Job Management in Grids of MOSIX Clusters

David Abramson†, Amnon Barak‡ and Colin Enticott§

† School of Computer Science & Software Engineering, Monash University, Clayton, Australia, 3800
davida@csse.monash.edu.au

‡ Institute of Computer Science The Hebrew University Jerusalem Jerusalem 91904, Israel
amnon@cs.huji.ac.il

§ CRC for Enterprise Distributed Systems Technology c/o School of Computer Science & SE Monash University, Clayton Australia, 3800
Colin.Enticott@csse.monash.edu.au

Abstract

EnFuzion and MOSIX are two packages that represent different approaches to cluster management. EnFuzion is a user-level queuing system that can dispatch a predetermined number of processes to a cluster. It is a commercial version of Nimrod, a tool that supports parameter sweep applications on a variety of platforms. MOSIX, on the other hand, is operating system (kernel) level software that supports preemptive process migration for near optimal, cluster-wide resource management, virtually makings the cluster run like an SMP. Traditionally, users either use EnFuzion with a conventional cluster operating system, or MOSIX without a queue manager. This paper presents a Grid management system that combines EnFuzion with MOSIX for efficient management of processes in multiple clusters. We present a range of experiments that demonstrate the advantages of such a combination, including a real world case study that distributed a computational model of a solar system.

1 Introduction

Commodity Clusters have become the platform of choice for cheap high performance computing [9]. Typically, these are built from standard, off-the-shelf components like commodity processors and networks, and use the Linux operating system. Two software packages that have been developed for managing computations on clusters are MOSIX [1] and EnFuzion, a commercial version of Nimrod [15] by Axceleon Inc [8].

MOSIX is an add-in to Linux that supports a single system image, and makes the cluster behave almost like a symmetric multi-processor (SMP). MOSIX provides mechanisms for migrating processes between the nodes of the cluster without the process being aware of where it is executing. Importantly, it allows processes to be migrated even when they are performing input-output and interprocess communication, and contains novel heuristics for deciding where and when to migrate processes. As a result, users do not need to concern themselves with the mapping of processes to nodes, nor how to mix processes across nodes with different speeds, since these are handled transparently by MOSIX. MOSIX does not provide any high level job queuing functions, and these need to be provided by other queue management packages. Thus it cannot actually limit the amount of work that is scheduled for execution, unlike queue management systems like EnFuzion, PBS [2], LSF [9][11] and Condor [12]. Without a separate queue manager, it is possible to overload a MOSIX cluster severely.

EnFuzion, on the other hand, is a high level queue management system developed for running large parameter sweep applications on clusters. EnFuzion allows a user to specify very large computational experiments in which the same program is run multiple times, each time with a unique set of parameters. Like MOSIX, EnFuzion determines the best node on which to run a process dynamically, but then never migrates the process once it starts running. Accordingly, it is less adept than MOSIX at accounting for variability in load. Also, Neither MOSIX or EnFuzion are able to spread load across multiple clusters.

Recently there has been much interest in building large "Grids" of high performance clusters [13]. Aggregating clusters is a natural way of achieving more power than is available in any one system, without permanently combining clusters into one machine. This requirement is not uncommon in organisations in which the clusters are owned by different departments, and in which the distribution of work is not uniform. For example, if two departments each own a cluster, then it should be possible for any one department to exploit both systems at one time providing the other department does not require their cluster at that time. On the other hand, when both departments require their machines at the same time, they behave as two totally separate systems and do not share any load. However, we have developed a separate package called the Multi EnFuzion Client (MEC) [3] that distributes parameter sweep experiments to multiple machines in which EnFuzion is installed. The MEC is based on our previous work with the Nimrod/G package [14][15], and exploits many of the same approaches to scheduling across different systems. Unlike other general Grid systems, the MEC is able to operate across firewalls effectively by establishing its own virtual private network.

