Deploying Fault-tolerance and Task Migration with NetSolve

James S. Plank[*] Henri Casanova[*] Micah Beck[*] Jack Dongarra[*†]

September 2, 1998

Abstract

Computational power grids are computing environments with massive resources for processing and storage. While these resources may be pervasive, harnessing them is a major challenge for the average user. NetSolve is a software environment that addresses this concern. A fundamental feature of NetSolve is its integration of fault-tolerance and task migration in a way that is transparent to the end user. In this paper, we discuss how NetSolve's structure allows for the seamless integration of fault-tolerance and migration in grid applications, and present the specific approaches that have been and are currently being implemented within NetSolve.

Keywords
Fault-tolerance, Scientific Computing, Computational Servers,
Checkpointing, Migration.

1 Introduction

The advances in computer and network technologies that are shaping the global information infrastructure are also producing a new vision of how that infrastructure will be used. The concept of a *Computational Power Grid* has emerged to capture the vision of a network computing system that provides broad access not only to massive information resources, but to massive computational resources as well. Such computational grids will use high-performance network technology to connect hardware, software, instruments, databases, and people into a seamless web that supports a new generation of computation-rich problem solving environments for scientists and engineers.

Grid resources will be ubiquitous. However, for the average scientific user, harnessing their power will present a challenge. Consider such a user. In the world of uniprocessor workstations, his life is relatively simple. Software packages such as MATLAB [21] and Mathematica [41] enable him to solve a wide variety of numerical problems with a convenient and flexible user interface. For less standard problems, he may obtain software solutions from a repository like Netlib. These are typically rather simple to incorporate into his programming platform. However, when his computational needs grow beyond the capability of a workstation, he must turn to parallel computing platforms. He will be driven to employ computational grids.

While computational grids offer tremendous computing power, they also combine and amplify all the well-known complexities of distributed and parallel computing environments. Moreover, they

[*]Department of Computer Science, University of Tennessee, TN 37996
[†]Mathematical Science Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831

deprive the programmer of convenient and flexible user interfaces. The obstacles and difficulties standing in the way of routine and effective use of such environments include the following:

- **Distributed ownership** – Often multiple machines are available to perform computation, but each is owned by a different person or group, which may need exclusive use of the machine periodically. Some machines have shared ownership, making the machines available on set schedules. Alternatively, a machine may never available for exclusive use, having the capacity to perform computations in one minute, and to be loaded by other computations in the next minute. Finally, distributed ownership implies that each resource must be considered transient as there is no single administrative authority. All of these characteristics make the task of managing the collection of machines as a single resource very difficult.

- **Platform heterogeneity** – While the processing power of a set of machines may be large, it may be hard to harness because each machine is a different type. This makes the task of partitioning a program among the available machines difficult. Moreover, it makes porting software difficult because different machines may have different compilers and libraries.

- **Network reliability and performance** – When machines are geographically separated, proximity becomes an important issue, because communication times are not uniform between all machines in the collection. Moreover, as larger distances separate machines, the chance of network disconnections increases.

- **Storage availability** – Like CPU capacity, both primary and secondary storage may exist in abundance on a grid, but their availability may be transient and hard to predict.

Thus, while the sheer processing power and storage capacity of computational grids make them attractive, their usability is a major concern. NetSolve, developed at the University of Tennessee and Oak Ridge National Laboratory, is a software environment for network computing that addresses this concern. Its central purpose is to enable the creation of complex applications that can deliver the immense power of computational grids to the desktops of users without being complicated to use or difficult to deploy. To achieve this aim, NetSolve uses a modular, *client-agent-server* architecture that composes a system in which processing platforms and computational software of arbitrary complexity become accessible to users through programming and problem solving interfaces that are already familiar and easy to use.

In this paper, we present the general structure of NetSolve, and how that structure accommodates grid computing. We then elaborate on how NetSolve integrates fault-tolerance and migration into its architecture so that it may harness the power of dynamically changing grid resources in ways that are transparent to the end user. This integration is split into two research directions. The first focuses on how NetSolve's design and implementation can enable fault-tolerance and migration, while the second simply uses NetSolve to explore a variety of approaches to fault-tolerance and migration. We will mention early results and design decisions throughout.

2 NetSolve

2.1 Overview

NetSolve is a software environment for networked computing designed to transform disparate computers and software libraries into a unified, easy-to-access computational service. It aggregates the hardware and software resources of any number of computers that are loosely connected across a network and offers their combined power through client interfaces that are familiar from the world of uniprocessor computing (e.g. MATLAB, simple procedure calls). It uses a *client-agent-server* paradigm (Figure 1) to deliver the power while hiding the complexity of the underlying system.

