MOSIX: HOW LINUX CLUSTERS SOLVE REAL WORLD PROBLEMS

Steve McClure and Richard Wheeler

USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

MOSIX: How Linux Clusters Solve Real World Problems

Steve McClure
smcclure@emc.com

Richard Wheeler
ric@emc.com

EMC² Corporation
171 South Street
Hopkinton, MA 01748

Abstract

As the complexity of software increases, the size of the software tends to increase as well, which incurs longer compilation and build cycles. In this paper, the authors present one example of how clusters of Linux systems, using the MOSIX extensions for load monitoring and remote execution, were used to eliminate a performance bottleneck and to reduce the cost of building software. We present a discussion of our original software development cluster, an analysis of the performance issues in that cluster, and the development and modifications done to MOSIX and Linux in order to produce a solution to our problem. We finish by presenting future developments that will enhance our cluster.

Introduction

As computers increase their processing power, software complexity grows at an even larger rate in order to consume all of these new CPU cycles. Not only does running the new software require more CPU cycles, but the time required to compile and link the software also increases.

EMC Corporation [EMC] is a leading vendor of storage solutions. EMC's products are more than just cabinets full of disks and tapes: they rely on a large amount of software, both inside and outside the cabinets. Each member of our development group needs to compile and link millions of lines of code on a routine basis. Building all versions of this code on an individual developer's desktop system required approximately two hours (7,200 seconds). Spending this much time waiting for code to build is unacceptable.

One way to reduce this build time is to give each developer a higher-powered desktop computer, with better processors, huge amounts of memory, and locally attached storage. However, with dozens of developers, this is not a cost-effective solution. A variation on this is to use a very large, centralized build server, although this class of machine is extremely expensive as well.

The last possibility is to use a number of smaller computers as a build cluster, where each is attached to some common storage pool so they can all operate on the same source code base. The challenge in this solution is to distribute the compilation jobs across the cluster members evenly, insuring consistent results regardless of the node on which a process runs.

The Original Cluster

EMC uses a few small clusters of computers connected to NFS file servers as our software build environment. As the size of our group and our builds grew, our existing clusters gradually became overloaded when more than two developers initiated builds concurrently. Upgrading the nodes helped somewhat, but we needed a more cost-effective solution.

This original cluster (see Figure 1) consisted of 26 SUN Ultra 2 workstations, each with 512MB of RAM and dual, 300MHz UltraSparc-II processors each with 2MB of L2 cache. The backing store was a pair of EMC Celerra NFS file servers, each connected to an EMC Symmetrix for disk storage. Each Celerra contains 14 NFS data movers, each with an