In this paper, we explore the combination of EnFuzion, the Multi EnFuzion Client and MOSIX, and demonstrate that there are significant advantages in combining them. When EnFuzion and MOSIX are combined on a single

cluster, the system exhibits the excellent workload management aspects of MOSIX, whilst providing a powerful user-level queue management system for parameter sweep applications. This combination is unique. When further combined with the MEC, the system can be shown to scale to enterprise wide Grids of homogeneous clusters. Importantly, the MEC works without using general Grid middleware like Globus [13]. This is only possible because we assume a common cluster architecture and operating system (namely MOSIX). Whilst not as general as Grid computing, this is still applicable in many practical circumstances.

The paper begins with a more detailed discussion of how MOSIX and EnFuzion work, followed by a description of the MEC. Then we present results that demonstrate the advantages of combining EnFuzion and MOSIX, and we show how this extends to the MEC. Finally, we demonstrate a real parameter sweep application that is executed on machines in Australia and Israel, and highlight the performance of the system.

2 Job Management in Clusters

2.1 EnFuzion

EnFuzion is an application level package that provides a high level environment for the creation, distribution and management of large parameter sweep applications. EnFuzion is a commercial code that implements research ideas explored in the Nimrod project [15]. Parameter sweep applications are characterised by the execution of many jobs, each of which explores some part of a larger parameter space. Typically, a computational model is executed many times, whilst parameters to the model are varied. Apart from an initial scattering of files and input parameters, and a final gathering of results, the jobs are independent, can be distributed to a collection of networked processors.

EnFuzion consists of two main components, a tool called the `generator` and another one called the `dispatcher`. The `generator` takes a skeleton description of a computational experiment (called a plan file), and builds a file (called a run file) that indicates how the model is to be run, and what the actual parameter combinations are. The `dispatcher` takes a run file, and schedules and runs jobs on the nodes that are available at the time. It does this by sending each job to the next available processor, using a first-come first-serve allocation strategy. Importantly, once a job starts execution on a node, it remains there until it completes. As we will see later, this simple allocation strategy can generate pathologically bad assignments when varying execution times are mixed in the one run, or when multiple users launch a number of experiments at the same time, and in which the different experiments have jobs of varying execution times.

EnFuzion does not assume that the processors share a file system, and so files are copied from the machine that launches an experiment (called the root machine) to the computational nodes. The dispatcher arranges for the files to be copied before a job starts, and then copied back after it has completed.

Whilst EnFuzion handles a number of otherwise independent jobs on a set of otherwise separate processors, it presents a view to the use of a single experiment running on a single, very high performance, computer. This attribute has made it very popular for use by scientists and engineers who wish to utilise high performance clusters, but who do not wish to write parallel programs. Accordingly, EnFuzion has been used to support a wide range of applications [16].

2.2 MOSIX

MOSIX is a software package that enhances Linux with cluster computing capabilities. The core of MOSIX includes adaptive management algorithms and a preemptive process migration mechanism that transforms the cluster into a single system parallel computing environment, almost like an SMP.

The algorithms in MOSIX support load-balancing [5], memory ushering [6], parallel I/O [7] and cluster-wide file operations [7] These algorithms monitor uneven resource usage among the nodes and if necessary, assign and reassign processes (automatically) among the nodes in order to continuously take advantage of the best available resources. The MOSIX algorithms are geared for maximal overall performance, overhead-free scalability and ease-of-use.

The granularity of the work distribution in MOSIX is the Unix process. In MOSIX, each process has a unique home-node (where it was created), which is usually the login node of the user. The system image model is a computing cluster in which every process seems to run at its home-node and all the processes of a users' session share the execution environment of the home-node. Processes that migrate to a remote (away from the home) node use local (in the remote node) resources whenever possible but continue to interact with the user's environment by forwarding environment dependent system-calls to the home-node.

