Design, Implementation, and Performance of Checkpointing in NetSolve

Adnan Agbaria

Department of Computer Science
Technion - Israel Institute of Technology
Haifa 32000, Israel
adnan@cs.technion.ac.il

James S. Plank

Department of Computer Science
University of Tennessee
Knoxville, TN 37996
plank@cs.utk.edu

Technical Report UT-CS-99-433
University of Tennessee
November, 1999
http://www.cs.utk.edu/~plank/plank/papers/UT-CS-99-433.html

Submitted for publication. See the above URL for up-to-date publication information.

Abstract

While a variety of checkpointing techniques and systems have been documented for long-running programs, they are typically not available for programmers that are non systems experts. This paper details a project that integrates three technologies, NetSolve, Starfish, and IBP, for the seamless integration of fault-tolerance into long-running applications. We discuss the design and implementation of this project, and present performance results executing on both local and wide-area networks.

1 Introduction

Checkpointing and rollback recovery is a well-studied research area for enabling long-running applications to be fault-tolerant. Many basic checkpointing algorithms [EAWJ99, MS99] and optimization techniques [Pla99] have been developed for uniprocessor and parallel computing systems, and several checkpointing libraries and systems have been implemented [EZ92, HKW95, PBKL95, TL95, WHV$^+$95, Ste96, CPL97, RS97, AF99]. However, for the typical scientific user, actually using a checkpointing system is a difficult task. All systems require the user to

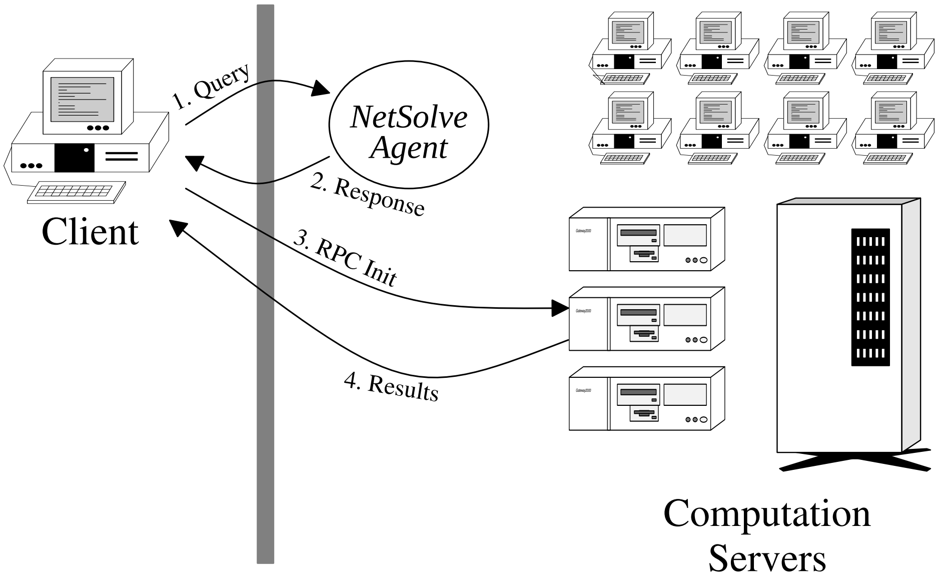

Figure 1: The structure of NetSolve applications

port a library and recompile or relink their code subject to a number of restrictions imposed by the library. These restrictions range from strong typing of the source code [RS97] to restricted file I/O [PBKL95, CPL97] to static linking of runtime libraries [AF99], to restricted communication patterns [CPL97]. One restriction shared by all checkpointers is that no connections to the outside world may be open while checkpointing is underway.