Figure 1: The Original Cluster

26 SUN Ultra 2s

FDDI Ring

2 EMC Celerra NFS Servers

SCSI

2 Symmetrix Systems
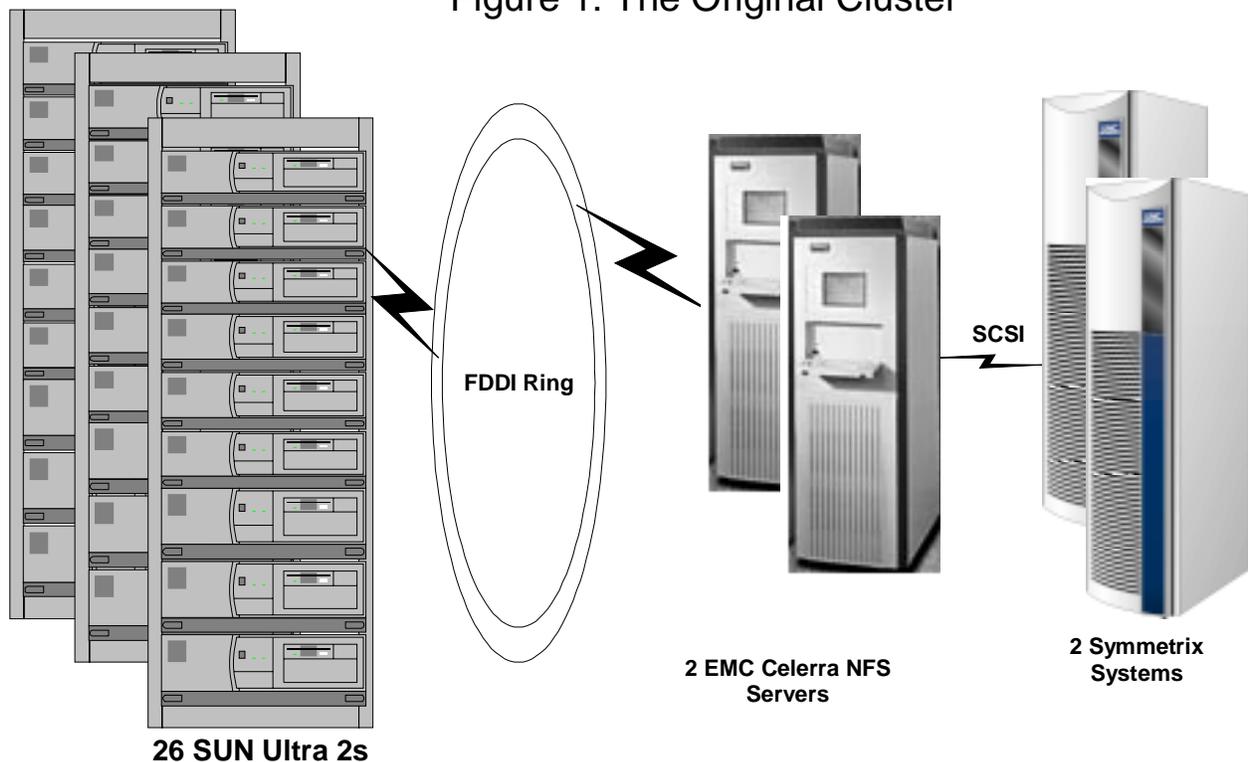
independent connection to the network. The network was a FDDI ring connecting all workstations and NFS data movers. Each user was given a single Symmetrix volume mapped to a single physical disk, accessible through one of the NFS data movers.

Our original cluster software was Platform Computing's Load Sharing Facility (LSF) version 3.1 [Platform] running on top of Solaris. The LSF solution is based on GNU make(1), which spawns parallel makes across the cluster, as well as some proprietary user-level software.

On the original, idle cluster, a typical user could build a complete set of binaries in ~13.5 minutes (813 seconds). This is a good improvement over the original two hours, but the real problem appears when many users want to build at once.

When two users run jobs on the cluster at the same time, performance degrades severely as the cluster starts to run out of resources and every node becomes too busy to accept new jobs. In this case, each user's build takes 22 minutes (1324 seconds) to complete. In addition, if subsequent users try to start new jobs, they are locked out of the build cluster with busy cluster

error messages. In this case, LSF reports that all cluster nodes are busy and it will not start any more jobs. These delayed builds begin only when the original jobs finish and the nodes are no longer busy.

In addition to the above original cluster, and at the same time the following investigation and implementation of the MOSIX cluster was underway, another LSF cluster was being assembled for use in another group. This LSF cluster consisted of 37 Sun Ultra 2 workstations, each with dual 400MHz UltraSparc-II CPUs each with 2MB of L2 cache, and 512MB of RAM. These workstations were connected to a Fore ESX-2400 100Mbit switch with 96 ports. We were able to run some test numbers on this cluster, and were able to run a maximum of four users concurrently. While this cluster was not investigated extensively by us, the performance and costs are shown in the figures and charts in relation to the other clusters and nodes.

The Investigation

Due to the problems cited above, we initiated a project to investigate alternative cluster technologies. Given one author's past experience with an earlier version of

MOSIX [Barak et al., 1993], we tried to use the Linux-based MOSIX cluster at Hebrew University [MOSIX] to build our code images.