MOSIX supports preemptive (completely transparent) process migration, which can migrate almost any process, any time, to any available node. After a process is migrated, a link layer at the remote node intercepts all its system-calls. If a system-call is site independent, it is executed at the remote node. Otherwise, the system-call is forwarded to the home-node, where it is executed on behalf of the process. The above scheme is particularly useful for CPU-intensive processes. The next section describes a mechanism to optimize the performance of processes with intensive I/O and/or file operations.

Unlike most network file systems, which bring the data from the file server to the client node over the network,

the MOSIX algorithms attempt to migrate the process to the node in which the file resides.

Usually, most file operations are performed by a process on a single file system. The MOSIX scheme has significant advantages over other network file systems as it allows the use of a local file system. Clearly this eliminates the communication overhead between the process and the file server (except the cost of the process migration itself).

We note that the process migration algorithms monitor and weigh the amount of I/O operations vs. the size of each process, in an attempt to further optimize the decision whether to migrate the process or not.

3 Grids of Clusters

We have built a tool called the Multi-site EnFuzion Client (MEC) that implements some of the scheduling heuristics in our previous research tool, Nimrod/G [14][15], but uses the EnFuzion dispatcher to launch jobs on individual clusters. Unlike Nimrod/G, which uses the Globus middleware layer, the MEC uses standard internet protocols like TCP/IP and tools like SSH to communicate and launch services on remote machines.

The MEC operates by launching the EnFuzion dispatcher on each cluster, as shown in Figure 1, and then balancing the jobs across them. Because many clusters are secured behind firewalls, the MEC establishes a secure connection using SSH [4]. It then sets up a number of tunnelled sockets so that all traffic passes through one port, obviating the need to punch holes in the firewall. This is in contrast to most Grid middleware layers like Globus that do not work well when firewalls are in place, which is a huge problem in Grid computing. Our solution effectively builds a virtual private network.

The MEC uses a simple heuristic which attempts to ensure that each cluster has the correct number of jobs in their queue so that all clusters to finish within a short time of each other. The algorithm measures the rate that the various dispatchers are consuming jobs, and then allocates new jobs accordingly. For example, if one cluster is consuming jobs at twice the rate of another, the former will be given more jobs to process. Job allocations are maintained by the MEC by moving waiting jobs between each of the local dispatchers' queues. This is done until all the queues are empty.

A Java applet, which is invoked from a standard browser, is available for viewing the status of the experiment at any stage during the run, and is shown in Figure 2. This is a major enhancement over EnFuzion's GUI which required that the launching node must be using X-windows and the X-windows client must remain alive through the duration of the run. The Java applet will also allow multiple people to view the run.

Using the MEC we have been able to run experiments across a number of otherwise independent EnFuzion based clusters, achieving a Computational Grid using standard commercial software.

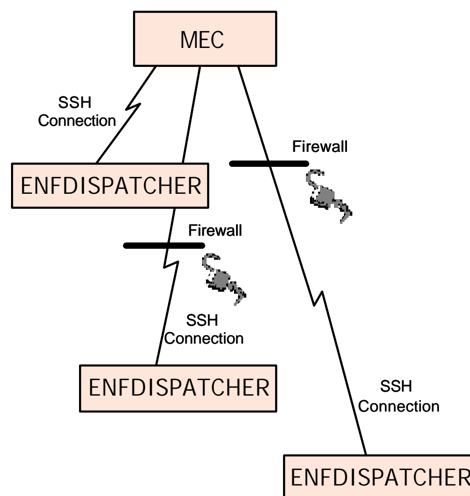
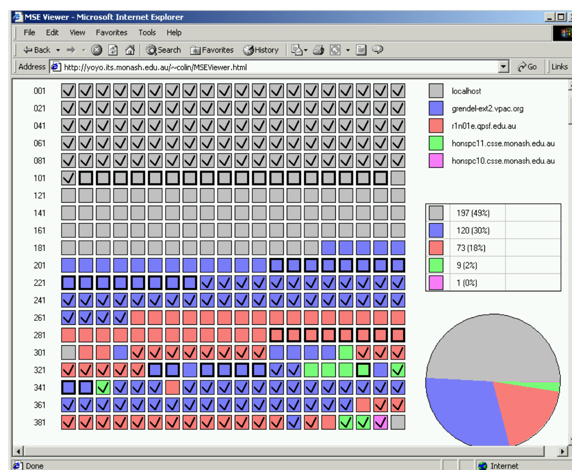