Because of all of these factors, few scientific users actually employ checkpointing in their applications. This paper describes a research project whose goal is to embed checkpointing seamlessly into long-running applications for scientific programmers. To achieve this goal, we combine three software systems, NetSolve [CD97a], Starfish [AF99], and IBP [PBE$^+$99]. In this experience report, we describe each software piece and how the pieces are integrated, focusing on the important design decisions. These are:

- A user interface with few complexities.
- An efficient checkpoint library that is fairly simple to embed into server code, and whose restrictions do not limit the user's application.
- A checkpointing storage substrate that facilitates restart and migration across administrative domains, and automatic garbage collection.

We close with a few performance studies in a variety of local and wide area settings.

2 The Components

The system is based on three components: NetSolve, Starfish, and IBP. We first describe these components.

2.1 NetSolve [CD97a]

NetSolve is a brokered remote procedure call (RPC) environment as depicted in Figure 1. The user is termed a *client*, and is typically executing code on a PC or laptop. When the client wishes to perform a computationally complex task, he or she makes a NetSolve client call, specifying the name of the task, plus the arguments. The NetSolve client software (linked to the client in the form of a library) manages the completion of this task, which we will refer to as a “service.”

First an *agent* is contacted with a query (step 1), specifying the service name and the size of the arguments. The agent maintains information on a collection of computational servers, which may be uniprocessors, multiprocessors, massively parallel machines, Condor workstation pools [TL95], etc. This information consists of machine parameters (speed, memory, available software), plus current load information. The agent returns an ordered list of candidate servers to the client (step 2), who then picks a server (typically the first on the list) and initiates a RPC to that server (step 3). The server performs the service, and completes the RPC, returning the results to the client (step 4).

Although not depicted in Figure 1, there may be multiple agents managing overlapping server pools. Additionally, servers may span multiple geographic and administrative domains, of which the clients may or may not be a part. One of NetSolve’s strengths is the wide variety of clients that it supports. The NetSolve client code may be linked with C, C++ and Fortran, running on both Unix and Windows platforms. Additionally, it may be used from within the popular scientific toolkits Matlab and Mathematica, and from Microsoft Excel. The NetSolve release contains server software for dense and sparse linear algebra routines and other commonly-used scientific codes (e.g. ARPACK, FitPack, ItPack, MinPack, FFTPACK, LAPACK, QMR, etc.). Users may configure servers to run custom code as well with the aid of some Java tools [CD97b].

2.2 Starfish [AF99]

Starfish is a transparent checkpointing library originally developed to embed fault-tolerance and migration into MPI applications. The Starfish checkpointing mechanism is a standard core dump mechanism that has served as the basis for many checkpointers (see papers by Tannenbaum [TL95] and Plank [PBKL95] for throrough discussions of these types of checkpointers). Starfish checkpoints periodically, triggering checkpoints by timer interrupts.

This checkpointer is a library to be linked with Solaris-based programs. No recompilation of any source code is required. Starfish implements the copy-on-write optimization [EJZ92, PBKL95] so that the act of checkpointing may be overlapped with the execution of server code. Like most checkpointers, Starfish imposes restrictions on file I/O, requires static linking of shared libraries, and prohibits the use of interprocess communication.

2.3 IBP [PBE$^+$99]

The *Internet Backplane Protocol (IBP)* is a mechanism for managing storage on the wide area. IBP servers are daemons that provide local storage (disk, tape and physical memory) to remote clients that link the IBP client library. IBP is useful for checkpointing applications because it allows programs to store their checkpoints into a remote storage entity, perhaps one in a different administrative domain. Therefore if the machine executing the program fails and remains inoperative for a long period of time, the program may be restored on a separate machine, again perhaps in a different administrative domain.

IBP has two features that enable it to serve storage on the wide area as a networking resource:

- **There are no user-defined names.** IBP clients allocate storage, and if the allocation is successful, then it returns three *capabilities* to the client — one each for reading, writing, and management. These capabilities are text strings, and may be viewed as server-defined names for the storage. The elimination of user-defined names facilitates scalability, since no global namespace needs to be maintained. To make an IBP client call for reading, writing or management, the client must present the server with the proper capability.

