A Scalable Software Architecture Booting and Configuring Nodes in the Whitney Commodity Computing Testbed[1]

Samuel A. Fineberg

NAS Technical Report NAS-97-024
October 1997

MRJ Technology Solutions
Numerical Aerodynamic Simulation
NASA Ames Research Center, M/S 258-6
Moffett Field, CA 94035-1000

Abstract

The Whitney project is integrating commodity off-the-shelf PC hardware and software technology to build a parallel supercomputer with hundreds to thousands of nodes. To build such a system, one must have a scalable software model, and the installation and maintenance of the system software must be completely automated. We describe the design of an architecture for booting, installing and configuring nodes in such a system with particular consideration given to scalability and ease of maintenance. This system has been implemented on a 40-node prototype of Whitney and is to be used on the 500 processor Whitney system to be built in 1998.

1. Work performed under NASA Contract NAS 2-14303

1.0 Introduction

Recent advances in “commodity” computer technology have brought the performance of personal computers close to that of workstations. In addition, advances in “off-the-shelf” networking and operating system technology have made it possible to design a parallel system made purely of commodity components, using a public domain operating system, at a fraction of the cost of MPP or workstation components. The Whitney project, at NASA Ames Research Center, attempts to integrate these components in order to provide a cost effective parallel testbed.

While the cost/performance benefits of using commodity components may be clear, there are also obvious problems in scaling such a system to more than a couple of dozen nodes. One of the key issues in building such a system is designing a node installation and management software that scales. This issue, while it may seem to be secondary, is in fact vital to building a serviceable and useful MPP system. For example, when 100 nodes are delivered, how long does it take to integrate and install them. Also, when a node fails, how hard is it to swap in a new system to replace it. If it takes 20 minutes to install a new node, it would take a week to install a 500 node system. Therefore, we must build a software system that is automatic, requires minimal system operator intervention, and allows spare nodes to acquire their identity easily. Further, we must do this without requiring the node to have a keyboard or monitor installed, since providing these would be expensive and would require prohibitive amounts of space in a large system.

In this paper an architecture for such a system is described. This system is currently working in the 40-node Whitney prototype and it has been designed to be scalable to the next Whitney system, with around 500 nodes. While there are likely to be additional scalability issues that will appear as Whitney grows, the prototype system works well and is designed in such a way that additional capacity can be added if necessary.

The rest of this paper is organized as follows. Section 2 describes the overall architecture of Whitney, both hardware and software. In Section 3, trade-offs in designing a boot/configuration architecture are described as well as the specific architecture design utilized in Whitney. Finally, Section 4 describes future work and additional scalability issues that may be faced in the final Whitney design.

2.0 Whitney System Architecture

As an experiment in using commodity technologies, Whitney is designed to take advantage of low cost node hardware as well as public domain software and operating systems. To date, we have chosen to use the Linux OS and Intel Pentium Pro processor based systems. However, the architecture of Whitney is not dependent on the operating system or on the node hardware. Instead, we concentrate on a design that is flexible and that does not rely on any uncommon hard-

ware or software features. This means that the Whitney design could be built on top of almost any UNIX system or Windows NT utilizing any stand alone system (i.e., workstation or PC).