Figure 1 - Architecture of the MEC

Figure 2 - MEC progress applet



4 Combining EnFuzion with MOSIX

Combining EnFuzion and MOSIX yields a powerful platform, where EnFuzion generates, allocates and queues jobs to the cluster and MOSIX manages and optimizes the load distribution between nodes within the cluster. In particular, EnFuzion benefits from MOSIX's ability to perform preemptive process migration, and MOSIX benefits from a queue management system.

Although EnFuzion initially distributes the processes evenly across the cluster, it cannot determine how the processes will behave once started. For example, a process with one set of parameters might have very different memory requirements to ones running with a different set of parameters. In most cases, this is nondeterministic and two processes of large memory

requirements may end up on a node while another node has two processes that have little memory requirements. This may cause swapping on a node and affect the total performance of the cluster. With MOSIX's preemptive process migration, such processes could be moved to other nodes to reduce swapping. Furthermore, MOSIX will balance the cluster's load if it is not even. This will happen either while an EnFuzion run is finishing (some nodes may become idle) or the CPU speeds of the cluster's nodes are different.

Augmenting this arrangement, in which MOSIX is used to balance the load within the cluster and EnFuzion is used to queue jobs for the nodes, with the MEC, produces a system that has the ability to distribute jobs across a Grid of clusters, and then to optimize the load within each cluster. In the next section, we present the performance characteristics of such an arrangement.

5 Performance Results

This section presents the performance of several test Grid configurations, ranging from single clusters, to those which are distributed across a campus, across the metropolitan area and across continents. In each case, we executed simple tests to demonstrate the benefits of combining EnFuzion and MOSIX in the way described in the last section. The last part of the section presents tests using a real world application that simulates supernovas.

Initially, we used 2 clusters that were in the same room and were connect by a single 100Mb/sec Ethernet.

Cluster 1: four nodes, each with a dual Pentium 500MHz processor, 256MB RAM and a local disk, connected by a 100Mb/Sec hub.

Cluster 2: four nodes, each with a dual Pentium 800MHz processor, 256MB RAM and a local disk, connected by a 100Mb/Sec switch.

We executed two types of tests, one with unpredictable execution times and the other with unknown memory sizes, as follows:

CPU Test: Sixteen CPU intensive processes, with varying execution times and random arrival order. Eight processes had 1000 execution units, four processes had 200 and the remaining four had 100 execution units.

Memory Test: Eight memory intensive processes, with fixed execution times and random sizes ranging from 10MB to 200MB.

Hybrid Test: 128 CPU intensive processes, with fixed execution times and fixed size of 16MB.

None of the tests involved file I/O, unless swapping occurs in the memory test due to an allocation of processes that exceed the memory size. The results in the following tables are the completion times of all the processes, averaged over five executions.

5.1 Standalone EnFuzion and MOSIX clusters

We begin by comparing the performance of the standalone EnFuzion and MOSIX packages when each package was executed independently. In EnFuzion, the node "job limit" was set to two (which limits each CPU to one process), causing the remaining processes (if any) to queue. In MOSIX, all the processes were launched simultaneously from one node.