- **Storage may be constrained to be *volatile* or *time-limited*.** An important issue when serving local storage to remote clients is being able to reclaim the storage. IBP servers may be configured so that the storage allocated to IBP clients is *volatile*, meaning it can go away at any time, or *time-limited*, meaning that it goes away after a specified time period.

The transient nature of IBP storage leads us to refer to the units of IBP storage as buffers.

3 Putting it all Together

The structure of NetSolve with checkpointing is depicted in Figure 2. In a nutshell, the NetSolve servers are linked with Starfish and store their checkpoints in IBP buffers. When a server fails, the computation is rolled back to the most recent checkpoint and restored on a new server. The client receives results from whichever server completes the computation. In such a way, the client ends up executing fault-tolerant and migratable code by simply linking with the NetSolve client library.

There is much more detail in the implementation. We first describe the exact client-agent-server interaction. The agent must be aware of server architectures, and whether the server code for a particular computational

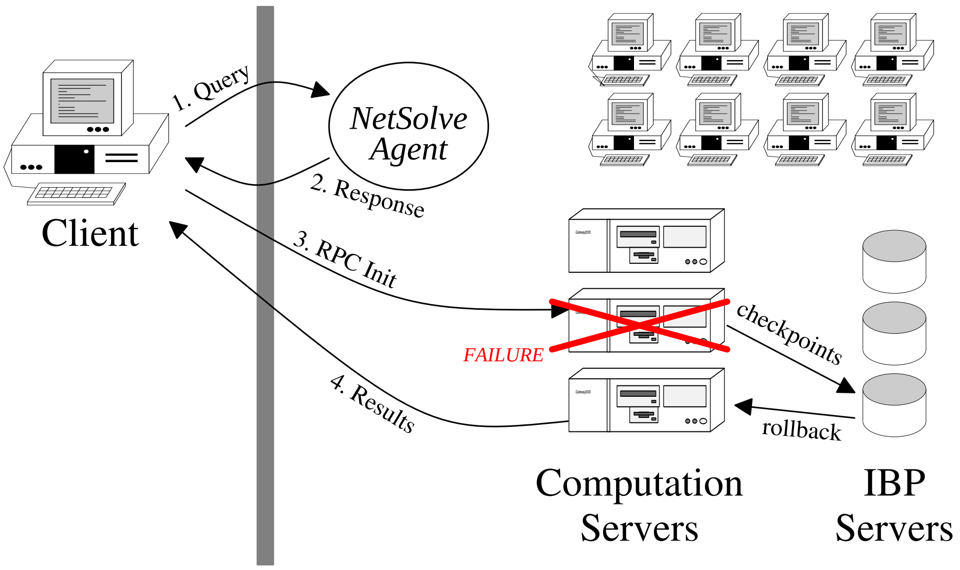

Figure 2: NetSolve with checkpointing

service has been linked with the Stafish library. This information is returned to the client as part of the response to the client's initial query. If the client selects a server that is enabled with checkpointing, then as part of the RPC initiation, the server allocates an IBP buffer on a nearby server. We call this buffer the "naming buffer." The capabilities of the naming buffer are returned to the client.

When the server initiates a checkpoint, it first allocates an IBP buffer for the checkpoint. This is a time-limited allocation for some fixed period of time greater than the checkpoint interval. When the checkpoint is stored in the IBP buffer, the capabilities of this buffer are stored in the naming buffer, and if necessary, the previous checkpoint buffer is deleted.

The original NetSolve distribution has failure detection and primitive fault-tolerance. Server failures (which may be defined as excessive load) are detected by the NetSolve clients and/or the agent. When a failure is detected, the client is contacted and instructed to restart the service on a new server. With checkpointing, the client can select a server with the same architecture as the failed server, which can then roll the computation back to the most recent checkpoint. The client presents this new server with the capabilities of the naming buffer, which allow the new server to find the checkpoint buffer and restart the computation. Obviously, this new server may continue checkpointing as well. If there is no server that can restart the computation from the checkpoint, then the client selects the best available server to restart the computation from the beginning.