2.1 Hardware

The Whitney hardware architecture is shown in Figure 1. Whitney consists of a

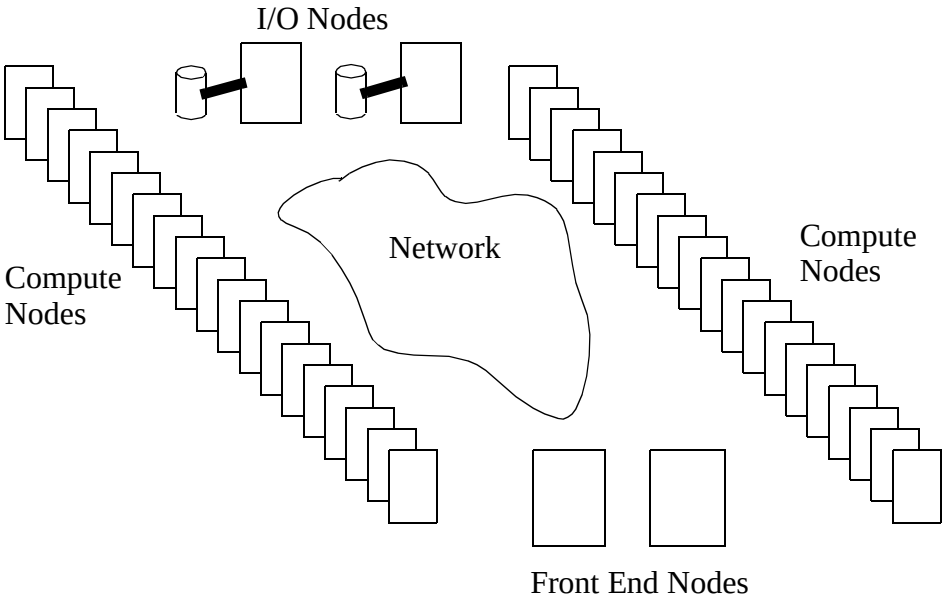


Figure 1: Whitney system architecture block diagram

large set of compute nodes, each of which is a small desktop system. Therefore, each node contains a hard disk, memory, a floppy drive, etc. Some of these nodes have special functions, as determined by Whitney's system software, and may also have additional disk (for I/O nodes), network connections, compiler software, etc. The Whitney nodes are all attached using an interconnection network, and files reside on either a front end node or on a parallel file system that is built on top of the I/O nodes. The target configuration for Whitney is 50 processors in Fiscal Year (FY) 1997, 500 processors in FY 1998, and 5000 processors in FY 1999/2000.

Currently, Whitney consists of 36 compute nodes, 3 I/O nodes, and 1 front end node. The compute nodes are configured as follows:

- Intel Pentium Pro 200MHz/256K cache
- ASUS P/I -P65UP5 motherboard, Natoma Chipset
- ASUS P6ND CPU board
- 128 MB 60ns DRAM memory
- 2.5 GB Western Digital AC2250 hard drive
- Trident ISA graphics card (used for diagnostic purposes only)

They also contain at least one DEC “tulip (21140)” based Fast Ethernet card and a Myrinet card. The final interconnection network for Whitney is still under design. Whitney attempts to deliver the most performance per dollar, so we have been evaluating a number of commodity and special purpose networks including Fast Ethernet and Myrinet. The ideal network will be scalable to 1000’s of nodes and will deliver adequate performance for the lowest cost. For more information on this evaluation see [BeN98, PeF97].

The I/O nodes are similar to compute nodes except that they contain 256MB RAM, two Pentium Pro processors, and two 9GB SCSI disks. The Front end node is a single processor system with 128MB RAM, a 4GB SCSI disk, and a second Fast Ethernet connection for routing to the general NAS network and the Internet.

2.2 Operating System Architecture

While the hardware architecture for Whitney is relatively simple, there were some major challenges in designing a manageable base software architecture for such a large system. First, we had to decide on what type of an operating system we would run on each node. If some sort of distributed operating system was utilized, as in the OSF/1 AD on the Intel Paragon [Int93, Zaj93], there would be a large amount of overhead for coordinating system nodes. This would almost certainly overload the network and result in poor system performance as the system scaled to a large number of nodes. Another approach would be to run a small bootstrap loader on each node, as in the iPSC/860 [Int91] or SUNMOS [Rie94, Whe94] on the Intel Paragon. This approach scales, but leaves the processing nodes with limited I/O capability, i.e., the nodes can not open network connections outside of the machine and nodes can not have local disk or virtual memory. Another issue that exists with both the distributed OS approach and the bootstrap loader approach is maintainability. Neither of these models is utilized in a widely deployed system, so the operating system would have to be built specifically for Whitney, or ported to Whitney. In addition, we could not benefit from the economies of scale derived from using a workstation or PC operating system.

The approach that was taken was to utilize a widely deployed off the shelf UNIX based operating system, Linux, on all nodes throughout the system. While the basic operating system is consistent, the components installed differ depending on node functionality. Compute nodes contain a stripped down version of the system, containing only those components needed for running application codes, i.e., run-time libraries, shells, debuggers, etc. I/O nodes also have a stripped down version of the system, with the addition of the I/O server software. The front end systems have all of the necessary compilers, editors, etc., needed to build application codes for the compute nodes.

While this basic approach has been used to build scalable systems in the past such as the IBM SP-2 and the Meiko CS-2 [CaF96], making such a system

maintainable with 100's (much less 1000's) of nodes is still difficult. One of the main pieces we have chosen to use for integrating the system is the Portable Batch System[1] (PBS). In its "parallel aware" form, PBS daemons run on each node. They start user jobs, make sure jobs complete correctly, enforce resource limitations, and measure resource usage. This is far more scalable than a "single system image" because only the actual parallel jobs need to be managed as an ensemble. Further, it distributes the system management tasks and puts the more system oriented functions on the actual node they control while still maintaining a central task scheduler.