MOSIX adds load information about the cluster, process migration, and various other clustering features to Linux without adding any new, cluster-specific APIs. This allows us to continue to use the existing compilers and build environments with little to no modifications. (Note that we use a standard GNU tool set on Linux and our other UNIX hosts).

Although MOSIX did an excellent job measuring the load across the nodes, relying on its process migration mechanism to distribute the sub-makes across the cluster nodes failed. MOSIX on Linux redirects many system calls for migrated processes through the network to a stub process that runs on the process' creation node. Since make(1) forks children locally and since compilation is system call intensive, few if any migrations actually occurred. When make(1) sub-processes were forced to migrate, we observed a huge performance drop.

After discussion with the MOSIX team, we suggested that one way to avoid this performance degradation was to use static placement of the builds instead of migration. This uses the MOSIX information monitoring abilities in order to determine process placement and then uses a local daemon on each cluster node that creates the process on the selected node. This approach avoids the performance degradation of remote system calls by using remote execution and static placement instead of local forking and dynamic migration.

As the MOSIX team worked on the remote executor, EMC started a more in-depth analysis of our existing builds. A snapshot of the build environment and source code was taken and used in all tests to ensure consistent results in our experiments. Our test measured a full, clean build (all previous results were removed from prior runs each time). Like our original cluster, source code and binaries were stored in NFS. It was also noted that large portions of the builds were not parallelized. This problem would have to be tackled later in order to see greater performance gains.

The next step in the investigation was to characterize the performance of single computers and determine a good choice for the nodes of the cluster. After running the build environment on various computing nodes, it was determined that the best hardware platform based on cost, size, network port considerations and

performance was a VA Linux [VA/Linux] 3500. Each VA Linux 3500 is a quad-CPU Intel Xeon computer, running each processor at 500MHz with 512KB of L2 cache on each processor, and 1GB of RAM. Using a single VA Linux 3500, the existing builds for a single user for a single job would take a little over 37 minutes (2233 seconds).

In addition to characterizing the performance of the individual candidate nodes, we profiled the build itself. Using slow-CPU nodes, the majority of a build's elapsed time is spent computing. As we moved to faster CPUs, the builds became increasingly I/O bound. Depending on the particular stage of the build, it could be either I/O bound or CPU bound. Overall, it was determined that the I/O requirements for a single build were huge: reading in the source files, writing out object files, linking binaries, etc. This finding reinforced our understanding of why MOSIX's default migration failed initially: since MOSIX currently does not perform local I/O, it instead sends the I/O back to the node where the process originated. The most recent release of MOSIX allows remote processes to perform I/O to some cluster file systems directly, as explained further in the Future Work section of this paper.

MOSIX Enhancements

The MOSIX team produced a set of tools, called the MExec/MPMake package that built on the kernel load information as described above. This package can be combined with GNU make(1) to spawn remote jobs across the cluster by using the load information to determine the best node on which to place the remote job. In this way, processes are distributed across the cluster and are able to do local I/O. The MExec/MPMake package consists of a small server that runs on each node of the cluster. This server acts as a proxy that creates local processes when passed the command name, argument list, etc. from the remote node. EMC sponsored the development of this package and encouraged its release under the GNU Public License (GPL).

A small client sends commands to these servers based on the node with the lowest load in a list supplied to the command. This client only ran on nodes in the MOSIX cluster. An additional, non-MOSIX dependent client was created that can send commands to a specific node in the cluster, so that the remote make can be started on the cluster from computers that are not members of the MOSIX cluster. This client currently runs on Linux, FreeBSD, Solaris, and

Windows. This is useful because users do not have to log directly into the MOSIX cluster; instead, scripts and local makefiles remotely start jobs on the cluster. Process migration is disabled so that processes stay on the nodes where they start.

File System Issues

Consistency is the responsibility of the build environment or the distributed filesystem. EMC addresses this by turning off name, data, and attribute caching in NFS. Otherwise, nodes do not see newly created files or directories, which results in inconsistent or failed builds. While this creates consistency, it also greatly hinders performance, as network traffic is much higher. Users of a cache coherent file system, like GFS [GFS], should be able to run with caching enabled.