Table 1 shows the execution times (in Sec.) and the Standard Deviation (S.D.) of the CPU, Memory and Hybrid tests on Cluster 1 and on Cluster 2 under each of the two packages. As expected, the adaptive process management policy of MOSIX consistently outperformed the static job placement scheme of EnFuzion when the number of jobs is about the same as the number of processors. For example, the relatively long execution times of the memory test by EnFuzion are due to paging resulting from poor initial allocation of processes to nodes. Whilst it is possible to assist EnFuzion in making a better allocation, it requires some modification and tailoring of the application code and some experiments have nondeterministic memory requirements. MOSIX, on the other hand was able to correct such allocations using its "memory ushering" algorithms [6]. However, the Hybrid test highlights the behaviour of MOSIX when the number of jobs exceeds the number of processors dramatically. In this test MOSIX was not able to run the experiment because the memory requirements exceeded the available swap space. This test clearly highlights the need for a queuing system with MOSIX. We did not bother running this test on both clusters.

Table 1. Standalone EnFuzion and MOSIX times

Test	Package	Cluster 1		Cluster 2	
		Time	S.D.	Time	S.D.
CPU	MOSIX	921	1	580	1
CPU	EnFuzion	1300	121	970	87
MEMORY	MOSIX	316	58	303	133
MEMORY	EnFuzion	2976	3987	3568	3561
Hybrid	MOSIX	Would	not run	Not done	
Hybrid	EnFuzion	784	3	Not done	

5.2 Execution on a single cluster

As shown in 5.1 EnFuzion has a queuing capability that controls the number of processes that are dispatched to the nodes of a cluster. MOSIX, on the other hand is better suited to manage already allocated processes throughout the execution. By combining the two packages, one can use EnFuzion to dispatch a predetermined number of processes to a cluster, then to use MOSIX to manage the processes within a cluster, to best utilise the resources throughout the execution.

To check this idea in a single cluster, we used EnFuzion to launch the processes and compare the results with MOSIX enabled or disabled. For the CPU test, the EnFuzion "job limit" was set to four (two processes per

CPU) while the memory test was executed with a job limit of two (one process per CPU, as in the previous test).

The results, presented in Table 2, show that the execution times of EnFuzion with MOSIX enabled are 26-2500% better than the times with MOSIX disabled. We note that the large standard deviation (S.D.) of the EnFuzion with MOSIX disabled is due to EnFuzion allocating the initial jobs based on the order and timing of the nodes launching (effectively random).

Table 2. Execution times (Sec.)

Test	EnFuzion with MOSIX	Cluster 1		Cluster 2	
		Time	S.D.	Time	S.D.
CPU	Enabled	917	1	578	2
CPU	Disabled	1157	84	958	235
MEMORY	Enabled	290	68	205	67
MEMORY	Disabled	3260	67	5203	3516

Without MOSIX, we can see that the EnFuzion times for the same CPU tests with "job limit" of four (CPU test with MOSIX disabled in Table 2) were up to 12% faster than with a "job limit" of two (CPU test with EnFuzion in Table 1). The CPU tests indicate that a shorter completion time could be obtained by running multiple processes per processor. This will be discussed in 5.3.

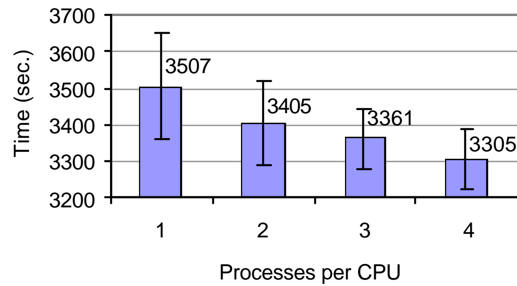
In the remaining of this section, we executed only the CPU test. All the experiments were conducted by using the MEC with one EnFuzion dispatcher per cluster and with MOSIX enabled in each cluster.

5.3 How many processes per CPU?

Comparing the times of the CPU test of EnFuzion (without MOSIX) in Table 1 and Table 2 shows an improvement by increasing the number of processes per CPU. Increasing this further without MOSIX could cause nodes that have accumulated longer processes to finish later and thus to a longer completion time. The MOSIX's ability to migrate processes removes this problem and thus the possibility to overload the cluster.