3.0 Booting, Configuring, and Maintaing Compute Nodes

One of the major scalability bottlenecks in systems such as Whitney is how to go from an unconfigured node, as delivered from the manufacturer, to a fully configured compute node. Another problem is; how does one upgrade a compute node when a new version of the operating system becomes available? Finally, what does one do when a node fails? Whitney's approach to these problems is to treat compute nodes (though not necessarily I/O and front end nodes) as interchangeable components. Every node has exactly the same operating system and system software on their hard disk, and other than two files which contain the node's identity, there is no way to distinguish between nodes. This makes maintenance much easier. When a node fails one can simply replace it with another one, only the two files must be changed. Then, hardware failures can be diagnosed after the bad node has been replaced, leaving the main system functional. However, there are still many problems that need to be solved for this approach to work.

3.1 Loading and configuring node software

Assuming you have 500 PCs; how do you make them Whitney nodes? The software could be loaded off of a CDROM, floppies, or the network. A CDROM would work, but then you would have to build a custom CDROM for Whitney and each node would need its own drive (unless if you wanted to move a portable drive 500 times). Floppies are not practical because it would require dozens to load a functional system. The answer, of course, is to install through the network, which is already attached to every node. However, to make this system manageable, we still need to ensure that nodes can be installed with the minimal amount of operator intervention. For example, if it takes an operator 1 minute per node it requires 8.5 hours to install a 500 node system, if it takes 10 minutes per node it will take 3.5 days. Therefore, Whitney nodes should be able to install and configure themselves automatically. Installing and configuring nodes automatically on a network is quite possible, but in order to do so, the server must be able to tell the identity of a node in order to set its two unique files correctly and to prevent network address conflicts.

1. For more information see http://science.nas.nasa.gov/Software/PBS

One approach is to burn some unique identifier in to the system's ROM. This information can then be exchanged with the server to determine the node's identity (e.g., the UNIX "BOOTP" protocol). This approach works, but it requires someone to create hundreds of unique ROMs. Instead of these unique system ROMs, one could use ethernet addresses (which are unique identifying numbers burned in to ROM) to identify a node. Unfortunately, the problem with both of these approaches is that the system administrator will be forced to manage a list of hundreds (or thousands) of obscure numbers. When a node fails, the operator would have to change a number in some table with the identifier of the spare to be swapped in, or the ROM or ethernet card from the failed node would have to be swapped with the spare. This poses a major impediment to system maintainability, and the initial configuration of the system would be daunting.

Another approach is to dynamically assign node identity. Then, as nodes come on line they could be assigned their network address and that address will last only until they are turned off or go down. The server would manage a pool of addresses and allocate/deallocate them as necessary. This approach scales well, however, it has one major drawback. That is, when a node fails, how do you tell which node is down? Normally, the way a node failure is detected is when a node stops responding to the network. If node addresses are dynamically assigned, there is no way to tell where a node resides physically from its network address. This is less of a problem with a small system, but with hundreds of systems it would be virtually impossible to troubleshoot node failures if node addresses are not static.

The final approach, which was adopted for Whitney, is to have a device that identifies a node that can be easily moved between systems (i.e., it does not require the system's case to be opened). When a node fails, one must simply move this device to the new system and it will magically take on the identity of the failed node. One such device is a ROM that can be attached through a parallel or serial port (e.g., a software lock like the HASP[1]). The problem is that these are expensive and they would require some sort of custom "BOOTP" style server. We have still not eliminated this approach as a possibility, but for the early stages of Whitney, we instead decided to use a 3.5" floppy disk as the uniquely identifying device. This has several advantages. Floppies are cheap and easily created. We can even use Whitney nodes to create their own floppies or new floppies for other systems. Second, floppies can hold real files, so we actually store the two unique node files on the floppy and they can simply be copied to the Whitney node upon booting. Finally, if you also use this floppy for booting, the system can be booted even when there is no data on the system's disk without requiring a special network boot BIOS. There are also some disadvantages to floppies. They wear out, they can be erased or corrupted, and we still do have to create one for each system as part of the initial system install.