The Test MOSIX Cluster

With the arrival of the new MOSIX tools, we built our first test cluster. The test cluster consisted of 8 VA Linux 3500 computers (1GB of RAM, 4-way 500MHz Intel Xeon processors each with 512KB of L2 cache) and one gatekeeper node that is a Compaq 550MHz Intel Pentium III with 128MB of RAM. We used the same backing store as the original LSF cluster, but replaced the original cluster's FDDI ring with a Cisco

6506 100Mbit switch with 96 ports.

The Compaq gatekeeper box is used as a non-computing member of the cluster; all MOSIX build jobs are submitted through this gatekeeper. Since all MOSIX nodes are peers, this is not a necessary step. However, it creates a nice place to keep track of all incoming jobs, run monitoring software, or control access to the cluster without wasting an expensive machine. Note that if this node is removed, jobs can be sent directly to the computing members of the cluster.

On this test cluster, the total time for a single user to build alone on an idle cluster was about 9 minutes (537 seconds).

The next test was to have multiple users running jobs at the same time. Running with six different users concurrently, the time for each user to complete their build was 26 minutes (1560 seconds). Note that all users were slowed down equally, *i.e.* they all shared the load of the cluster equally.

The Production MOSIX Cluster

Once we proved that the test cluster worked well, we added 16 additional nodes, bringing the total up to 24 VA Linux 3500 computers (now shipping with 550MHz CPUs) (see Figure 2).
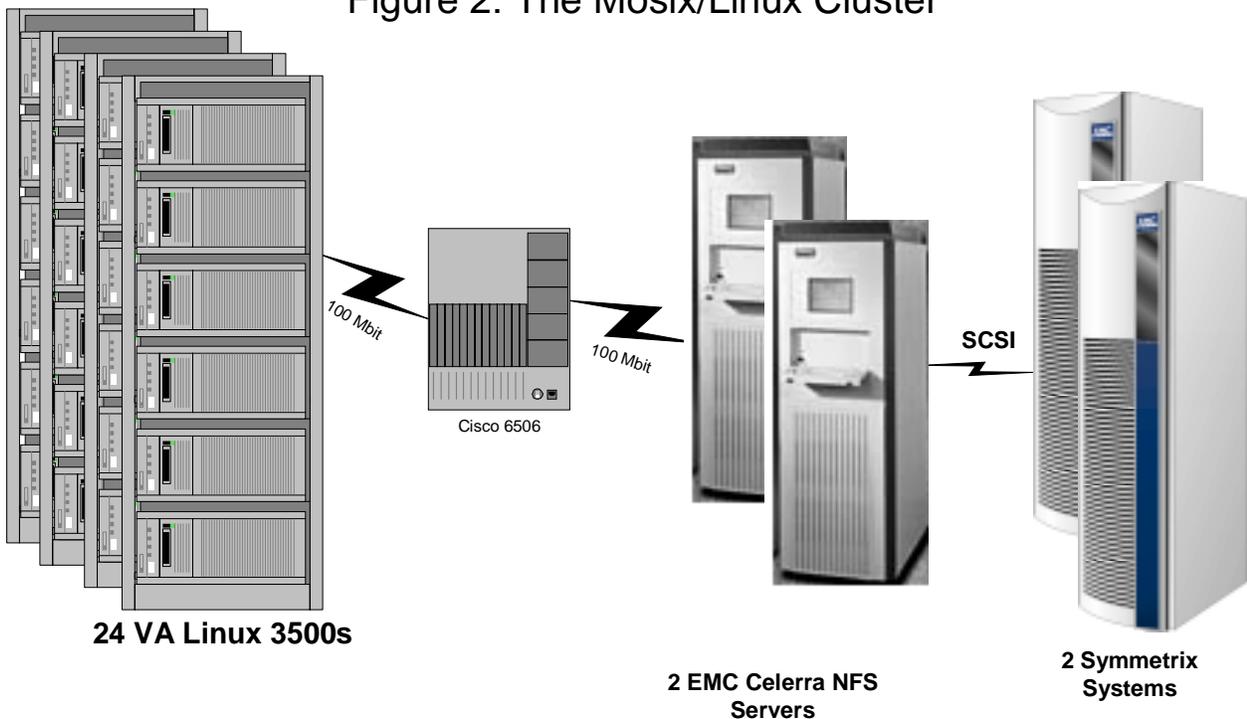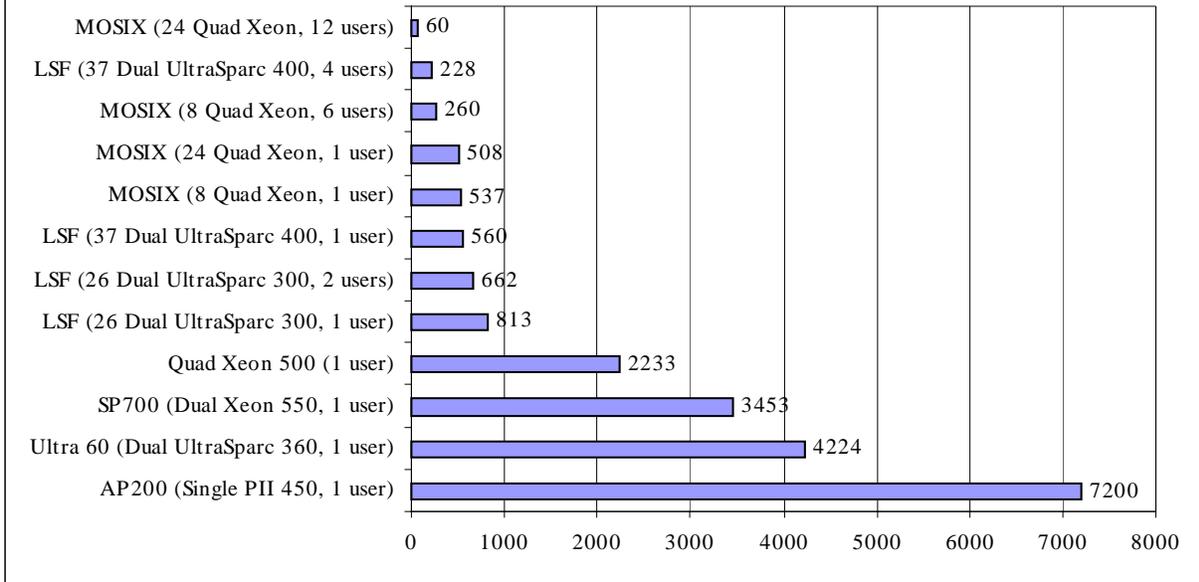