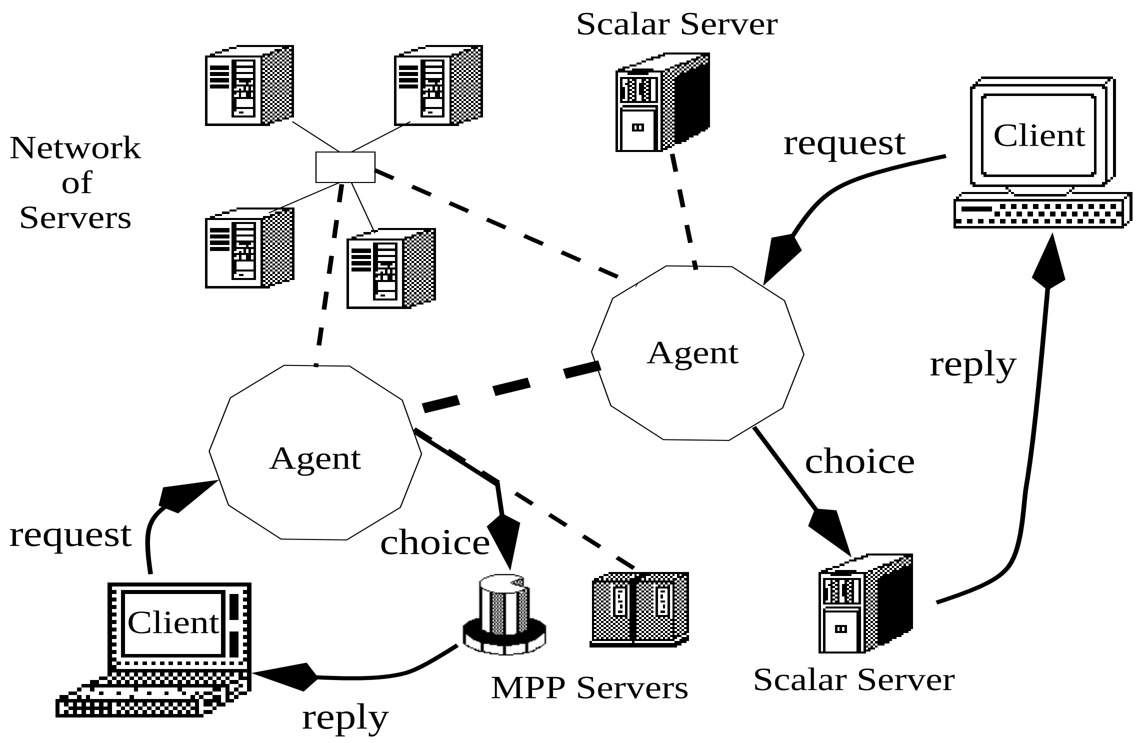

Figure 1: NetSolve's organization

A NetSolve server may be arbitrarily complex, from a uniprocessor to a MPP (Massively Parallel Processor) or a large networked cluster of machines; moreover there is no upper bound on the number of servers that can be aggregated to form a NetSolve resource pool, and new servers can be added with little effort.

When a user wants a certain computational task to be performed, he/she can use any one of a number of conventional software clients to contact an agent with the request. The agent keeps track of information about all the servers in its resource pool, including their availability, load, network accessibility, and the range of computational tasks they can perform. The agent then selects a server to perform the task, and the server responds to the client's request. The client's computations are performed remotely at the server. The client's data is sent to the server and the server uses its installed software libraries to perform the computation. When the server finishes the computation, it sends the result back to the client and the agent is notified of the completion of the task.

As shown in Figure 1, there may be multiple instances of NetSolve agents on the network, and different clients may contact different agents depending on their locations. The agents exchange information about their different servers, and allow access from any client to any server if desirable. NetSolve can be used either via the Internet or on an intranet, such as inside a department of a university, without participating in any Internet-based computation.

2.2 Strengths of NetSolve

The elements of the NetSolve architecture are for the most part well understood. Depending on your perspective, NetSolve's design can be viewed as an enhanced client-server architecture, or as an enhanced RPC (remote procedure call) environment. However, its three-tiered approach achieves a logical separation that allows each part to do what it does best without worrying about the details of the other parts. Specifically:

- **The client simply specifies the problem**. This simplicity of use is not only exactly what users typically want, it's what users actually get with uniprocessor packages such as MATLAB or LAPACK [2]. In the parallel world, however, even the simplest software packages burden the user with setting up the parallel environment, distributing data, specifying block sizes and processor grids, and so on. And this does not take into account heterogeneity, failures, or load-balancing. With NetSolve, the user goes back to the uniprocessor model, and lets the NetSolve agent and pool of servers worry about the other details.
- **The servers can optimize computations for their particular architectures**. One of the biggest challenges in writing software for solving problems on parallel systems is making the software general-purpose enough to work on a variety of parallel platforms, yet customizable enough to take advantage of the architectural features of each platform. The typical result is that the software either has sub-optimal performance and is easy to use, or has optimal performance, but places a significant burden on the user. ScaLAPACK [6], for instance, achieves excellent performance on a variety of parallel and distributed computing platforms, but is orders of magnitude more difficult to port and use than its uniprocessor counterpart, LAPACK. With NetSolve, every server can be set up with software that performs optimally for that server, without any end user's involvement.
- **The agents act as resource brokers**. The agents perform the tasks that are not logical for the clients or servers, such as monitoring and managing resources, services and computations. The agent is the "third part" of the three-part design of NetSolve, which liberates the other two parts. Namely, it enables the clients to focus solely on specifying their problems, and it enables the servers to focus solely on solving their problems.

2.3 Current State of NetSolve Development

At present NetSolve exists as a prototype that is being used in several research institutions. This prototype implements NetSolve's client-agent-server model in the following way.

- Clients may be C or Fortran programs linked with the NetSolve library, Matlab sessions that use the NetSolve `.mex` file, Mathematica sessions that use NetSolve via `MATHLINK`, Java applications or applets that call the NetSolve Java class library. Additionally, NetSolve provides a Java GUI so that users can directly interact with NetSolve without writing any program (e.g. from the Web).
- Agents are C programs that run as stand-alone daemons.
- Servers may be registered with one or more agents, and may then be annotated with the software services that they can perform. Servers may register or unregister at any time.

Currently, the agents and servers run on all variety of Unix machines, and clients may run on Unix and Windows 95/NT.

We have developed a suite of server routines that are very easy to install as part of any uniprocessor server. These correspond to standard computations from different fields, such as linear algebra, FFTs, optimization, curve fitting, iterative methods, etc. We have also developed servers that run any of the ScaLAPACK [6] routines on any parallel processing environment that supports PVM or MPI. It is a straightforward task to embed other software into a server. Finally, we have also developed servers composed of a collection of workstations managed by Condor [25]. These are described below in section 5.1.

Though NetSolve is still a prototype, its potential for providing easy access to large computational resources has already attracted early users within the scientific community. Below are two illustrations of such early deployments:

- **Image processing with NetSolve:** Researchers at the ICG institute at Graz University of Technology, Austria, currently use NetSolve to make sophisticated image processing functions available for remote execution to a large community of users (See [9]).
- **Neuro-science with NetSolve:** At the Computational Neurobiology Lab of the Salk Institute, the NetSolve prototype is being integrated into MCell, which is a software that performs three dimensional Monte Carlo simulations of cellular microphysiology for biologists and biochemists. MCell simulations require solutions to extremely large data parallel problems in multidimensional parameter spaces. NetSolve is ideal for this kind of problem. Because of NetSolve's ability to serve up a vast array of computational resources, including the 256 node Cray T3E located at the San Diego Supercomputer Center, integration of MCell with NetSolve is expected to enable MCell simulations that previously took two weeks to run to finish in an afternoon.

3 Transparent Fault-Tolerance and Load Balancing

As mentioned earlier, the computational resources on a grid may be vast, but due to distributed ownership and the nature of loosely connected systems, their availability may be quite variable.

Privately owned workstations may be available for computation one minute, then revoked by the owner the next. Shared machines may exhibit variable load, or may have set schedules for exclusive use. Under these conditions, issues of fault-tolerance and load-balancing become especially important. There has been a vast amount of research on embedding fault-tolerance and load-balancing into parallel and distributed computing platforms. Approaches that have been explored include user-transparent checkpointing and migration libraries (e.g. [10, 12, 38, 25]), programming paradigms that facilitate the task of fault-tolerance or load balancing (e.g. [27, 35]), or modified algorithms for performing certain specific computations in a fault-tolerant manner (e.g. [7, 20, 30]). While the effectiveness of these techniques has been demonstrated experimentally, none of them have made a large impact on the scientific computing community because using them requires far too much end user involvement.

All the types of fault-tolerance and load balancing listed above fit seamlessly into NetSolve's structure. We divide fault-tolerance and load balancing into **inter-server** and **intra-server** categories. For example, the agent may keep tabs on the progress of a server. If the server is not progressing at a reasonable pace, then the agent (or client) may instruct another server to perform the computation. With a little more effort, the agent may be able to instruct another server to actually continue the computation using the current state of the old server. These are examples of inter-server techniques for fault-tolerance and load balancing. Intra-server techniques are possible as well. For instance, servers of all kinds may employ transparent or non-transparent techniques to render themselves resilient to faults. Servers composed of multiple machines may implement local load balancing to make themselves more efficient.