In the next test, we increased the total number of processes to 128, adjusted the number of processes per CPU in each benchmark from 1 to 4 and executed on Cluster 2. Figure 3 shows the performance gained by increasing the load of the cluster. From the figure it can be seen that overloading each node with more than one process actually resulted in increasingly better performance. Obviously, this effect does not continue beyond a certain small number of processes. One explanation for this phenomenon is that having a greater number of processes executing ensures that the longer processes had shorter remaining execution by the time the load reduces to 1.

Figure 3. Varying load on a MOSIX cluster

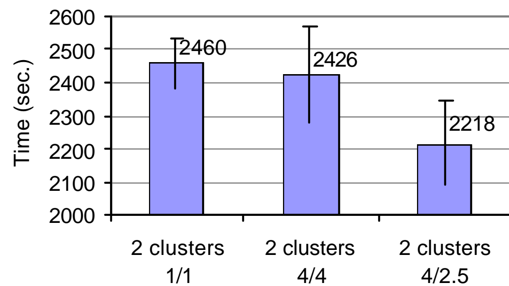


5.4 A 2-cluster Grid

In this experiment, we executed the CPU tests on Cluster 1 and Cluster 2 (totalling 8 nodes) forming a "same floor", 2-cluster Grid. Each cluster was managed by its own MOSIX and the Grid allocation of processes was done by the MEC and the EnFuzion dispatcher. We executed this experiment with 1 process per CPU, 4 processes per CPU and with a weighted combination of 2.5 processes per CPU on cluster 1 (5 processes per node) and 4 on cluster 2.

Figure 4 shows a slight improvement moving from 1 process per CPU to 4 processes per CPU. This was expected from the results of the previous section. During these tests, it was observed that the faster cluster finished long before the slower one. Consequently, we implemented a more sophisticated load balancing algorithm, in which the load was allocated relative to the CPU power of the clusters, allowing them to complete at approximately the same time. As a result of this, the jobs completed about 9% earlier.

Figure 4. Different loads on two MOSIX clusters



Comparing Figure 3 with Figure 4, shows the speed increase by adding the 500MHz cluster to the 800MHz cluster in a small Grid. This combinations of machines has a lower bound for the execution time of 2033 seconds, however, due to the granularity of the processes execution time, this cannot be achieved. Using a balanced load gave the closest result.

5.5 A 5-cluster grid

At this stage, we introduce three new clusters:

Cluster 3: Four dual CPU 700MHz nodes connected by 1000Mb/sec. switch. This cluster was located on another Monash campus, 10km away.

Cluster 4: Eight single CPU nodes of varying speeds, (four 1GHz, three 750MHz and a 500MHz node). They were all connected by a 100Mb/sec. switch. These machines were located at the Victorian Partnership for Advanced Computing (VPAC), across town 20km away.

Cluster 5: Eight single CPU nodes, all with 1GHz CPUs and connected by 100Mb/sec. switch. This cluster was located at the Hebrew University of Jerusalem, over 13,500 km away - on the other side of the world.

Figure 5. Increasing the number of clusters

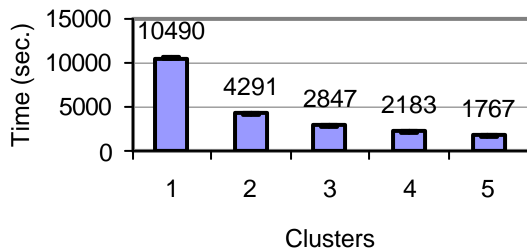


Figure 5 shows the gain achieved by adding clusters to help process the jobs. Increasing the number of CPUs to work on the problem reduces the total time, in spite of the communication delays between the clusters. By balancing the number of executing processes on each of the cluster, each cluster would finish processing at a similar time, producing the quickest complete results. These results average 338 seconds over the theoretical best. This value is also half the average duration the execution time process on the slowest cluster proving that the increase of time is caused by the granularity of processes.