Figure 2: The Mosix/Linux Cluster

24 VA Linux 3500s

Cisco 6506

2 EMC Celerra NFS Servers

2 Symmetrix Systems

Figure 3: Average seconds per build per user

Platform	Seconds per build per user
MOSIX (24 Quad Xeon, 12 users)	60
LSF (37 Dual UltraSparc 400, 4 users)	228
MOSIX (8 Quad Xeon, 6 users)	260
MOSIX (24 Quad Xeon, 1 user)	508
MOSIX (8 Quad Xeon, 1 user)	537
LSF (37 Dual UltraSparc 400, 1 user)	560
LSF (26 Dual UltraSparc 300, 2 users)	662
LSF (26 Dual UltraSparc 300, 1 user)	813
Quad Xeon 500 (1 user)	2233
SP700 (Dual Xeon 550, 1 user)	3453
Ultra 60 (Dual UltraSparc 360, 1 user)	4224
AP200 (Single PII 450, 1 user)	7200

On the next test, all 24 nodes were used. An interesting effect, as shown in Figure 3, was that the build time remained mostly constant for a single user, yielding a slightly improved build time of 8.5 minutes (508 seconds). Further investigation using a network analyzer revealed that the 100Mbit connection from the switch to the NFS data mover was completely overwhelmed by the compute nodes. We set up a network analyzer to monitor traffic on the switch. The switch showed that once we started the build process, the link from the switch to the data mover would become 100% utilized, while the links to the individual nodes would be at most 5% to 10% utilized. Upgrading the network is the next step to overcoming this limitation.