---

1. see http://www.hasp.com

However these issues are easily overcome with some proactive maintenance (i.e., frequent creation of spares and floppy replacement).

Once a node has its identity, installing and configuring node hardware is fairly simple. Workstations have had this capability for many years. The only special requirements that Whitney must adhere to are that the installation process must be completely automated. However, this is fairly easy since all nodes are to be configured identically. In the next section the actual boot/install/configure process implemented on Whitney is described.

3.2 The boot/configure architecture of Whitney

When a node is booted, there are several possibilities. It can be fully installed and configured, it can have some information on its hard disk, but not be fully configured, or it can have a blank hard disk. To eliminate the need for operator intervention when installing Whitney nodes, the Whitney nodes can boot regardless of their previous state. If a node has a blank hard disk, a node disk image is automatically copied to the node disk. If a node's disk is valid, the node is reconfigured based on both the node specific information contained on the boot floppy and other configuration scripts on the server. A block diagram of this process is shown in Figure 2.

Referring to Figure 2, nodes boot from their floppy drive. The floppies must therefore contain the most recent kernel as well as the node's TCP/IP configuration information and the Linux bootstrap loader (LILO). Normally when linux is booted, only a single device can be specified for the root filesystem. Linux therefore boots and attempts to mount root from a partition on the first IDE disk /dev/hda. However, because we did not want to assume that disks must be configured before booting, we modified the Linux kernel such that it will attempt to mount an NFS root if the first attempt to mount a root filesystem fails.

Assume that a disk starts out blank. When Linux attempts to mount it, the mount will fail. Then, we instead mount a special NFS root filesystem. This NFS root continues to boot and then runs a special install script. The install script first checks if a count of the number of unsuccessful installs (stored on the floppy) is too high. This prevents nodes from installing forever in the case of a bad disk. If this check succeeds, a disk image is copied from the file server on to the root filesystem (/dev/hda). This disk image is simply a minimal installation of Red Hat Linux without any modifications from what comes off the Red Hat CD. All changes to this base image are made in other scripts. By not modifying the CD install image, we can easily generate a new disk image, and all changes from this