When the computation completes, the server returns the results to the client and deletes all IBP buffers. Note, however, that if other errors occur, such as NetSolve agent failure, client failure, or NetSolve system shutdown, the time-limited nature of the IBP buffer allocation will make sure that spare checkpoint files are eventually deleted.

As stated above in section 2.2, Starfish places restrictions on the programs that it checkpoints. The only restriction that is a potential problem for NetSolve server code is the prohibition on external connections. While performing a computation, a server only needs to have an open connection to the client when performing the initial RPC interactions and when delivering the results. Thus, Starfish does not start checkpointing until the initial RPC interations are over, and it stops once the server starts delivering results. Typically, NetSolve server codes perform only basic file I/O operations, which are checkpointable by standard means [PBKL95].

The selection of checkpointing interval and checkpointing IBP servers is performed by the agent. The optimal checkpoint interval may be approximated by a simple function of checkpoint overhead and failure rate [Vai97, PE98], which are both parameters that the agent can estimate. IBP server proximity currently estimated using static metrics. A test implementation of NetSolve integrates the Network Weather Service [WSH99] into the NetSolve system so that the agent can make more accurate predictions of computation server performance and IBP server proximity.

4 Benefits of This Architecture

There are several benefits that this design has in terms of performance, functionality and deployability:

- **The user is insulated from checkpointing details.** In the best case, the user is employing NetSolve to perform common computations such as dense linear algebra. In this case, the NetSolve server setup is trivial, and the user can unknowingly receive the benefits of remote computation and checkpointing even while using Excel on a Windows-based laptop. This is a level of deployability that is typically unheard of in scientific programming.

- **The user's program can have outside connections.** All checkpointing systems restrict connections outside the scope of the programming environment. In other words, while checkpointing systems typically work when all processors are part of the same programming system (for example through the use of PVM or MPI [CCG$^+$95, Ste96, AF99]), they only allow programs to interact with the outside world by checkpointing (or logging) before *each* interaction [EZ94, EAWJ99]. With NetSolve, the client may initiate a service while maintaining other external connections. This service can checkpoint, fail, rollback, and continue to operate correctly irrespective of the state of the client and its connections to other processing elements. This even works if the client starts the service asynchronously (i.e. in the background while it performs other tasks). Thus, NetSolve's restricted programming model achieves a clean separation of client and server that allows the server to checkpoint while the client does other things.

- **Migration can occur across the wide area.** NetSolve and IBP both manage resources from different administrative domains, serving cycles and storage to potentially unrelated users and applications. With checkpointing to IBP, it is possible to migrate these services from one domain to another, so long as the server machine architectures are identical.

- **It will work in a lent-resource environment.** Similarly to the above, NetSolve and IBP are both able to manage spare resources (computation and storage) that have limits on their usage. In particular, processors may be revoked due to ownership, and storage may impose time limits on allocation. The inclusion of checkpointing into the NetSolve system means that these resources may be employed by remote computations. This funcationality is similar to that provided by the Condor project [TL95].

- **Storage ownership is separated from the computation.** Pruyne and Livny have noted that strategic placement of checkpoints at locations external to the computation processors can improve performance [PL96]. The use of IBP in NetSolve is identical to the use of checkpointing servers in [PL96] and should improve performance similarly.

5 Performance Case Studies

We briefly detail three performance case studies. In each of these, we have a NetSolve client running Matlab, a NetSolve agent, two NetSolve servers and one IBP server all running on different machines. The Matlab client makes a NetSolve call to the `dmatmul` service (matrix multiplication), which gets serviced by one of the NetSolve servers. The server checkpoints to the IBP server, and either it completes without failure, or it fails. When the failure is detected, the second server takes over the service, reading from the checkpoint, and completes the service.