The important point about the use of all such techniques within the NetSolve framework is that they are totally transparent to the end user. Thus, by employing NetSolve, a user can reap the benefits of research on fault-tolerance and load balancing without actually incorporating it into their code. In fact they may reap these benefits without even knowing it! In this way, NetSolve can make a major contribution in the area of *deploying* support for fault-tolerance.

4 Inter-server Paradigms

If a server fails or is not progressing fast enough, then it is logical for the agent to detect this and move the computation to another server. In this section we discuss different levels of inter-server fault-tolerance that will be explore, and how they fit into the NetSolve framework.

4.1 The Current Approach

The first level of inter-server fault-tolerance is implemented in the current NetSolve prototype. The agents periodically collect information about the servers and their status. If a server that is currently executing a computation becomes unavailable, then the agent selects a new server to take over the computation. Currently, this involves simply having the client contact the new server, and start the computation anew. The agents use a variety of mechanisms for determining server availability, including standard Unix calls such as `uptime` to determine server load. We have also incorporated the Globus Heart Beat Monitor [17] into the agents to perform reliable failure detection. We plan to integrate the efforts of the Network Weather Service [42] into the NetSolve agents as well.

4.2 Integrated Storage Services

A simple way to improve upon the current model is to have the new server query the old server, and if the old server is alive but overloaded, then the old server can send the state of the computation to the new server, which resumes processing from that point. This can be done in a generic manner as in Condor [25], or it can be done in a more application-specific manner, as in the PUL library [35].

This simple method is a good first step, and indeed provides more fault-tolerance than most currently used software for scientific computation. However, it is limited by the fact that the failed server must somehow provide its state to the new server. To address this problem, we introduce a new class of server to the NetSolve framework, called *storage servers.* At a high level, storage servers, just like regular servers, are managed by the agents. However, their job is not to compute, but to hold data. Storage servers will store checkpoints of the computations running on other servers. Very simply, when a computation server decides it should save its state, it selects a storage server with the help of a NetSolve agent, and sends its checkpoint to the storage server. The storage server may store the data in any way that it chooses - in physical memory or on secondary storage. If the storage server needs to make some guarantees about availability, it may replicate the data. Note that the storage server does not need to be concerned with the format of the checkpoint data or with what the data means. If an agent detects that a server has failed or is not performing a computation quickly enough, then it selects a new server to resume the computation from the most recent checkpoint in the associated storage server. This removes any dependency on the old server, which may not be able to respond to queries from the new server. When the computation is finished, the server or the agent may notify the relevant storage servers so that they may discard their checkpoints.

Thus, the concept of inter-server fault-tolerance fits nicely within the framework of NetSolve. One may view storage servers as computation servers that merely hold data and do not perform computations. The management of the all servers is still performed by the agents, which, as in the non-fault-tolerant case, do not concern themselves with the actual computations or checkpoint files. Instead they provide a repository for information about servers and a broker for the resources of servers. As discussed below in Section 5.3, this design of inter-service fault-tolerance allows for interesting interactions of inter and intra-service fault-tolerance.

A first prototype of storage servers has already be developed and will be distributed with the next NetSolve software release.

5 Intra-server Fault-tolerance

NetSolve's logical separation of clients and servers and its dynamic access to heterogeneous resources make it an ideal test-bed that can use real applications to compare the performance of different techniques for fault-tolerance and load balancing. In this section we present different techniques for fault-tolerance and load-balancing that we are considering implementing within NetSolve. These are termed *intra-server* techniques because they are implemented within one server. We also discuss the mixing of intra-server and inter-server techniques for fault-tolerance.

5.1 The Current Approach

Currently, NetSolve allows a pool of workstations managed by Condor [25] to act as a NetSolve server. Condor provides transparent checkpointing and restart so that computations may be moved from loaded/failed machines to idle/available machines. As such, it is able to make use of idle cycles in a shared workstation environment. Employing Condor is a significant first step towards intra-server fault-tolerance. However, it has the following limitations:

- Condor can only migrate jobs between machines of the same architecture.
- Condor can only migrate jobs within its server.
- Condor only works with serial (non-parallel) programs.
- Condor never completely migrates off of its original "host" machine. In particular, system calls are always forwarded to and executed by the original host machine.

In the following sections, we describe the approaches that we take to fault-tolerance and load balancing that attempt to alleviate these limitations.