The most noticeable improvement occurred when multiple users would build simultaneously. With 12 users building concurrently on different NFS data movers, the time for each user to complete a build increased from 8.5 minutes (508 seconds) to about 12 minutes (715 seconds). Although this is about 40% slower than the time spent building on an idle cluster, it is still faster than a single user running a build on the original LSF cluster when idle. If two or more users share an NFS data mover, those users do see significant decreases in performance as they are contending for the same physical device.
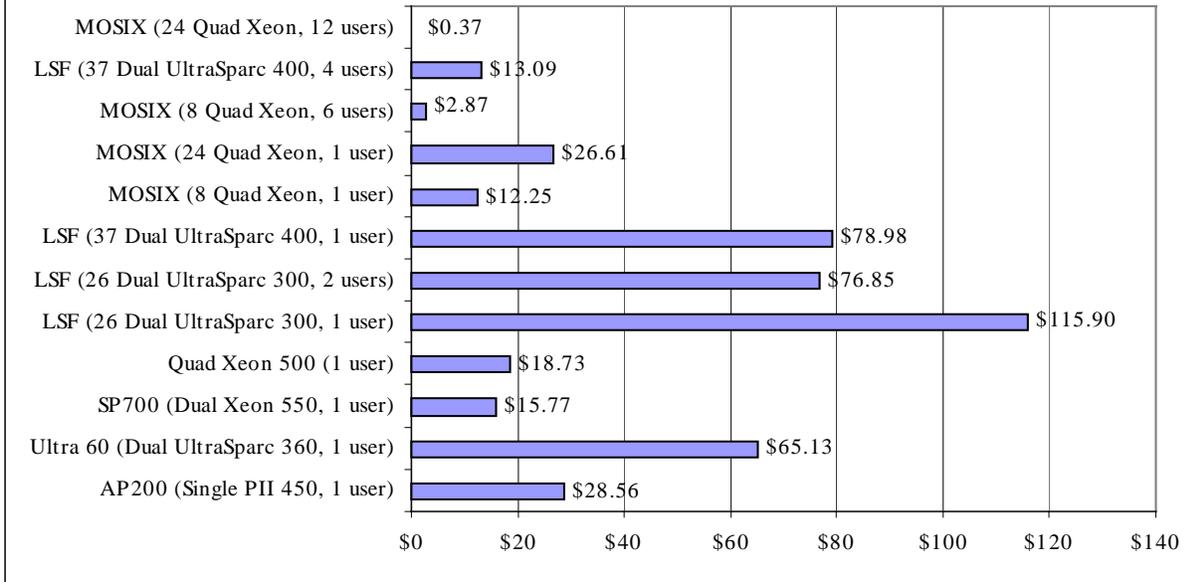
Performance Analysis

In Figure 3, we detail the average build time in seconds per user for the various platforms tested. The numbers were achieved by taking the time in seconds, from start to finish, of the user's build jobs and then dividing by the number of users building concurrently to get the average build time per user.

As Figure 3 shows, we were able to gain approximately 1100% improvement in performance when 12 users are using the Production MOSIX Cluster as compared to the two users using the LSF Cluster. Note that the original LSF cluster allowed a maximum of two users at once. This was a major restriction since we potentially have upwards of 20-25 developers trying to build concurrently.

