Encouraging the Unexpected: Cluster Management for OS and Systems Research

Ronan Cunniffe, and Brian A. Coghlan,

Department of Computer Science
Trinity College Dublin, Ireland
{ronan.cunniffe, brian.coghlan} @cs.tcd.ie

Abstract. A framework for cluster management is proposed that enables a cluster to be more efficiently utilized within a research environment. It does so by removing cluster management to a management node, leaving the compute nodes as essentially bare machinery. Users may schedule access to one or more of the compute nodes via the management node. At the scheduled time, a previously-saved image of their research environment is loaded, and the session begun. At the end of the session the user may save a new image of the environment on the management node, to be reloaded at another time. Thus the user may work with a customized environment, which may even be a fledgling operating system, without fear of interference to other researchers. This enables the capital investment of a systems research cluster to be amortized over a greater number of researchers.

1. Introduction

We assume clusters represent a significant capital investment, and that in order to maximise the utilization of that investment they are very closely managed, with jobs scheduled on the compute nodes by management software such as CCS [1], PBS [2,3], LSF[4] or LoadLeveller [5] integrated with the working environment. It is likely then, that any modifications to the working environment that might jeapordize stability will be very unwelcome.

Hence OS or systems research, that is likely to jeopardize stability, requires either a private cluster or diplomatic negotiations over how far any researcher can modify the shared working environment. The former is a very inefficient use of funding, the latter a severe constraint on research targets. Here we propose a scheme that allows the researcher the illusion of a private cluster whilst in fact using a shared cluster.

2. The MultiOS Framework

What we wish to describe here is the general philosophy rather than a specific implementation, since it is felt that any implementation may need to be adapted to site-specific tools (schedulers, for instance) and the adaptability should be part of the philosophy. However, describing the framework is probably best done in concrete terms, by describing the cluster for which it is being initially designed, how it will operate on that cluster, and what hardware/software tools are required.

Our cluster is made up of sixteen PC compute nodes and two storage servers, linked by a switched fabric of SCI links. In addition, there are two NIS servers, a http server and a firewall server. The compute nodes are arranged in logical groups of 4. Each compute node has 256MB of DRAM and a 2GB local hard-disk. The storage servers are connected to a large RAID, and all machines are connected to the external network via 100Mb/s Ethernet. All normal access to the cluster is through this network connection and physical access to compute nodes and servers is minimised. The MultiOS server will execute either on an extra server node, or on one of the existing servers. Over time, it is hoped to increase the number of compute nodes.

The central idea behind the MultiOS framework is that between two successive zero-management ‘research’ sessions there is a ‘management’ session where *all* access is suspended, and the environment installed on the local hard-disks can be changed by management software according to the schedule or via user interaction with the MultiOS server. Three requirements must be met to do this. The MultiOS server must be able to:

1) *force* a reboot of any compute node on demand (via a hardware mechanism)
2) gain control of a compute node during boot - before any environment starts.
3) install and run a management environment that does not use the local disk.

2.1 A Hardware Reset Mechanism

There is no guarantee that a running environment will respond gracefully to a request to shut down, indeed the highly experimental work this framework is designed to support is quite likely to crash or lock up the hardware it is running on. Disruption of the schedule is not acceptable, nor is demanding human intervention. MultiOS *must* be able to regain control after a crash. In our case, this will either be implemented using a LonWorks network [6] with the module in each compute node wired to the reset pin, or a parallel switched 100Mbps Ethernet fabric with modified wake-on-LAN [7].

2.2 Control must be passed to the MultiOS Server during Boot.

This can be done by using the standard protocols for booting diskless workstations but in a slightly non-standard way. Most PC Ethernet cards have a socket for a ‘bootrom’, which can be recognised by the normal boot sequence and invoked before any other bootable device. This bootrom sends a BOOTP [8] request to find the machine’s own identity and the name of a file to download, and then uses TFTP [9] to download it. Once downloaded, that file is executed.

Obviously, if two different files are downloaded on two successive boots, the compute node will boot differently. This is what MultiOS does, alternating between two executables. The first executable is the management environment; the second is a tiny program which simply passes control straight on to the boot-block of the local hard-disk, so that the compute node boots into whatever target environment has been installed, as though the network interrogation phase never occurred.

2.3 A Special Management Environment which does not use the Local Disk.

This special management environment could be a specialised program, or a suite of specialised programs. Many already exist [10, 11, 12], but hardcoded solutions are not flexible - it is difficult to take advantage of a hardware configuration or network topology unless the program is re-written for each specific cluster. It also does not easily exploit possible redundancy of transmission, where multiple compute nodes are to be loaded with near-identical images. Essentially this approach suffers from conception in a vacuum, where every new optimisation requires new tools or low-level modifications to existing ones.

A much more flexible and powerful approach is to use a fully featured operating system, and to assemble custom solutions from its native toolkit. In our case this OS is Linux - the management environment downloaded via TFTP is actually a Linux kernel configured as for a diskless workstation - mounting some of its filesystems in memory, and the remainder over NFS (the OS images for instance). The high-level tools for moving OS images across the network are then built on the standard UNIX commandline tools, such as *dd, gzip, diff, rcp*, etc. The main costs of using Linux are the image storage (a 'minimal' Red Hat 6.0 install, slightly modified for diskless operation, is 20MB per node and 80MB shared by all), and the network bandwidth consumed by multiple parallel accesses to this resource.