5.2 Coordinated Checkpointing and Rollback Recovery

The most straightforward scheme for providing fault-tolerance for parallel programs is to employ coordinated checkpointing and rollback recovery. At set intervals (determined by parameters such as program size, storage speed, etc. [40]), the processors involved in a computation save their computation states to stable storage. This is called a coordinated checkpoint. Following a failure of any or all components, an equal number of processors use the states on stable storage to restart the computation from the point of the checkpoint. Several checkpointing libraries have been written for performing coordinated checkpointing on various parallel computing platforms. For example, MIST [10] and CoCheck [38] provide transparent checkpointing for PVM and MPI programs on networks of workstations, CLIP [12] provides semi-transparent checkpointing for Intel Paragon programs, and the PUL-library [35] provides non-transparent checkpointing for a certain class of parallel applications. Here, transparency refers to the amount of programmer involvement necessary to get checkpointing to work.

As a first step, we will add non-transparent coordinated checkpointing as a basic method for intra-server fault-tolerance. The reason for making it non-transparent is because doing so facilitates making the checkpoints platform-independent, and therefore the processor configuration following a failure does not need to be constrained by the configuration at the time of checkpointing. For example, the program may restart on a different number of processors or on processors of differing architectures.

We have already developed a prototype of a NetSolve server for the ScaLAPACK package that performs portable checkpointing and rollback. We will then attack other parallel server packages. The obvious next step is to employ the storage servers described in Section 4.2 to store the checkpoints so that the applications may be restored on any server, not just the server with failed processors.

5.3 Diskless Checkpointing Techniques

One way to improve the performance of coordinated checkpointing is to remove stable storage from the protocol. This is called diskless checkpointing [30]. Here, the coordinated checkpoint is stored in the memory of the application processors and combined with error correcting coding and some extra checkpointing processors so that the loss of a limited number of processors may be tolerated. These techniques have already been embedded into some of the ScaLAPACK routines and have demonstrated excellent performance [30]. There are three interesting research directions for diskless checkpointing in NetSolve.

First, we will incorporate the above code into the ScaLAPACK servers. Note that the structure of NetSolve allows us to embed such application-specific fault-tolerant techniques into the servers so that users may take advantage of them without any knowledge of the underlying fault-tolerant concepts. This would be a breakthrough in bringing complex research concepts into more mainstream use.

Second, there are novel ways in which we can leverage diskless checkpointing to mix intra and inter-server fault-tolerance. For example, instead of storing the error correcting coding in checkpointing processors within a server, we may store them at the storage servers, which may facilitate the migration of the application to another server, or perhaps free the server from having to allocate checkpointing processors altogether.

Finally, by incorporating diskless checkpointing into the server applications, we may discover interesting new fault-tolerant paradigms. This was done with the right-looking matrix factorizations of ScaLAPACK, which did not perform as well as some of the other ScaLAPACK algorithms when diskless checkpointing was employed. The result was a new paradigm for fault-tolerant matrix algorithms called checksum and reverse computation [24]. We anticipate that there will many more opportunities to explore new fault-tolerant paradigms within individual server applications. These may be based on some combination of checkpointing, algorithm-based fault tolerance [20] or backward assertion [7] techniques.

5.4 Distributed Shared Memory and Other Programming Paradigms

There are some problems that lack the regular structure of (for example) dense matrix operations, and map more naturally to the shared memory paradigm of parallel programming [1, 23]. Since NetSolve does not constrain the server, such programs may employ shared memory solutions in the servers without the end user being aware of it. The use of shared memory has implications for fault-tolerance and load balancing. In particular, checkpointing strategies may take advantage of the replication and redundancy inherent in shared memory systems to achieve better performance. This has been explored by researchers for transparent runtime libraries that implement distributed shared memory [8, 11, 22, 39], and for programs that make explicit use of data structures with shared memory semantics [5, 33].

Relatedly, there has been research on fault-tolerant shared tuple spaces [3] and other models of parallel programming such as farming [36], master-slave [4], and coarse-grained dataflow [14] that are more restrictive than general message passing, and facilitate the addition of fault-tolerance and computation migration. A great strength of NetSolve is that if a server functionality maps well to one particular programming paradigm, then the server may implement that paradigm, and thereby allow that functionality to have fault-tolerance and load balancing embedded in it rather seamlessly.

We anticipate implementing a few fault-tolerant servers with some of these programming paradigms. There are three obvious benefits to such implementations. First, they will add to the number of deployable fault-tolerant servers available with NetSolve. Second, they will enable us to compare the performance and tradeoffs of using traditional fault-tolerant strategies, (such as coordinated checkpointing on message-passing architectures) and using fault-tolerant strategies that make use of a specialized programming paradigm. Finally, these implementations may serve as examples for other researchers to implement their own fault-tolerant servers for different applications.