5.6 A real-world test

The tests performed so far are designed to explore theoretical boundaries of cluster and Grid computing. In this section we tried a real world application. This experiment uses a code developed by Dr Kurt Liffman from the Centre for Stellar and Planetary Astrophysics (CSPA) at Monash University [17]. "The code models the early solar system, where the protoSun is surrounded by an accretion disc, which we call the solar nebula. It is thought that the planets formed from the solar nebula approximately 4.56 billion years ago. From the magnetic interaction between the Sun and the surrounding disc, a high speed, bipolar jet flow (speed ~ 200 km/s) is formed from the inner regions of the disc. Such jet flows are ubiquitous amongst early forming stellar system. The typical structure of the jet flow is that two jets are formed in the inner region of the disc, where the jet flows are pointing in a direction perpendicular to the disc. About 500 small silicate and metal spheres (size ~ 0.1 mm) are injected into the jet flow and their subsequent motion is monitored.

Each experiment executes this program 1100 times exploring over half a million particles. Typically, such particles are ejected by the jet flow and move across the accretion disc. Some particles re-enter the accretion disc, while others are ejected from the system. The purpose of the code is to measure the abundance of iron to silicon in the ejected and recaptured particles"[1]. This type of computation is ideal for EnFuzion, because each computation is uncoupled and the results can be aggregated at the end.

Table 3. Resources available for real world experiment

Cluster	Location	Dist. (km)	Number of Processors	Total GHz
Monash 1	Monash Caulfield	0	16	10.4
Monash 2	Monash Clayton	10	30	21.8
VPAC	Melbourne	20	7	6.3
Hebrew Univ	Jerusalem	13,500	48	40.0

For the execution of this experiment, we used all available resources, including 23 dual processor nodes at Monash University, 7 single processor nodes at VPAC and 39 dual and single processor nodes at Hebrew University. See Table 3 for more details.

Table 4. Results from real world application

Total GHz	Resources	Execution time (hrs)	Theoretical best (hrs)	Efficiency
1	One computer	520	520	100%
38.5	All but Hebrew Univ	14.2	13.5	95%
78.5	All	6.75	6.6	98%

Executing all 1100 processes on a single 1 GHz node took about 520 hours to complete. The resources shown in Table 3 produced the results in Table 4. The theoretical best time are calculated from dividing the time to execute on a 1GHz processor by the total computational power, as measured in total GHz. We are able to do this because the application actually scales well with the speed of the processor. From the results in Table 4, we were able to achieve close to the approximated best time. Once again, this was within the time of the average process duration on the slowest node (56 minutes).

[1] This text was supplied by Dr Liffman.

The results illustrate that the MEC was effective in distributing the load across a number of independent clusters whilst taking advantage of the MOSIX local cluster management.

6 Conclusions

In this paper we have explored the combination of two otherwise separate packages that present a view of a multi process cluster as a single computational resource. The results of a number of controlled experiments highlight scenarios in which each package is superior. Specifically, MOSIX performs better workload allocation than EnFuzion when the number of jobs is comparable with the number of processors. This is because MOSIX can reassign the processes dynamically. On the other hand, EnFuzion's queueing capability performs better when the number of jobs is much larger than the number of processors because it avoids overloading the processors. In the limit, MOSIX is unable to run experiments that require too much of a single resource, like swap space. When combined, we achieve the best of both systems.

By using the Multi EnFuzion Client we are able to combine clusters into a computational Grid. Using this we reduce the execution time of a large computational experiment that would have taken over 500 hours, to one that ran in under 7 hours, using resources distributed across the globe, and running at an efficiency of 98%. This demonstrates the combination is practical and useful for real computational science experiments.

