on Principles of Programming Languages (POPL 2005)*, pages 195–208, 2005.

[14] J. E. Flaherty, R. M. Loy, C. Özturan, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. Parallel structures and dynamic load balancing for adaptive finite element computation. *Applied Numerical Mathematics*, 26:241–263, 1998.

[15] I. Foster and C. Kesselman. The Globus Project: A Status Report. In J. Antonio, editor, *Proceedings of the Seventh Heterogeneous Computing Workshop (HCW '98)*, pages 4–18. IEEE Computer Society, March 1998.
[16] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.
[17] D. Gannon, R. Bramley, G. Fox, S. Smallen, A. Rossi, R. Ananthakrishnan, F. Bertrand, K. Chiu, M. Farrellee, M. Govindaraju, S. Krishnan, L. Ramakrishnan, Y. Simmhan, A. Slominski, Y. Ma, C. Olariu, and N. Rey-Cenevaz. Programming the Grid: Distributed Software Components, P2P and Grid Web Services for Scientific Applications. *Journal of Cluster Computing*, 2002.
[18] A. S. Grimshaw, W. A. Wulf, and "the Legion team". The legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, January 1997.
[19] W. Gropp and E. Lusk. Dynamic process management in an MPI setting. In *Proceedings of the 7th IEEE Symposium on Parallel and Distributeed Processing*, page 530. IEEE Computer Society, 1995.
[20] A. Iamnitchi and I. Foster. On fully decentralized resource discovery in grid environments. In *International Workshop on Grid Computing*, Denver, Colorado, USA, November 2001.
[21] N. Karonis, B. de Supinski, I. Foster, W. Gropp, E. Lusk, and J. Bresnahan. Exploiting hierarchy in parallel computer networks to optimize collective operation performance. In *14th International Parallel Distributed Processing Symposium (IPDPS'00)*, pages 377–384, Cancun, Mexico, May 2000.
[22] F. Kon, R. H. Campbell, M. D. Mickunas, K. Nahrstedt, and F. J. Ballesteros. 2K: A Distributed Operating System for Dynamic Heterogeneous Environments. In *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing (HPDC'9)*, pages 201–208, Pittsburgh, August 2000.
[23] H. Lamehamedi, Z. Shentu, B. Szymanski, and E. Deelman. Simulation of dynamic data replication strategies in data grids. In *Proceedings of the 12th Heterogeneous Computing Workshop*, 2003.
[24] K. E. Maghraoui, T. Desell, B. K. Szymanski, J. D. Teresco, and C. A. Varela. Towards a middleware framework for dynamically reconfigurable scientific computing. In L. Grandinetti, editor, *Grid Computing and New Frontiers of High Performance Processing*. Elsevier, 2005.
[25] K. E. Maghraoui, B. K. Szymanski, and C. Varela. An architecture for reconfigurable iterative MPI applications in dynamic environments. In *Proc. Sixth International Conference on Parallel Processing and Applied Mathematics (PPAM 2005)*, Lecture Notes in Computer Science. Springer Verlag, 2005.
[26] Message Passing Interface Forum. MPI: A message-passing interface standard. *The International Journal of Supercomputer Applications and High Performance Computing*, 8(3/4):159–416, Fall/Winter 1994.
[27] J.-F. Remacle, J. Flaherty, and M. Shephard. An adaptive discontinuous Galerkin technique with an orthogonal basis applied to compressible flow problems. *SIAM Review*, 45(1):53–72, 2003.
[28] O. Sievert and H. Casanova. A simple MPI process swapping architecture for iterative applications. *International Journal of High Performance Computing Applications*, 2003.
[29] P. Stelling, I. Foster, C. Kesselman, C.Lee, and G. von Laszewski. A Fault Detection Service for Wide Area Distributed Computations. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing*, pages 268–278, Chicago, IL, 28-31 July 1998.
[30] K. Taura, K. Kaneda, and T. Endo. Phoenix: a Parallel Programming Model for Accommodating Dynamically Joining/Leaving Resources. In *Proc. of PPoPP*, pages 216–229. ACM, 2003.
[31] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The Condor Experience. *Concurrency and Computation: Practice and Experience*, 2004.
[32] S. S. Vadhiyar and J. J. Dongarra. A performance oriented migration framework for the grid. In *CCGrid, IEEE Computing Clusters and the Grid*, Tokyo, Japan, may 2003.
[33] A. Vahdat, T. Anderson, M. Dahlin, D. Culler, E. Belani, P. Eastham, and C. Yoshikawa. WebOS: Operating System Services For Wide Area Applications. In *Proceedings of the Seventh IEEE Symposium on High Performance Distributed Computing*, July 1998.
[34] C. Varela and G. Agha. Programming dynamically reconfigurable open systems with SALSA. *ACM SIGPLAN Notices. OOPSLA'2001 Intriguing Technology Track Proceedings*, 36(12):20–34, Dec. 2001. http://www.cs.rpi.edu/~cvarela/oopsla2001.pdf.
[35] C. A. Varela, P. Ciancarini, and K. Taura. Worldwide computing: Adaptive middleware and programming technology for dynamic Grid environments. *Scientific Programming Journal*, 13(4):255–263, December 2005. Guest Editorial.
[36] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service: A distributed resource performance forecasting service for metacomputing. *Future Generation Comput. Syst.*, 15(5-6):757–768, October 1999.