Extending these ideas to inter-server fault-tolerance is also an interesting avenue for research. For example, storage servers may be employed to mirror the state of shared memory, or perhaps to back up a shared tuple space. Similarly, storage servers may implement a remote memory service [16], which may improve the performance of inter-server checkpointing and migration.

5.5 Migration Issues

There are three other research issues in migration that we will explore in the context of NetSolve. First is architecture independent checkpointing. Above, we describe a scheme where checkpointing is embedded directly into an application, giving it the ability to checkpoint and restart on differing computational platforms. There are other ways of duplicating this functionality, such as using a preprocessor to automate the task of embedding type program information into the checkpoint [32] employing restricted languages that may embed architecture independent checkpointing into the compiler and runtime system [37] or employing special checkpointable data structures [5]. While prototype implementations of the above techniques have been promising, the structure of NetSolve allows us to test their use with real applications, compare their performance and deploy them. We anticipate that by exploring these concepts in the framework of NetSolve, we will gain a better understanding of how to write easy-to-deploy and efficient code for architecture independent checkpointing.

Second, the use of privately owned resources as NetSolve servers is an important issue. Specifically, the CPU capacity of most workstations is rarely utilized by their owners. Separate studies have shown that if a computation may be structured so that it only uses idle cycles of privately owned machines, an enormous amount of computation may be performed [13, 26, 28]. The issue of using privately owned resources has been touched in NetSolve upon by the Condor servers, but ideally the brokering of such resources should be performed by the NetSolve agents, and then the migration co-managed by the servers and the agents. We plan to leverage off the similar efforts of CARMI [31] and Globus [18] so that NetSolve can tap into the processing capacity of privately owned workstations.

Finally, The function of the NetSolve agent is closely related to that of a class of systems known as Object Request Brokers, most notably Object Management Group's CORBA [29]. These brokers allow a server to register an interface and facilitate clients in finding and attaching to these interfaces. The objective of such systems are to create an infrastructure for the construction of distributed enterprise level applications.

Because it is designed as a near-universal architecture, CORBA suffers from a combination of generality and structure. Clients and servers can be implemented in any programming language, but all interfaces are defined using object-oriented constructs. Since even OO languages have diverse type systems, CORBA defines its own in the form of the Interface Definition Language (IDL), and requires every language to map IDL to its own constructs. As a result, the CORBA user must make use of extensive mapping tools and invest significant effort just to connect to the system.

The Java world seeks to overcome the generality of CORBA by creating single-language solutions. These solutions make use of the fact that client and server are both implemented in Java to eliminate the need for an IDL. They make use of remote call mechanisms such as Sun's Remote Method Invocation (RMI) to pass complex structures transparently between client and server. Unfortunately, Java is not currently an appropriate standard for world's high performance software.

NetSolve takes a very practical approach: the interface is modeled after the Fortran type system, and mappings are defined for other programming languages. Because Fortran's type system is so simple, these mappings are generally straightforward. Because so much of the software interfacing to Netsolve is written in C and Fortran, there are few software barriers in porting a large body of single-processor code to Netsolve. While this framework does not have the potential to ease the task of brokering complex objects as in CORBA or HORB, it does not enforce a programming paradigm (e.g. object-oriented programming) that would get in the way of porting the large body of code that is implemented in C and Fortran. Simply put, NetSolve is a tool that works without getting in the way. It is an open research area to investigate how more complex brokering paradigms fit into the NetSolve framework.

5.6 Related Grid-oriented Software

As grid-based computing has begun to emerge, approaches to grid-oriented software that are both similar to and different from NetSolve's are under development. Ninf [34], is a project that bears similarities to NetSolve, and a *bridge* has been developed so that NetSolve and Ninf can share resources and clients. The Network-Enabled Optimization Server (NEOS) [15] is in some ways comparable to NetSolve. The differences, however, are significant. First, NEOS addresses a specific field of computational science (optimization) whereas NetSolve, can integrate virtually any processing of user data. Second, NetSolve's software architecture allows it to be deployed on any scale with great flexibility as opposed to NEOS which is centralized at the Argonne National Laboratory. Third, NetSolve provides several more user interfaces than NEOS, including a Matlab and two Java interfaces.

Other than Condor, the two leading middleware systems for creating computational grids are Globus [18] and Legion [19]. They address somewhat different audiences. NetSolve targets any scientist or engineer and provides them with a high level service. By contrast, both Globus and Legion are built on their own lower level directory and communication services, and therefore are significantly more elaborate to deploy. However, both Globus and Legion can be accommodated by the NetSolve model (as agents/servers). As they gain maturity we intend to review the current NetSolve design/implementation and to gradually integrate new components from these systems. As mentioned in Section 4.1, we have already integrated the Globus Heart Beat Monitor into NetSolve. Analogous investigations of the integration of NetSolve with Legion's approach to grid computing are also underway.