The work in this paper has highlighted the potential for queue managers to benefit from information that MOSIX normally keeps in the execution of a process. For example, we are interested in exploring the idea of exporting dynamic parameters such as memory usage, processor utilisation, etc, to improve the allocation of jobs to clusters in the first instance.

Further, EnFuzion, like other queue management systems, allocates processes to processors so that the node is not overloaded. However, in section 5.3 we demonstrated that this under states the amount of work a machine can do, and that sometimes it is possible to place more processes on the machine. In our work, we were able to determine the optimal number of processes per processor through experiment, and then we instructed the MEC on how to allocate processes. In the future, this should be done automatically. Accordingly, we plan to produce a queue scheduler that works more closely with MOSIX in allocating.

Another area of interest is in using MOSIX's process migration techniques for deciding when, and how, a process can be migrated from one cluster to another one. This is a complex topic and requires further research.

In addition, we reported a new software tool called the Multi EnFuzion Client (MEC), which draws on research results from the Nimrod project. The MEC supports the aggregation of multiple homogeneous clusters into a Grid. Unlike Nimrod, the MEC works in a "standard" Internet environment without special Grid middleware like Globus, and without special security requirements. This is possible only because we assume a homogeneous machine base, whereas general Grids are mixes of heterogenous machines, operating systems and environemts. Further, the MEC is able to operate in the presence of firewalls unlike much other grid middleware.

Acknowledgment

We thank Glen Pringle and David Arnold and Arkady Zaslavsky at Monash University for their support, David Bannon and Sudarshan Ramachandran at the Victorian Partnership for Advanced Computing, Danny Braniss, Assaf Spanier, Ephraim Silverberg and Chana Slutzkin at Hebrew University, Kurt Liffman from the C.S.I.R.O. This project is supported by the CRC for Enterprise Distributed Systems and GrangeNet.

References

[1] http://www.MOSIX.org

[2] http://www.openPBS.org

[3] Multi EnFuzion Client, http://www.csse.monash.edu.au/~cme/mec

[4] SSH, http://www.ssh.com

[5] Barak, A., La'adan, O. and Shiloh, A., "Scalable cluster computing with MOSIX for Linux", Proc. 5-th Annual Linux Expo, Raleigh, NC, pp. 95-100, May 1999.

[6] Barak and Braverman, "Memory ushering in a scalable computing cluster", Journal of Microprocessors and Microsystems, 22 (3-4), pp. 175-182, 1998.

[7] Amar, L., Barak, A. and Shiloh, A., "The MOSIX parallel I/O system for scalable I/O performance", Proc. of the 14th International Conference Parallel and Distributed Computing and Systems (PDCS 2002), Cambridge, MA, pp. 495-500, November 2002.

[8] http://www.axceleon.com.

[9] Sterling, T., "Beowulf cluster computing with Linux", MIT Press Cambridge, MA, 2001, ISBN:0-262-69274-0

[10] http://www.platform.com

[11] Zhou, S., Zheng, X., Wang, J. and Delisle, P., "Utopia: A load sharing facility for large, heterogeneous distributed computer systems", Software-Practice and Experience, 23(12), pp 1305-1336, December 1993.

[12] Livny, M., High-Throughput Resource Management. In The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers, 1999.

[13] Foster, I. and Kesselman, C., Globus: A Toolkit-Based Grid Architecture. In The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers, 1999.

[14] Abramson, D., Buuya, R. and Giddy, J., "A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker", to appear, Future Generation Computer Systems. Volume 18, Issue 8, Oct-2002.

[15] Abramson, D., Giddy, J. and Kotler, L., "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?", International Parallel and Distributed Processing Symposium (IPDPS), pp 520-528, Cancun, Mexico, May 2000.

[16] Abramson, D., "Parametric Modelling: Killer Apps for Linux Clusters", The Linux Journal, #73, pp 84 – 91, May 2000.

[17] Centre for Stellar and Planetary Astrophysics (CSPA), http://www.maths.monash.edu.au/astro/