We report results from three separate computing environments: CLUSTER, LOCAL and WIDE. CLUSTER is a tightly-coupled cluster computing environment. The machines are all dual-processor Sun UltraSPARC-2's with 256 Mbytes of RAM, connected by a 155 Mbps ATM network. LOCAL is a department-wide environment, where the NetSolve client and agent are Sun UltraSPARC-1's, and the other machines are lower-end SparcStation-5's. All machines are connected by the Computer Science department's backbone network at the University of Tennessee. Finally WIDE is a wide-area, multi-institutional environment where the client, agent and IBP server are running on UltraSPARC-1's at Tennessee, while the NetSolve servers are running on two UltraSPARC-1's at Princeton University. Communication between the two institutions is done over the standard Internet. In the

CLUSTER test, the machines are dedicated to the experiment. In all other tests, the machines are undedicated.

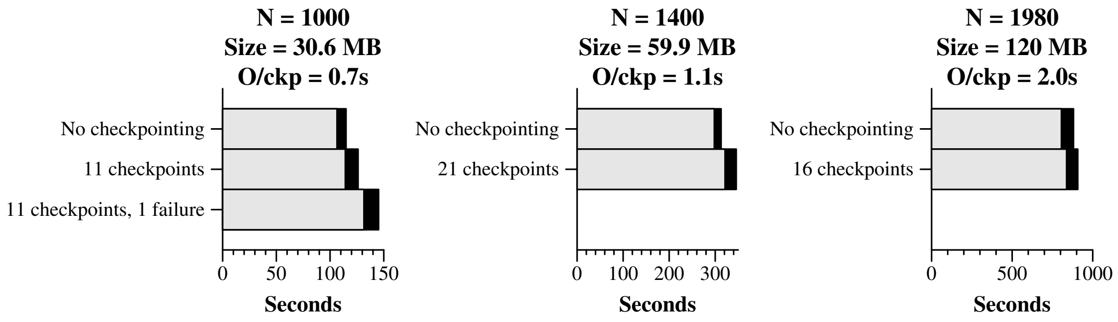

Figure 3: Performance of `dmatmul` on the CLUSTER environment.

Results from the CLUSTER environment are displayed in Figure 3. In this and other graphs, The light shaded areas are the server times only. The dark areas add the client interaction times. As expected, the CLUSTER environment exhibits high performance. The ATM network, large physical memories, and copy-on-write optimization combine for extremely high performance. For example, on the $N = 1000$ run, the overhead of checkpointing every ten seconds on the total client/server transaction is 9.7 percent, and the overhead of checkpointing every ten seconds and absorbing one failure is 26 percent.

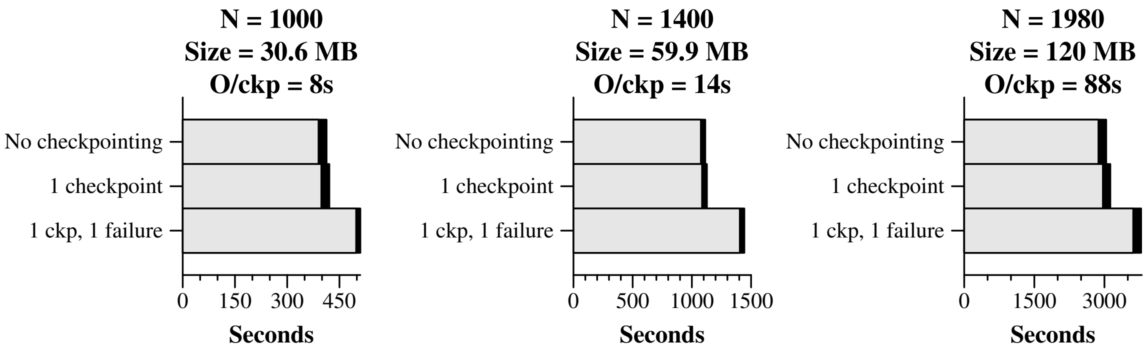

Figure 4: Performance of `dmatmul` on the LOCAL environment.