6 Conclusion

NetSolve is an environment for networked computing whose goal is to deliver the power of computational grid environments to users who have need of processing power, but are not expert computer scientists. It achieves this goal with its three-part client-agent-server architecture. In order to deliver the full capacity of grid resources, NetSolve must deal with the potential for these resources to be unstable, which means that fault-tolerance and/or computation migration must be employed. We have described how the current version of NetSolve addresses these issues, and how NetSolve will evolve to address them more completely. The most significant impact of this research is in NetSolve's *deployability* of techniques for fault tolerance and migration. Specifically, by incorporating primitives for inter-server fault-tolerance within the NetSolve model, and by developing fault-tolerant software for NetSolve servers, we can deliver fault-tolerance and migration to end-users without any burden on the end user.

References

[1] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. TreadMarks: Shared Memory Computing on Networks of Workstations. *IEEE Computer*, 29(2):18–28, February 1996.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide, Second Edition.* SIAM, Philadelphia, PA, 1995.

[3] D. E. Bakken and R. D. Schilchting. Supporting Fault-Tolerant Parallel Programming in Linda. *IEEE Transactions on Parallel and Distributed Systems*, 6(3):287–302, March 1995.

[4] A. Baratloo, P. Dasgupta, and Z. M. Kedem. CALYPSO: A Novel Software System for Fault-Tolerant Parallel Processing on Distributed Platforms. In *4th IEEE International Symposium on High Performance Distributed Computing*, August 1995.

[5] A. Beguelin, E. Seligman, and P. Stephan. Application Level Fault Tolerance in Heterogeneous Networks of Workstations. *Journal of Parallel and Distributed Computing*, September 1997.

[6] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[7] D. Boley, G. H. Golub, S. Makar, N. Saxena, and E. J. McCluskey. Floating Point Fault Tolerance with Backward Error Assertions. *IEEE Transactions on Computers*, 44(2), February 1995.

[8] G. Cabillic, G. Muller, and I. Puaut. The Performance of Consistent Checkpointing in Distributed Shared Memory Systems. In *Proceedings of the 1995 European Intel Supercomputer Users' Group Meeting*, 1995.

[9] H. Casanova and J. Dongarra. NetSolve's Network Enabled Server: Examples and Applications. *IEEE Computational Science & Engineering*, to appear.

[10] J. Casas, D. L. Clark, P. S. Galbiati, R. Konuru, S. W. Otto, R. M. Prouty, and J. Walpole. MIST: PVM with transparent migration and checkpointing. In *3rd Annual PVM Users' Group Meeting*, Pittsburgh, PA, May 1995.

[11] M. Castro, P. Guedes, M. Sequeira, and M. Costa. A checkpoint protocol for an entry consistent shared memory system. In *Thirteenth ACM Symposium on Principles of Distributed Computing*, Los Angeles, CA, August 1994.

[12] Y. Chen, J. S. Plank, and K. Li. CLIP: A Checkpointing Tool for Message-Passing Parallel Programs. In *SC97: High Performance Networking and Computing*, San Jose, November 1997.

[13] P. E. Chung, Y.Huang, S. Yajnik, G. Fowler, K. P. Vo, and Y. M. Wang. Checkpointing in CosMiC: a user-level process migration environment. In *Pacific Rim International Symposium on Fault-Tolerant Systems*, December 1997.

[14] D. Cummings and L. Alkalaj. Checkpoint/Rollback in a Distributed System Using Coarse-Grained Dataflow. In *24th International Symposium on Fault-Tolerant Computing*, pages 424–433, Austin, TX, June 1994.

[15] J. Czyzyk, M. Mesnier, and J. Moré. NEOS : The Network-Enabled Optimization System. Technical Report MCS-P615-1096, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.

[16] M. J. Feeley, W. E. Morgan, F. H. Pighin, A. R. Karlin, and H. M. Levy. Implementing Global Memory Management in a Workstation Cluster. In *15th Symposium on Operating Systems Principles*, pages 201–212. ACM, December 1995.

[17] I. Foster, C. Kesselman, C. Lee, G. von Laszewski, and P. Stelling. A Fault Detection Service for Wide Area Distributed Computations. In *Proc. of the High Performance Distributed Computing Conference*, to appear.

[18] I. Foster and K Kesselman. Globus: A Metacomputing Infrastructure Toolkit. In *Proc. Workshop on Environments and Tools*. SIAM, to appear.

[19] A. Grimshaw, W. Wulf, J. French, A. Weaver, and P. Jr. Reynolds. A Synopsis of the Legion Project. Technical Report CS-94-20, Department of Computer Science, University of Virginia, 1994.