One of the bottlenecks we observed (as shown before) was the network connection to the NFS data movers being saturated by the compute nodes in the cluster. Preliminary testing on one NFS data mover shows that upgrading the link between the switch and the data mover to Gigabit Ethernet reduces build times by approximately 23%. Upgrading this link and the data mover to its newest release reduces build times by approximately 42%. Our best times were measured with the Gigabit Ethernet, upgraded data movers, and with a striped disk file system behind the NFS data

Figure 4: Average cost per build per user (normalized to 1440 builds daily)

Configuration	Cost
MOSIX (24 Quad Xeon, 12 users)	$0.37
LSF (37 Dual UltraSparc 400, 4 users)	$13.09
MOSIX (8 Quad Xeon, 6 users)	$2.87
MOSIX (24 Quad Xeon, 1 user)	$26.61
MOSIX (8 Quad Xeon, 1 user)	$12.25
LSF (37 Dual UltraSparc 400, 1 user)	$78.98
LSF (26 Dual UltraSparc 300, 2 users)	$76.85
LSF (26 Dual UltraSparc 300, 1 user)	$115.90
Quad Xeon 500 (1 user)	$18.73
SP700 (Dual Xeon 550, 1 user)	$15.77
Ultra 60 (Dual UltraSparc 360, 1 user)	$65.13
AP200 (Single PII 450, 1 user)	$28.56

mover. This configuration reduces our current build times by 51%, making the cluster more than twice as fast.

Cost Analysis

For the original LSF cluster, total cost is approximately $660,000, with software costs of $61,141.25 up front and an annual cost of $16,835 for support licenses for a two year life span, as shown in Figure 4. The Production MOSIX cluster cost is approximately $390,000, with no recurring software costs, for a two year life span, also shown in Figure 4. In addition, having the full source code of the system and tools enables us to debug and fix problems as they arise. Detailed cost information is shown in Appendix A.

Overall, cluster costs were reduced by 41% (about $270,000) while performance was increased at least eleven fold. In Figure 4, we use these numbers to compute the cost per build per user (assuming a two year life span for each cluster). In all cases, the chart assumes that builds run 24 hours a day. The numbers are all normalized to the equivalent of building 1440 builds per day (the amount that the fastest cluster – the Production MOSIX cluster - could sustain). Stated another way, the numbers show what the cost per build

is if you could linearly scale each solution to achieve 1440 builds per day.

Administration

The Production MOSIX Cluster administration consists of monitoring the cluster for availability. Once the cluster was turned on for general use, the cluster has needed little to no attention or intervention other than two hardware memory failures. Users can submit jobs directly from Linux, FreeBSD, Windows, or Solaris clients.

By comparison, the LSF Cluster administration consists of maintaining a license server for the LSF cluster management software and monitoring the cluster for availability. Only Solaris clients can submit jobs directly to the cluster, and each Solaris client wishing to submit jobs to the cluster must be pre-configured by the cluster administrator as a client to the cluster.

Future Work

We are currently working with the MOSIX team to explore several enhancements to the cluster. One obvious shortcoming is the lack of support for a cache-coherent, distributed file system like GFS. Working

with EMC, the MOSIX team has added this support, enabling a process to issue its I/O system calls locally, even after migration. This should allow the cluster to use its normal dynamic process migration for a larger group of I/O intensive tasks, including our distributed builds. EMC's efforts will focus on tuning the rest of the components of the system: the network will be upgraded to Gigabit Ethernet and users' directories will be striped across multiple NFS data movers to exploit EMC's high-end storage systems.

Related Work

Process migration has been an active area of research since the early 1980s, but has had limited success in the commercial world. In early systems, like the original MOSIX implementation, migration was added to existing systems by restructuring the internals of the base operating system. This approach produced a great deal of transparency: user processes could migrate freely without any requirement to link against special libraries or use special system calls. A significant drawback was the difficulty of maintaining the migration code in the continually changing host operating system.

Sprite [Douglis and Ousterhout, 1987], developed at UC Berkeley, simplified this process by introducing the concept of the home node as is used in the current MOSIX implementation. In this scheme, migrated process redirect some of their system calls back to their creation node. This retained the transparency of migration for user processes, but also had a performance impact for certain types of processes.

In the same era, work started on user-level process migration. This approach traded transparency and performance for portability. For example, little to no changes were required to port Condor [Litzkow, 1987] to a new host platform. Condor had a strong influence on the development of Utopia [Zhou et al., 1994], the academic precursor of LSF, and Loadlevelor from IBM.