Results from the CLUSTER environment are displayed in Figure 4. As would be expected, the performance of the service is slower due to the slower processors. Likewise, the performance of checkpointing, recovery, and the contact with the client are all worse due to the slower interconnection network. However, in all cases, rolling back from the checkpoint improves performance over restarting from the beginning.

Finally, results from the WIDE environment are displayed in Figure 5. In these graphs, the black boxes are much larger due to the fact that the input and output matrices are being passed across the Internet. Interestingly,

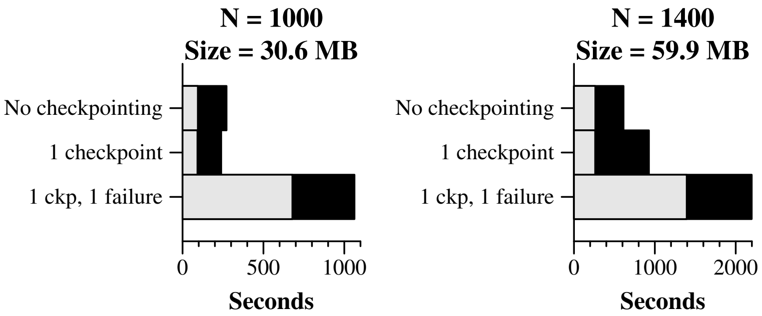

Figure 5: Performance of `dmatmul` on the WIDE environment.

even though the checkpoints too are being passed across the Internet, the checkpoint overhead is negligible in comparison to the fluctuation due to non-dedicated access. Once again, this is due to the copy-on-write optimization. However, when a recovery is required, the checkpoint file must be moved across the Internet before recovery may begin, resulting in a severe performance penalty. In this instance, a restart from the beginning would perform better than restarting from the checkpoint. This experiment serves to underscore that it is more important to select the recovering server to be close to the checkpoints than it is to select the checkpointing server to be close to the checkpoints. This is because checkpoints are taken asyncronously, while state restoration is by nature synchronous.

6 Conclusion, Limitations and Deployment

In this paper, we have described a system architecture that brings fault-tolerance and migration to scientific users who need not be computer systems experts. There are two main limitations to this system. First, if a user is not making use of the core NetSolve system services (e.g. linear algebra subroutines) listed in section 2.1, then the “not an expert” label applies less forcefully, as the user must learn how to configure the NetSolve servers. Although this task is made easier by Java-based tools [CD97b], it is a level of complexity higher than simply using NetSolve from Matlab or Excel.

The second limitation is the restriction that the checkpointing and recovering machine must be of the same architecture. This limitation arises from the fact that Starfish is a core-dump style checkpointer. The architecture of the system could easily be extended to use more portable checkpointing substrates, such as applications that implement their own checkpointing and rollback recovery with the help of libraries such as `libft` [HKW95], or a toolkit that embeds portable checkpoints into arbitrary programs [RS97]. We are exploring using the Porch toolkit [Str98] to add portable checkpointing to the core NetSolve services.

As described above, this checkpointing system has been implemented and tested. It is anticipated that it will be included as part of the official NetSolve distribution (`http://www.cs.utk.edu/netsolve`) in the year 2000.

7 Acknowledgements

This material is based upon work supported by the National Science Foundation under grants ACI-9876895, CCR-9703390 and CDA-9529459, and by the Department of Energy under grant DE-FC0299ER25396. The authors thank Dorian Arnold for help with modifying NetSolve, and Roy Friedman and Jack Dongarra for their support.

References

[AF99] A. Agbaria and R. Friedman. Starfish: Fault-tolerant dynamic MPI programs on clusters of workstations. In *8th IEEE International Symposium on High Performance Distributed Computing*, 1999.