[20] K-H. Huang and J. A. Abraham. Algorithm-Based Fault Tolerance for Matrix Operations. *IEEE Transactions on Computers*, C-33(6):518–528, June 1984.

[21] The Math Works Inc. *MATLAB Reference Guide*. 1992.

[22] G. Janakiraman and Y. Tamir. Coordinated Checkpointing-Rollback Error Recovery for Distributed Shared Memory Multicomputers. In *13th Symposium on Reliable Distributed Systems*, pages 42–51, October 1994.

[23] K. L. Johnson, M. F. Kaashoek, and D. A. Wallach. CRL: High-Performance All-Software Distributed Shared Memory. In *15th Symposium on Operating Systems Principles*, pages 213–228. ACM, December 1995.

[24] Y. Kim, J. S. Plank, and J. Dongarra. Fault Tolerant Matrix Operations using Checksum and Reverse Computation. In *6th Symposium on the Fontiers of Massively Parallel Computation*, October 1996.

[25] M. Litzkow and M. Livny. Experience with the Condor Distributed Batch System. In *Proc. of IEEE Workshop on Experimental Distributed Systems*. Department of Computer Science, University of Winsconsin, Madison, 1990.

[26] M. W. Mutka and M. Livny. The available capacity of a privately owned workstation environment. *Perfomance Evaluation*, August 1991.

[27] V. K. Naik, S. P. Midkiff, and J. E. Moreira. A Checkpointing Strategy for Scalable Recovery on Distributed Parallel Systems. In *SC97: High Performance Networking and Computing*, San Jose, November 1997.

[28] D. A. Nichols. Using Idle Workstations in a Shared Computing Environment. *Operating Systems Review: Proceedings of SOSP-11*, 21(5):5–12, November 1987.

[29] R. Orfali and D. Harkey. *Client/Server Programming with Java and CORBA*. John Wiley & Sons, Inc, 1997.

[30] J. S. Plank, Y. Kim, and J. Dongarra. Fault Tolerant Matrix Operations for Networks of Workstations Using Diskless Checkpointing. *Journal of Parallel and Distributed Computing*, 43:125–138, September 1997.

[31] J. Pruyne and M. Livny. Parallel Processing on Dynamic Resources with CARMI. In *First IPPS Workshop on Job Scheduling Strategies for Parallel Processing*, April 1995.

[32] B. Ramkumar and V. Strumpen. Portable Checkpointing and Recovery in Heterogeneous Environments. In *27th International Symposium on Fault-Tolerant Computing*, 1997.

[33] D. J. Scales and M. S. Lam. Transparent Fault Tolerance for Parallel Applications on Networks of Workstations. In *Usenix 1996 Technical Conference on UNIX and Advanced Computing Systems*, San Diego, January 1996.

[34] S. Sekiguchi, M. Sato, H. Nakada, S. Matsuoka, and U. Nagashima. Ninf : Network based Information Library for Globally High Performance Computing. In *Proc. of Parallel Object-Oriented Methods and Applications (POOMA), Santa Fe*, 1996.

[35] L. M. Silva, J. G. Silva, S. Chapple, and L. Clarke. Portable Checkpointing and Recovery. In *Proceedings of the HPDC-4, High-Performance Distributed Computing*, pages 188–195, Washington, DC, August 1995.

[36] L. M. Silva, B. Veer, and J. G. Silva. Checkpointing SPMD Applications on Transputer Networks. In *Scalable High Performance Computing Conference*, pages 694–701, Knoxville, TN, May 1994.

[37] B. Steensgaard and E. Jul. Object and native code thread mobility among heterogeneous computers. In *15th Symposium on Operating Systems Principles*, pages 68–78. ACM, December 1995.

[38] G. Stellner. CoCheck: Checkpointing and Process Migration for MPI. In *10th International Parallel Processing Symposium*, April 1996.

[39] G. Suri, B. Janssens, and W. K. Fuchs. Reduced Overhead Logging for Rollback Recovery in Distributed Shared Memory. In *24th International Symposium on Fault-Tolerant Computing*, pages 279–288, June 1994.

[40] N. H. Vaidya. Impact of Checkpoint Latency on Overhead Ratio of a Checkpointing Scheme. *IEEE Transactions on Computers*, 46(8):942–947, August 1997.

[41] S. Wolfram. *The Mathematica Book, Third Edition*. Wolfram Median, Inc. and Cambridge University Press, 1996.

[42] R. Wolski. Dynamically forecasting network performance to support dynamic scheduling using the Network Weather Service. In *6th High-Performance Distributed Computing Conference*, August 1997.