The rare commercial implementations of process migration include Locus Computing's migration support in OSF1/AD for the Intel Paragon [Zajcew et al., 1993] and Platform Computing's LSF.

For a comprehensive survey of process migration, see Milojicic, et al [Milojicic et al., 2000]. In this survey, many of the seminal migration papers are presented as a collection, including the early MOSIX, Condor, and Sprite papers.

Conclusion

Our project demonstrates that it is possible to create high performance, distributed build environments from commodity hardware and open source software, such as Linux and MOSIX. In addition, this project demonstrates that collaboration between the open source community, an industrial systems group, and our system administrators works well. Together, we produced a system with an order of magnitude performance improvement at less cost than our original cluster. In the future, as software becomes more complex and requires more CPU processing power, the cost benefits will become both more apparent and more important.

Acknowledgements

References

Barak, A., Guday, S., and Wheeler, R. (1993) The MOSIX Distributed Operating System. *Lecture Notes in Computer Science, Vol. 672, Springer-Verlag.*

Douglis, F. and Ousterhout, J. (September 1987). Process Migration in the Sprite Operating System. *Proceedings of the Seventh Conference on Distributed Computing Systems,* pages 18-25.

EMC² Corporate Home Page: http://www.emc.com

Global File System Home Page: http://www.globalfilesystem.org

Litzkow, M. (June 1987). Remote UNIX – Turning Idle Workstations into Cycle Servers. *Proceedings of the Summer USENIX Conference,* pages 381-384.

Milojicic, D., Douglis, F., Paindaveine, Y., Wheeler, R., Zhou, S. (to appear in 2000). Process Migration Survey. *ACM Computing Surveys*.

Milojicic, D., Douglis, F., and Wheeler, R. (February 1999). Mobility: Processes, Computers and Agents. *Addison-Wesley Longman and ACM Press*.

MOSIX Project Home Page: http://www.mosix.org

Platform Computing Corporate Home Page: http://www.platform.com

Popek, G. and Walker, B. (1985): The Locus Distributed System Architecture. *MIT Press*.

VA/LINUX Corporate Home Page: http://www.valinux.com

Zajcew, R., Roy, P., Black, D., Peak, C., Guedes, P., Kemp, B., LoVerso, J., Leibensperger, M., Barnett, M., Rabii, F., and Netterwala, D. (January 1993). An OSF/1 UNIX for Massively Parallel Multicomputers. *Proceedings of the Winter USENIX Conference*, pages 449–468.

Zhou, S., Zheng, X., Wang, J., and Delisle, P. (December 1994). Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems. *Software-Practice and Experience*.

Appendix A

Table 1 shows the cost of the individual cluster nodes, based on the full manufacturer's list price as of February 1, 2000. Note that any possible discount was not factored into the calculations.

Table 1: Hardware prices	
VA Linux 3500 Quad Xeon - 1GB/550Mhz	$14,215.00
Compaq AP200 PII - 128MB/450Mhz	$2,085.00
Compaq SP700 Dual Xeon - 512MB/550Mhz	$5,004.00
Sun Dual Ultra 2 - 512MB/300Mhz/FDDI	$20,870.00
Sun Dual Ultra 2 - 512MB/400Mhz	$21,090.00
Sun Dual Ultra 60 - 512MB/360Mhz	$13,815.00
Cisco 6506 + 96 100BaseT ports	$46,975.00
Fore ESX-2400 + 96 100BaseT ports	$40,780.00

Table 2 shows the total costs of each of the clusters, including network and software costs, but does not include the cost of the storage (since the storage costs are constant and can be shared across clusters).

Table 2: Cluster prices (2 year life)	
LSF (26 Dual UltraSparc 300, FDDI)	$663,601.25
LSF (37 Dual UltraSparc 400, 100Mbit)	$953,036.25
MOSIX (8 Quad Xeon, 100Mbit)	$160,695.00
MOSIX (24 Quad Xeon, 100Mbit)	$390,220.00