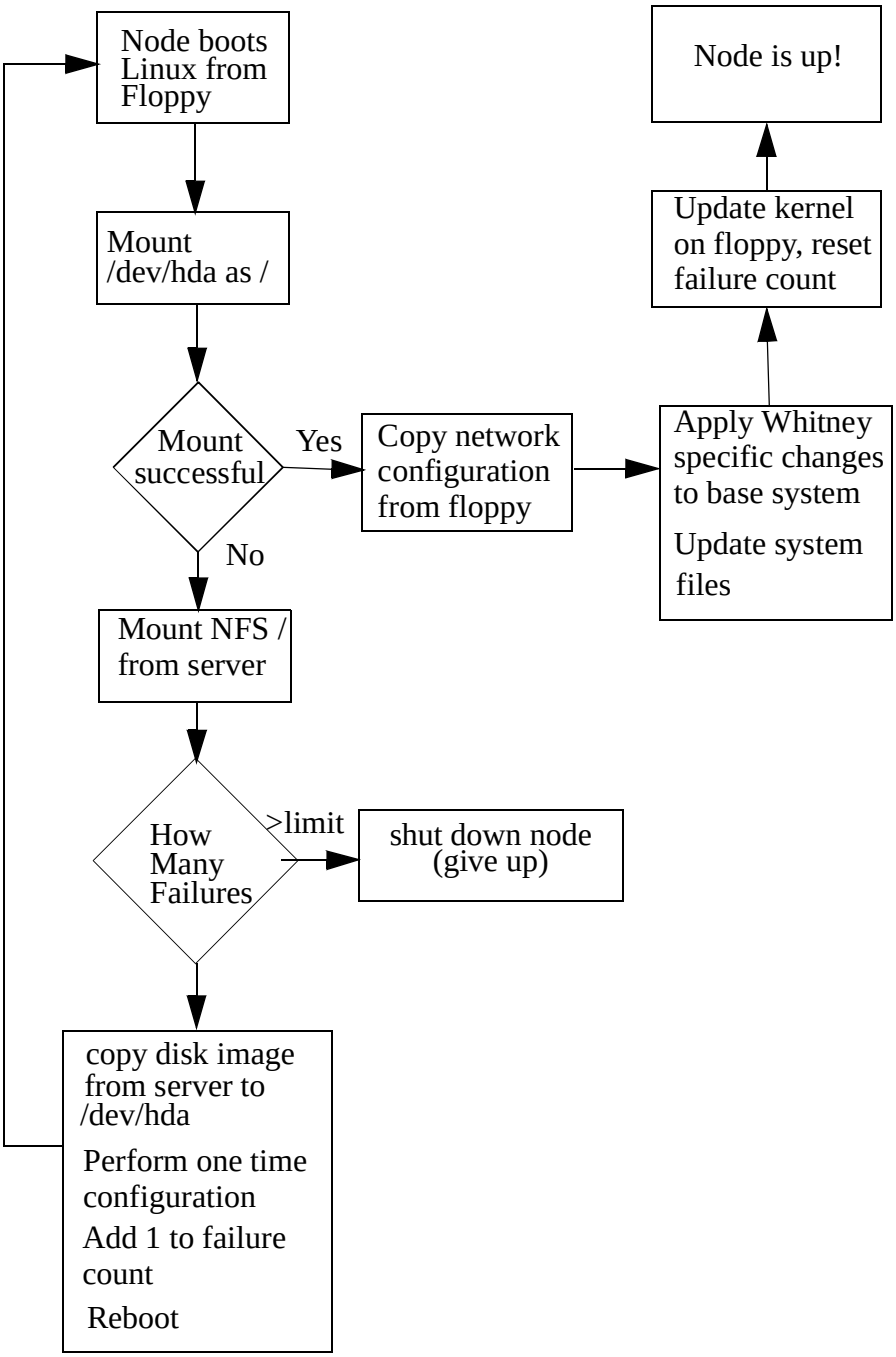


Figure 2: Whitney node boot time configuration process

base image are self documented by the scripts that make them. Table 1 summarizes the names and functions of the various scripts used to configure Whitney nodes.

After the initial disk image is copied to the root disk, the “pre-install” script performs a generic set of one time configuration changes. These changes are

Table 1: Whitney installation scripts

Script	Network state	Root device	location	Description
pre-install	up	NFS	install root on server (located in /export/roots)	One time changes and changes needed to bootstrap self configuration process.
setup-early	down	/dev/hda	node root disk	Changes needed to configure network, i.e., copy TCP/IP configuration from floppy. Also, floppy maintenance (check floppy file system, reset install count).
setup-late	up	/dev/hda	/usr/admin NFS filesystem on server	Copy system files from /usr/admin to appropriate directories, update floppy kernel from /usr/admin, make other necessary changes, update setup-early script from /usr/admin.

Whitney specific, but are not specific to the particular node being installed. The types of changes made at this point are those that are either one time changes that can not be repeated, or changes that are needed to make the boot procedure work after the root disk is installed. For example, the /usr/admin directory (where Whitney specific files are NFS mounted) is created, initial /etc/hosts and /etc/passwd files are copied to the root disk, and the “setup-early” script is copied to the root disk. Upon completion of the initial node configuration, the node is rebooted (now with a valid filesystem on its hard disk).

After the node reboots, the Linux kernel is once again loaded from the node’s floppy. However, this time when the system attempts to mount /dev/hda, it succeeds. The node then begins booting normally. Before the node’s network comes up, it runs the “setup-early” script. This script copies network configuration information from the floppy to the node’s hard disk. It also performs any other configuration steps that are needed before the node’s network comes up (e.g., setting up routing tables).

Finally, the node brings up its network as well as remote (NFS mounted) filesystems. Then it runs the “setup-late” script. This script performs any other node configuration, and updates all global configuration files and scripts from the copies kept on the file server.

The setup-early and setup-late scripts are run every time a node is booted with a valid filesystem. Therefore, they must perform their changes in a way that is not affected by whether the changes have already been made or not. Further, they must replace any files that may have been set incorrectly in the past (e.g., if we change a node’s IP address by moving boot floppies, all of the old networking files must be replaced with the new ones).

After running “setup-late,” the node is ready and configured. This procedure works well in most cases. However, if nodes become corrupted without actually destroying the file system, this may not work. In these cases, an operator must intervene, however, the only intervention needed to fix a corrupted node is to force it to re-install itself. We do this by booting a special disk that contains a small linux kernel and file system. When this disk boots it runs a script that corrupts the first hard drive’s superblock, thus triggering the node to re-install itself on the next boot.