3. The MultiOS Server

We have described the low-level mechanism for transparently switching between research environments. A permanent high-level control mechanism - the MultiOS server - is also needed, to implement user commands and to provide status information. A further requirement is to allow users to reserve some or all of the cluster in advance.

In line with our preferences for adaptability, it is proposed not to integrate a custom node reservation system or scheduler into the MultiOS server, but to use an external program. A basic implementation will be provided, but a modular approach allows it to be easily replaced. Likewise, we do not intend to integrate a user interface into this server, allowing for whatever variations on user access are desired. We intend to provide a web-based console, but this may not always be appropriate. The overall structure of the MultiOS 'server', therefore, is really a set of three elements: the user interface program, the reservation system, and the actual MultiOS server program that controls the low-level operation of the framework.

4. Security Issues

The MultiOS scheme introduces important security issues. Four primary areas of concern have been identified.

Firstly, the BOOTP and TFTP protocols were designed to be small rather than secure, and are vulnerable to a variety of attacks, denial-of-service (DOS), IP and MAC address spoofing, etc. A successful attack results in the node running an OS of the attacker's choice. Machines booting in this way have to be considered as totally unprotected, and therefore a weak point in whatever network(s) they belong to. The only effective workaround is to boot from a separate physically secure network.

Secondly, the image server must allow external access to the environment images, both for users and for the MultiOS framework, and both read and write access must be subject to authorization. However, since a management cycle involves access from the compute nodes, the user's authorization must be transmitted explicitly to them. Transmissions can be encrypted, but transport layer security is useless if one of the parties is compromised. Again, using a switched private network eliminates this concern.

Thirdly, it is ironic that the behaviour of a cluster being reset is indistinguishable from a well-synchronised distributed-denial-of-service (D-DOS) attack on the MultiOS server, mounted by a large number of high-bandwidth attackers. Services such as BOOTP, TFTP, and the network filesystems NFS and SMB are commonly configured to shut themselves down when attacked, rather than overload the server machine they run on. For the purposes of MultiOS, either the cluster-resetting must be done in staggered fashion to keep the load below the triggering threshold, or this mechanism must be disabled, and the cluster made physically secure against such attacks instead. Since the MultiOS server directly controls the compute node reset mechanism, the former is likely to be easier.

The final area of security risk is the user interface. A web-based interface is favoured for customisability, platform independence and ease-of-use, but hard experience teaches that web servers can be crashed. This is a secondary argument for splitting the user-interface away from the MultiOS server. It can now reside on a separate machine, communicating through an encrypted channel. In this scenario, a webserver crash will shut down interactive access to MultiOS and prevent changes being made to the schedule, but will not affect the cluster itself.

5. Summary

A framework for cluster management has been proposed that is specifically tailored to OS and systems software research. Users may schedule access to one or more of the compute nodes via a separate management node, the MultiOS server. At the appointed time, a previously-saved image of their research environment will be loaded from an image storage server, and the session begun. At any time during the session the user may save a new image of the environment, to be reloaded at another time. Hence a number of users can be accommodated, each within their own environment. This enables the capital investment of the cluster to be amortized over a greater number of researchers.

Work on the framework, called MultiOS, began in October, 1999, and is still in progress. Our thanks to Prof. J. G. Byrne for his support.

References

1. Keller, A., Reinefeld, A., "CCS Resource Management in Networked HPC Systems", Proc. Heterogeneous Computing Workshop HCW'98, 1998

2. Portable Batch System Documentation, MRJ Ltd., 1998. http://pbs.mrj.com/docs/html

3. Henderson, R.L., "Job Scheduling under the Portable Batch System, In: Job Scheduling Strategies for Parallel Processing", Feitelson, D.G. and Rudolph, L. (eds), LNCS, pp.279-294, Vol.949, Springer-Verlag, 1995.

4. Load Sharing Facility Suite 3.2 Documentation, Platform Computing Inc., 1998.

5. Prennis, A.jnr, "Loadleveller: workload management for parallel and distributed computing environments", Proc. Supercomputing Europe (SUPEUR'96), October 1996.

6. Foster, G.T., Glover, J.P.N., Warwick, K., "Flexible Distributed Control of Manufacturing Systems Using Local Operating Networks", Proc. LonUsers International Fall Conference, 1995

7. Magic Packet Technology White Paper, AMD Publication no.20213, Advanced Micro Devices Inc., November 1995

8. Wimer, W., "Clarifications and Extensions for the Bootstrap Protocol", IETF Request For Comments Document no.1542, October 1993.

9. Sollins, K., "The TFTP Protocol (Revision 2)", IETF Request For Comments Document no.1350, July 1992.

10. Rembo Technology, http://www.bpbatch.org

11. Free Software Foundation, "Grand Unified Bootloader" http://www.gnu.org/software/grub.en.html

12. Yap, K; Savoye, R; "Network Interface Loader" http://nilo.sourcefourge.net/