[CCG+95] 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.

[CD97a] H. Casanova and J. Dongarra. NetSolve: A network server for solving computational science problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, 1997.

[CD97b] H. Casanova and J. Dongarra. The use of Java in the NetSolve project. In *15th IMACS World Congress on Scientific Computation, Modeling and Applied Mathematics*, Berlin, 1997.

[CPL97] 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.

[EAWJ99] E. N. Elnozahy, L. Alvisi, Y-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. Technical Report CMU-CS-99-148, Carnegie Mellon University, June 1999.

[EJZ92] E. N. Elnozahy, D. B. Johnson, and W. Zwaenepoel. The performance of consistent checkpointing. In *11th Symposium on Reliable Distributed Systems*, pages 39–47, October 1992.

[EZ92] E. N. Elnozahy and W. Zwaenepoel. Manetho: Transparent rollback-recovery with low overhead, limited rollback and fast output commit. *IEEE Transactions on Computers*, 41(5):526–531, May 1992.

[EZ94] E. N. Elnozahy and W. Zwaenepoel. On the use and implementation of message logging. In *24th International Symposium on Fault-Tolerant Computing*, pages 298–307, Austin, TX, June 1994.

[HKW95] Y. Huang, C. Kintala, and Y-M. Wang. Software tools and libraries for fault tolerance. *IEEE Technical Committee on Operating Systems and Application Environments*, 7(4):5–9, Winter 1995.

[MS99] D. Manivannan and M. Singhal. Quasi-synchronous checkpointing: Models, characterization and classification. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):703–713, July 1999.

[PBE+99] J. S. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany, and R. Wolski. The Internet Backplane Protocol: Storage in the network. In *NetStore '99: Network Storage Symposium*. Internet2, http://dsi.internet2.edu/netstore99, October 1999.

[PBKL95] J. S. Plank, M. Beck, G. Kingsley, and K. Li. **Libckpt**: Transparent checkpointing under Unix. In *Usenix Winter Technical Conference*, pages 213–223, January 1995.

[PE98] J. S. Plank and W. R. Elwasif. Experimental assessment of workstation failures and their impact on checkpointing systems. In *28th International Symposium on Fault-Tolerant Computing*, pages 48–57, Munich, June 1998.

[PL96] J. Pruyne and M. Livny. Managing checkpoints for parallel programs. In *Workshop on Job Scheduling Strategies for Parallel Processing (IPPS '96)*, 1996.

[Pla99] J. S. Plank. Program diagnostics. In John G. Webster, editor, *Wiley Encyclopedia of Electrical and Electronics Engineering*, volume 17, pages 300–310. John Wiley & Sons, Inc., New York, 1999.

[RS97] B. Ramkumar and V. Strumpen. Portable checkpointing and recovery in heterogeneous environments. In *27th International Symposium on Fault-Tolerant Computing*, pages 58–97, June 1997.

[Ste96] G. Stellner. CoCheck: Checkpointing and process migration for MPI. In *10th International Parallel Processing Symposium*, pages 526–531. IEEE Computer Society, April 1996.

[Str98] V. Strumpen. Porch: Portable checkpoint compiler. `http://theory.lcs.mit.edu/~porch/`, 1998.

[TL95] T. Tannenbaum and M. Litzkow. The Condor distributed processing system. *Dr. Dobb's Journal*, #227:40–48, February 1995.

[Vai97] N. H. Vaidya. Impact of checkpoint latency on overhead ratio of a checkpointing scheme. *IEEE Transactions on Computers*, 46(8):942–947, August 1997.

[WHV+95] Y-M. Wang, Y. Huang, K-P. Vo, P-Y. Chung, and C. Kintala. Checkpointing and its applications. In *25th International Symposium on Fault-Tolerant Computing*, pages 22–31, Pasadena, CA, June 1995.

[WSH99] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15, 1999.