Once in operation, if changes need to be made to the compute nodes, these must be done in one of the three setup scripts. Then, the change can either be made manually or the nodes can be rebooted to make the change. If changes are not made in the scripts, they will likely disappear the next time a node is rebooted. While this may seem like a hassle, it enforces careful integration of changes and makes the scripts self documenting. This type of careful management of node configuration and easy reproducibility of changes is vital to operate a large system.

4.0 Additional Scalability Issues

The major scalability bottleneck in this design, assuming the network is adequate, is the server. Even with a 40 node system, if all nodes simultaneously try to install their internal disks at once the system will fail. To alleviate this problem, we try to prevent more than 10 nodes from simultaneously installing. This, however, may be a problem with a larger system because each install takes about 10 minutes. Therefore, in a larger system it will be important to replicate the server on several alternate server nodes. Then, depending on the location on the system’s network, the nodes must install from one of the multiple servers. This is relatively easy since the boot disk not only contains the node’s TCP/IP information, but also the address of the node’s boot/install server. The only time this may be a problem is if the system does not have all nodes on the same TCP/IP subnet with their server. If this is the case, TCP/IP routing must be set up prior to the nodes booting. Therefore, it is recommended that in the final Whitney there be at least some network in the system that connects nodes to servers without any routing.

Another possible bottleneck is in the use of NFS to run user jobs. While it may be possible to perform low bandwidth I/O across NFS with 500 nodes, it may not work. We will not necessarily be able to determine whether this strategy will work until the large system is built. If NFS does not work properly, it will be necessary to stage user files on to the nodes prior to execution. Also, until the parallel file system is available it will be necessary to stage files generated during job execution off of the nodes’ local filesystems. Fortunately this facility is already built in to PBS.

5.0 Conclusions and Future Work

We have described an architecture for maintaining system software on a commodity cluster based system. The system consists of a floppy disk containing a node's kernel and identification information, a server that contains both the initial installation image for each node as well as updated system configuration files, and a set of scripts that keep the nodes' configuration up to date regardless of its previous state. This system is designed for maximal scalability and ease of maintenance. While there will likely be additional issues encountered when building the 500 processor Whitney system, this work should form a good basis for the final software implementation.

6.0 References

[BeN97] J. Becker, B. Nitzberg, and R. Van der Wijngaart, "Predicting cost performance trade-offs for Whitney: a commodity computing cluster," *31st Hawaii International Conference on Systems Science*, to appear, January 1998.

[CaF96] T. L. Casavant, S. A. Fineberg, M. L. Roderick, and B. H. Pease, "Massively Parallel Architectures," in *Parallel Computers*, Casavant, Tvrdik, Plasil eds., IEEE Computer Society Press, Los Alamitos, CA, pp. 11-71, 1996.

[Int88] Intel Scientific Computers, "The iPSC/2 system," *Third Conference on Hypercube Concurrent Computers and Applications*, pp. 843-846, January 1988.

[Int91] Intel Supercomputing Systems Division, *iPSC/2 and iPSC/860 Users' Guide*, Intel Corporation, Beaverton, OR, 1991.

[Int93] *Intel Paragon XP/S User's Guide*, Intel Supercomputing Systems Division, Beaverton, OR, October 1993.

[PeF97] K. T. Pedretti and S. A. Fineberg, "Analysis of 2D torus and hub topologies of 100Mb/s ethernet for the Whitney commodity computing testbed," *NAS Tech. Report NAS-97-017*, NASA Ames Research Center, September 1997.

[Rie94] R. Riesen et. al., *Experience in implementing a parallel file system*, Sandia National Laboratory, 1994

[Whe94] S. Wheat et. al., "PUMA: an operating system for massively parallel multicomputers," *Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences*, 1994.

[Zaj93] R. Zajcew et. al., "An OSF/1 Unix for massively parallel multicomputers," *1993 Winter USENIX*, January 1993.



NAS TECHNICAL REPORT

Title:

A Scalable Software Architecture Booting and Configuring Nodes in the Whitney Commodity Computing Testbed

Author(s):

Samuel A. Fineberg

Reviewers:

"I have carefully and thoroughly reviewed this technical report. I have worked with the author(s) to ensure clarity of presentation and technical accuracy. I take personal responsibility for the quality of this document."

Two reviewers must sign.

Signed: _____

Name: _Jeff Becker_____

Signed: _____

Name: _Parkson Wong_____

After approval, assign NAS Report number.

Branch Chief:

Approved: _____

Date:

NAS Report Number: