Examensarbete BEE03:07

Performance comparison of Clustering/Distributed Computing tools

Emma Roos

Blekinge Institute of Technology

June 2003

Blekinge Institute of Technology
Department of Telecommunications and Signal Processing
Examiner: Patrik Carlsson, BTH

Abstract

The Department of Telecommunications and Signal Processing (ITS) at the Blekinge Institute of Technology had several unused computers that they wanted to use in a so-called cluster. On this cluster they want to run computations and simulations, since it should go faster to do this on several computers instead of just one. A cluster consists of two or more computers that via special software spread the load created by different programs between the machines in the cluster. The project was about comparing two different kind of cluster systems: OpenMosix and Ganglia. OpenMosix is a kernel extension to Linux with some different tools for administrating it, while Ganglia consists of several stand alone programs divided into two main parts: a monitoring core and an execution environment.

The comparison involved: what hardware and operating systems the cluster systems worked on, how the installation and administration of them is done, how to add new machines (so called nodes) to them, what components and tools comes with them plus that a performance test of how long a certain parallel job took was made. Some information about what kind of programs that could and couldn't run on the cluster was also gathered.

The conclusion that was made were that openMosix fits best on the kind of cluster ITS wants and that Ganglia could be used to monitor the cluster.

In the end two small how-tos (one for openMosix and one for Ganglia), describing how to install and administrate the systems, were written.

Contents

List of figures

List of tables

Preface

This report is divided into six chapters: *Introduction to the project*, *Introduction to clusters, openMosix*, *Ganglia, Performance test of Ganglia and openMosix* and *Discussion and conclusions*. The first chapter consists of an overview what the project was about and how it was executed. Chapter 2 gives a basic introduction on what clusters are and describes the three basic types of clusters that exists. The third chapter is about the first of the two clustering systems described in this report: openMosix, what components it has, how to install and administrate it plus what software works and doesn't work on it. Then in chapter 4 the other cluster system, Ganglia, is described. Just as in chapter 3 the tools and components, in this case for Ganglia, is listed plus there is information about how to install and administrate the system. In this chapter there is also information on how to run programs with Ganglia's execution environment. The fifth chapter describes the performance tests that were made to compare the two clustering systems and showing the results that came from these tests. Then in the last chapter there is a discussion and conclusion on what of the two clustering systems are best for ITS plus a mentioning on what other cluster systems that exists.

There are also five appendices. In *Appendix A* the data from the performance tests are presented. *Appendix B* is the openMosix how-to with information on how to install and administrate openMosix that was going to be written as a part of the project and in *Appendix C* the configuration files for openMosix that was used on the project machines are shown. *Appendix D* and *E* have the same things as appendix B and C but for Ganglia.

Chapter 1 Introduction to the project

The *Department of Telecommunications and Signal Processing* (ITS) at *Blekinge Institute of Technology* (BTH) had several idle computers. An idea was that these computers could be put together into a so-called cluster, where users can run calculations and simulations. This project is about comparing two clustering systems to see which one of them would be best for this kind of system.

The two cluster systems that ITS wanted investigated and compared was Ganglia[1] and openMosix[2]. Initially it was MOSIX[3] that was going to be investigated instead of openMosix, but while reading about Mosix the existence of openMosix was discovered. Since openMosix is under a free license (GPL = Gnu Public License[4]) while Mosix is not, plus for some other reasons, like more tools, better documentation and more frequent releases it was decided that openMosix should be tested instead.

The items to be compared were: what hardware and operating system the clustering system could run on, how easy they are to install and administrate and which one was best fitted for a system running computations and simulations. A performance test should also be done on a test system consisting of PC's and Sun's. A how-to about the clustering system that is considered best for the task should then be written.

The test system used during the project consisted of PC's and Sun (SPARC) machines connected via 10 Mbit half-duplex Ethernet. The computers first had Linux installed on them; the PCs ran the RedHat[5] distribution (version 7.3) and the Sun machines Debian[6] (version 3.0).

After that Ganglia and openMosix was installed. Ganglia was compiled on all the machines (including the SPARCs) while openMosix was only installed on the PC's since a port for SPARC didn't exist. When the systems had been installed the different administrative and monitoring tools were tested in order to evaluate the cluster administration.

After this was done it was time to start working on the performance tests by searching for information and read about how to do performance testing, looking up what benchmark or benchmarks that should be used and possible tools to see the performance. It was decided to use a token-ring simulation as a benchmark mainly because that it was the only program that could be found where parallel tests on both Ganglia and openMosix could be made.

Finally a conclusion on what system would be best for ITS was made and two how-tos was written, one for openMosix and one for Ganglia. These how-tos describe how to install, expand and administrate the clustering systems.

Chapter 2 Introduction to clusters

Traditionally it was common that companies and organizations who need large amounts of computing power used supercomputers and mainframes. These machines where usually built by just a few vendors and were very expensive. A lot of universities couldn't afford to buy these systems and therefore they started to research other alternatives. After a while the concept of clustering was born and this solution has become more used as performance of the cheaper and more available hardware has closed in on the supercomputers. Other reasons why clusters are used are: they avoid several problems resulting from system failures and they are easy to upgrade.

The idea behind clustering is that two or more computers spread the load between them, using the resources that are available on them. A computer that is part of a cluster is usually called a node and a cluster can grow in size by adding more nodes. How powerful a cluster is depends not only on what kind clustering software is used but also on how powerful the individual machines are and how fast their network connection to the rest of the cluster is.

Basically there are 3 different types of clusters: fail-over (or high availability), load balancing and high performance computing (HPC) clusters.

In a fail-over cluster there are two or more computers that regularly check that the service(s) on the other computer(s) is up and running. If a service on one of the computers breaks down another computer tries to take over.

Figure 2-1 shows a basic design of a fail-over cluster consisting of two nodes (A and B) that shares a file system. A heartbeat checks if the service is up and running. This heartbeat can either be that one node broadcasts to the LAN that it's up from time to time or that the nodes sends messages to each other asking if the other server is up. Figure 2-2 shows the same cluster as in figure 2-1 but there server A has gone down and the service on that machine have moved over to server B.

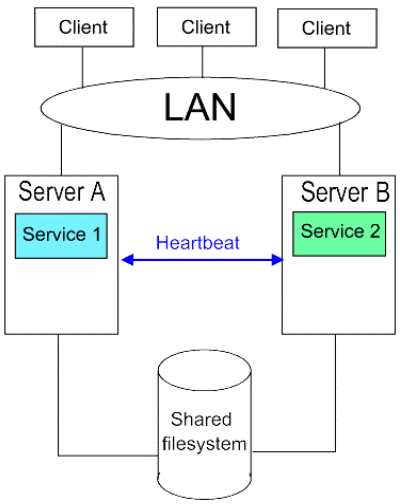


Figure 2-1: Example of a fail-over cluster.

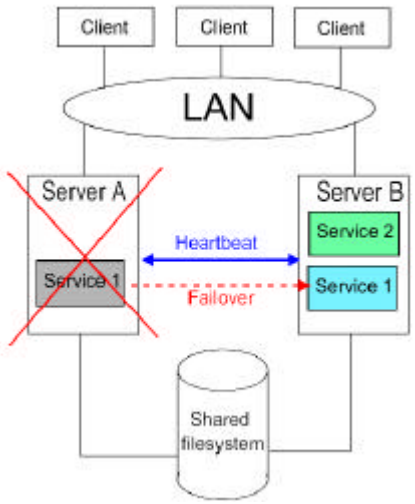


Figure 2-2: Example of a fail-over cluster there a service has gone down.

A load-balancing cluster builds on the concept that when a request for a service (for example a web-server request) comes to the cluster, it checks what machine is the least busy and then sends the request to that machine. Most of the time the load-balancing cluster also is a fail-over cluster, with the extra load balancing functionality and usually with more than two nodes.
In figure 2-3 an example of a load-balancing cluster is shown. Consisting of a monitoring server that keeps information about the load on three servers in the cluster. An incoming request for service gets informed by the monitoring server of which one of the three servers it should use (the one with the lowest load) and then the request goes to that server.

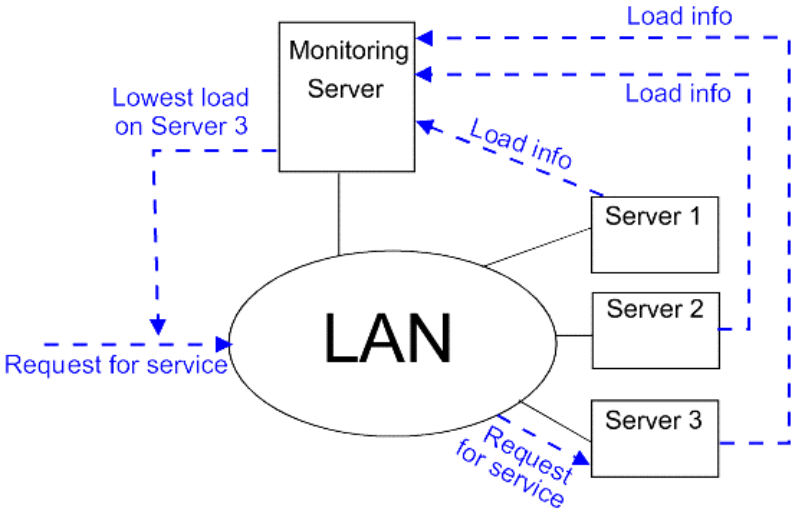


Figure 2-3: Example of a load-balancing cluster.

A high performance cluster has a kind of load-balancing feature; it tries to spread the load on the different nodes in the cluster, in order to gain performance. What mainly is done is that the programs are parallelized and different routines (that can run separately) are spread over to different nodes. This is the kind of cluster ITS wants.

Figure 2-4 shows a schematic of a HPC cluster and how it could operate. In this example a front-end node keeps information about the different nodes loads, receives jobs sent to the cluster and divides the jobs into smaller tasks if possible and then sends them to the nodes with the lowest load at the moment.

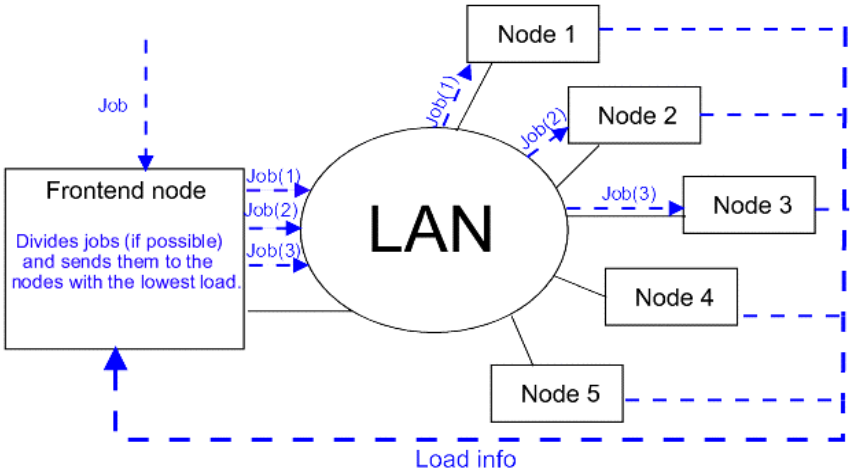


Figure 2-4: Example of a high performance cluster.

The difference between a HPC cluster and the message passing systems of RPC (Remote Procedure Call)[7, p27-34] and CORBA (Common Object Request Broker Architecture)[7, p35-37, p197-231][8] is that; they define a way for two processes to communicate with each other. When RPC/CORBA is used both the server and client programs needs the interfaces, defined by RPC/CORBA, implemented. These interfaces aren't connected to a specific programming language. HPC clusters mainly moves processes between the different nodes and there usually isn't a specific server. The programs that are using the cluster don't need to be programmed specifically to use a cluster either. Both CORBA and RPC can be used as a part of the cluster implementation though to help the different nodes in the cluster to communicate.

For more information about clusters read [9], [10] and [11].

Chapter 3 OpenMosix

3.1 Introduction to openMosix

OpenMosix is a GPL open source project that builds on Mosix. Mosix was developed at *The Hebrew University Institute of Computer Science and Engineering* and became proprietary software in late 2001. *Moshe Bar* started the openMosix project in February 2002 and it builds on the Mosix code but with several improvements and extensions.

OpenMosix is a Linux kernel extension for single-image clustering, it means that openMosix takes *n* PCs and gives users and applications the illusion of one single computer with *n* CPUs. Currently openMosix exists for Linux running on Intel x86 and Athlon processors.

OpenMosix automatically balances the load between the different nodes in the cluster according to their connection and the CPU speeds. The system administrator can also manually configure the load balancing. Nodes can join and leave the cluster without disrupting the rest of the cluster. The openMosix kernel is fully compatible with normal Linux and users' programs, files and other resources will all work as if on an ordinary Linux system without any needed changes.

For more information about openMosix history and how it works internally read [2], [12] and [13]. For more in depth reading on the components and installation instructions described in the rest of chapter 3 read appendix B and [12].

3.2 Components and tools

3.2.1 Components

OpenMosix has three main components: process migration, the Mosix File System (MFS) and Direct File System Access (DFSA). These components are described below.

Process migration

When a system runs openMosix a process can be started on one node but it will actually run on another node in the cluster: the process has migrated. Migration means that a process is split in 2 parts, a user part and a system part. The user part can then be moved to a remote node that is faster or have lower load. While the system part will stay on the node that started the process, the so-called unique home node (UHN). The system part is sometimes called the deputy process and takes care of resolving most of the system calls. The communication between the user- and system-parts are taken care of by openMosix. Not all processes can be split, or migrate. For more information about what programs that can spread over the cluster read chapter 3.5.

The Mosix File System (MFS)

MFS is a feature in openMosix that allows the user to access remote file systems in a cluster as if they are locally mounted. The file systems of the other nodes in the cluster can be found mounted under the directory /mfs. For example the files in /var on node 2 can be found in /mfs/2/var on all the machines in the cluster.

Direct File System Access (DFSA)

OpenMosix provides MFS with the DFSA-option. This option allows remote processes to perform some file system calls locally rather than sending them to their home node.

3.2.2 Userspace-tools

Userspace-tools is a package of different tools for openMosix. There are a number of useful programs:

omdiscd is a tool to auto-discover new nodes.
mps and *mtop* are openMosix aware version of the Unix tools ps and top.
migrate is a command to manually migrate processes.
mosmon is a terminal program to monitor the status of the different nodes.
mosctl is openMosix main configuration utility.
mosrun is a tool to run a command with certain openMosix settings, for example: make the command run on a certain node. There are several scripts that run the mosmon program with certain settings.
setpe is a manual node configuration utility.
tune is a calibration and optimization utility. (This tool was being totally rewritten when this project was made and were therefore not in the latest releases of the userspace-tools that existed at the time.)

The RPM[14] that is used to install these tools also changes some configuration scripts on the computer so that openMosix runs properly. It also adds an init.d script for openMosix so openMosix is started when the computer is rebooted. This script needs to be disabled and exchanged with omdiscd if auto-discovery is going to be used.

3.2.3 openMosixview

OpenMosixview[15] is a cluster-management GUI for openMosix. It contains five applications for monitoring and administration of the cluster. These applications are:

OpenMosixview	The main monitoring and administration application.
OpenMosixprocs	A process-box for managing processes on the different nodes.
OpenMosixcollector	A daemon that collects and logs cluster plus node information.
OpenMosixanalyzer	Analyses the data collected by openMosixcollector.
OpenMosixhistory	A process history for the cluster.

All these applications are accessible via the main application and the most common openMosix-commands are also accessible in the same program. Programs with different openMosix settings can also be started in openMosixview. There are priority-sliders for each

node so that a system administrator can set the load balancing priorities for the different nodes manually. OpenMosixview has been adapted to work with auto-discovery of nodes.

To be able to administrate openMosix via openMosixview the cluster needs to either have rlogin and rsh or ssh installed. These have to be configured so that the user root can log on all the nodes from the machine running openMosixview without using a password (rlogin and rsh using the .rhosts file and ssh using identity keys and the authorized_keys file). For further security when using ssh a so-called pass-phrase can be set for the identity keys. If the pass-phrase is set, then the program ssh-agent has to be started before it is possible to log on to the nodes without a password. The QT[16] libraries version 2.30 or above are also needed to install openMosixview. More information about openMosixview exists in appendix B and [15].

3.3 Installation and adding nodes

There are two ways to install openMosix on a machine; either compiling it or using the RPM-packages.

3.3.1 Compiling openMosix

If it's decided to compile openMosix then a pure vanilla kernel-source[17] is needed. What also need to be thought of is to use the right openMosix version depending on the kernel version. Do not use the kernel source that came with the Linux-distribution because it won't work. When the kernel source and the openMosix patch have been downloaded, patch the kernel, configure, compile and install the kernel.[1]

After that, download the openMosix Userland (userspace) tools and unpack the tar.gz-file, edit the configuration file, compile and copy the openmosix startup script. Create a /etc/mosix.map file that lists the computers in the cluster.

Before openMosixview is installed check that QT (and its source files) exists on the system. If it doesn't exist then download and install it. Now it's time to install openMosixview (this only needs to be done on the machine that is going to be used to monitor the cluster). Download and unzip/untar the openMosixview source tar.gz, then run the automatic setup script that comes with the code to configure and compile the different programs. After that openMosixview has been installed, the openmosixprocs program (that comes with openMosixview) should be copied to all the other nodes' */usr/bin* directory so that the processes on these nodes can be administrated via openMosixview. Finally the user root's settings for SSH on the different nodes need to be configured so that a password isn't needed to ssh to the different nodes. This is needed so that openMosixview can change the different openMosix settings on the different nodes.

[1] When trying to compile the kernel in the project there were problems getting a functional kernel and therefore openMosix was installed with the RPM instead.

3.3.2 Installing openMosix with RPMs

Installing openMosix with RPMs (on a RedHat system) is a little easier than compiling it manually. To install the kernel and userspace tools download the two RPMs for them and then use the rpm command to install them (both RPMs should be installed on the same command line, see appendix B for details). After that edit the /etc/mosix.map file or do the changes needed to use the auto-discovery system and openMosix is up and running.

When installing openMosixview make sure that QT is installed, if it isn't get the RPMs needed to install it and then use the rpm command to do this. After that just download the openMosixview rpm and install it with the rpm command on the node that you want to use to monitor the system. Then copy the program openmosixprocs to all the other nodes in the cluster and finally configure SSH for the user root so that a password isn't needed when the nodes try to connect to each other with that protocol.

3.3.3 Adding nodes to the cluster

There is two ways to add nodes depending on if the auto discovering daemon is used or not. If the auto-discovering daemon is used it's just to install openMosix with auto-discovering on the new machine and then it will be automatically added. Otherwise the machine has to be added to the file /etc/mosix.map on all the nodes in the cluster.

3.4 Administration

The configuration of the cluster exists in files lying in subdirectories in the /proc/hpc directory on the nodes. The flat files in the subdirectories can be changed both manually and via the userspace tools. In this directory there are 5 subdirectories:

admin Files for setting the different settings that openMosix has. Some files just tell if a setting is on or off for example MFS, others set different variables for example how fast the node should collect load-balancing information (decay statistics).
decay Keeps the decay statistics settings.

info Keeps information about the computer in binary format
nodes Have files with system information about the different nodes on the cluster.
remote Keeps information about remote processes that has migrated to the node.

There are also files with openMosix settings in the /proc/[PID] directories that keeps information about the processes on the system.

If auto-discovery isn't used then the different nodes should be listed in the /etc/mosix.map file on every node in the cluster, this file can be edited manually or with the setpe utility.

To shutdown openMosix the script /etc/init.d/openmosix which is a common inetd script should be used. For example if openMosix on the node is going to be restarted just type

/etc/init.d/openmosix restart as root[2]. OpenMosix can also be started and stopped in openMosixview.

Both the userspace tools and openMosixview do ease the configuration of openMosix.

3.5 Software that works and doesn't work on openMosix

A lot of programs can migrate to other machines on openMosix, but there are some exceptions: for example doesn't programs that uses shared memory migrate (they will only run on the machine that started the process), neither does programs using Linux pthreads (because of memory issues) or java programs that uses native threads. Most programs that do migrate on openMosix doesn't divide themselves to even the load, they just migrates to a faster machine or one with a lesser load if necessary. Certain programs do divide and spread themselves over several machines in the cluster, programs that are using pipes, java programs that uses green threads and programs written to work with PVM (Parallel Virtual Machine)[18] or MPI (Message Passage Interface)[19]. In table 3-1 are a number of useful programs listed with information if they are migrating or not with some comments. A program doesn't migrate can still run on the cluster, however it will only run on the machine that it started on just like on an ordinary Linux system.

Program	**Migrates?**	**Comment**
Apache[20]	No	Shared memory
Maple 8[21]	Yes	Computational functions for symbolic and numeric mathematics.
Mathematica[22]	No	Shared memory
Matlab 5[23]	Yes	
Matlab 6	No	Uses threads.
MySQL[24]	No	Shared memory
NS[26]	Yes	
NS2[26]	Yes	
Octave[27]	Yes	
R[28]	Yes	Needs to be patched. Change the SETJMP macro in *src/include/Defn.h* from: *#define SETJMP(x) sigsetjmp(x,1)* to: *#define SETJMP(x) sigsetjmp(x,0)*

Table 3-1: List of different programs and if they migrate or not on an openMosix cluster.

For lists on more programs that work and doesn't work with openMosix see [29] and [12].

[2] During the project Ganglia's gexec command was actually used to run that script on all the computers at the same time (for example if openMosix needed to be shutdown to test only Ganglia).

Chapter 4 Ganglia

4.1 Introduction to Ganglia

Ganglia were developed at the *University of California Berkeley* as a way to link clusters across the Berkeley campus. It grew out of the *Millennium Project*[30].

Ganglia consist of a monitoring and an execution environment. The monitoring core works on several architectures (for example Intel x86, Athlon and SPARC) and operating systems (for example on Linux, Solaris and Windows (beta)). The execution environment currently works only on Intel x86 and Athlon machines running Linux.

For more in depth reading on Ganglia, its components, tools and installation instructions described in the rest of chapter 4 read appendix D, [1], [31] and [32].

4.2 Components and tools

There are a few components and tools for Ganglia these are:

Authd is used to authenticate users and computers in the cluster with the help of RSA encryption[33].

Gexec is a program used to execute programs over the cluster. The user put the program they want to execute as an argument together with information on over how many computers they want to run the command on. They can specify the computers, which they want to run the command on, with the GEXEC_SVRS environment variable. With gexec there is a library libgexec.a that can be used to create/modify programs so they use the execution core.

Monitoring core (gmond) is a daemon to monitor the cluster. It the following tools:

> *The Ganglia Metric Tool (gmetric)* is a tool to monitor/check different host metrics that gmond doesn't measure.
> *The Ganglia Cluster Status Tool (gstat)* is a command line utility that can be used to get a status report of the cluster.

The Ganglia Monitoring C Library (libganglia) is a C(++) library for cluster developers who want to plug directly into the monitoring core to get status information.

Ganglia Meta Daemon (gmetad) collects data from different computers that runs gmond. The data is stored in a Round Robin database (RRD).

Ganglia Metad web interface presents the information generated by gmetad on a web page with the help of PHP[34] using XML[35].

Python Class and Client is for developers using the Python programming language[36] to allows their programs collect data from the monitoring core (wasn't tested in the project).

4.3 Installation and adding nodes

Just as openMosix, Ganglia's components can be installed either by compilation or by RPMs. The things to think of when installing the components is that authd should be installed before gexec, and the web interface should be installed after the monitoring core.

4.3.1 Compiling the Ganglia components

Compiling the execution environment

First a library called libE needs to be installed since authd needs this library to compile. This library is found on the Ganglia website together with the rest of the Ganglia components.
A public and a private RSA key needs to be created with the help of openSSL[37] and after that the keys should be put in the */etc* directory. After this authd can be downloaded, compiled and installed.

Now it's time to install gexec. The system will need xinetd on it for this component, so install it if the computer system doesn't already have it.
If the execution environment should be connected to the monitoring core, then the library libganglia needs to be installed before gexec is installed. Unfortunately with the newest version of the monitoring core libganglia only exist as RPMs. So if it's going to be installed on a machine that doesn't use RPM, then the alien[38] tool needs to be used. When libganglia has been installed configure gexec with a special flag that enables the connection to the monitoring core. After this compile and install gexec and last edit */etc/services* to add gexec's port.

Compiling the monitoring core

First check that the kernel supports IP multi-cast if it isn't then the kernel need to be recompiled with it included. If gmetad is going to be installed check that RRDTool[39] (a tool that that takes data from round robin databases and makes graphs), exists on the system, if it doesn't then install it. After that download, unpack, configure (either with or without gmetad and gexec support), compile and install the ganglia monitoring core files. Copy the *gmond.conf* file to the */etc* directory and gmond's startup script to */etc/init.d* and make so gmond will start up on reboot. If another multi-cast address and/or port should be used instead of the default one change the address in */etc/gmond.conf* file before you start the gmond daemon. If gmetad has been installed then create a directory */var/lib/ganglia/rrds* that will hold the round robin database. After this copy the file *gmetad.conf* to the */etc* directory and the gmetad startup script to */etc/init.d/*. Make sure so that gmetad is started on reboot and then start the daemon.

Installing the web interface

Make sure that the machine that the web interface is going to be on is running a web server with PHP, RRDTool and the monitoring core with gmetad installed. Install the Ganglia web interface by downloading its tar.gz file and copy it to the website's main web pages tree and then unpack the file there. Finally it's just to test the web interface in a web browser.

4.3.2 Installing Ganglia with RPMs

Installing the execution environment

First create and install the public and private RSA keys in the same way as described for compiling the execution environment. Check that xinetd is on the system otherwise install it. After that install the authd RPM package and then the gexec RPM package with the rpm command. The last thing that to be done is to add the gexec port in */etc/services/.*

Installing the monitoring core

Check so that the kernel runs multi-cast if not recompile the kernel with that option. If gmetad is to be installed then RRDTool also need to be on the system, if it isn't then download the RPM and install it. After that install the gmond RPM and if gmetad should be on the computer (this is needed for the web interface) then also install the gmetad RPM.

Installing the web interface

Make sure that the machine has PHP installed and that it runs a web server that can handle it. The computer should also have the monitoring core (especially gmetad) and RRD Tool installed. Now download the web interface RPM and install it.

4.3.3 Adding nodes to the cluster

Install the execution environment and the monitoring core on the new machine. Then check /etc/gmond.conf to make sure that the new machine uses the same port and multi-cast address as the rest of the computers. When the monitoring core on the new node is started the other machines should automatically notice it.

4.4 Administration

Most of the settings for the monitoring core exist in the files */etc/gmond.conf* and */etc/gmetad.conf*. The default files that comes with the RPMs/tar.gz explains (with comments throughout the files) quite well what the different settings do. Monitoring of the system is made with the monitoring core tools and the web interface.

4.5 Running Software on Ganglia

The program gexec is mainly useful if several instances of a program with the same settings should be run on several machines. It's for example prefect to start, stop and restart daemons on several machines at the same time or to run a system command like *uptime* on several machines. Information how libgexec.a can be used to modify programs to use the execution environment wasn't investigated since any documentation on the library couldn't be found.

Chapter 5 Performance test of Ganglia and openMosix

5.1 How the test was performed

5.1.1 Test computers and programs

The test was performed on three computers running RedHat Linux 7.3 connected via a 10 Mbit half-duplex LAN. The computers were 2 Celeron 1 GHz with 256 MB RAM named node1 and node3 and one Intel Pentium 300 MHz with 64 MB RAM named node2. Node1 was also a router and NAT (Network Access Translation protocol)[40] was installed on it so that the rest of the computers could use private IP addresses. This computer also ran an Apache web server that Ganglia used. The computers interaction via openMosix and/or Ganglia was then specifically configured for each test. Appendix C and E describes the test system and how Ganglia and openMosix was configured on the different test machines.

The program used for the performance test was a token ring simulation written for a course in teletraffic theory (course code: ETC014) at Blekinge Institute of Technology. The program runs several simulations, called replications, after each other and then compute different statistical variables. The test was run in such a way that the simulation divided up in smaller jobs (dependent on how many computers were in the test) that ran in parallel on the test cluster. The total number of replications for all the jobs was 600, for example a test with 2 computers ran 2 parallel jobs consisting of 300 replications each.

To get the time that each simulation took to execute the command *time* was used, this command had the accuracy of 10^{-2} seconds. The time that was used to compare the different systems was the real time in the system for the small job, of the ones started, in the simulation that got finished last. This time was used since it was that time it took for the entire simulation of 600 replications to finish.

Each test was done 30 times. The test data was then stored and the average, variance and 95% confidence interval for the different systems were calculated. The comparison between the different tests was made with the so called *approximate visual test*[41, p211-212].

The tests on the openMosix system was done by creating a small script that started all the small jobs, with the time command in front of each of them. After that openMosix got to take care of the load balancing so that the processes distributed themselves over the computers used in the test.

When doing the tests on a Ganglia system the small jobs, with the time command in front, got started with the gexec command. This command sent a job to each of the computers and the output of the simulation and the time command then got sent back to the prompt the gexec command was executed from.

When Ganglia were used together with openMosix the simulations were started in the same way as with just Ganglia but with openMosix running on the system too. Unfortunately openMosix didn't balance the jobs between the computers. This because gexec made so that

the processes started on the different machines got automatically locked to the node they got started on (The file /proc/[PID]/lock was set to 1).

5.1.2 Tests made

The performance tests were done on 4 different configurations of computers described below:

1. No cluster at all

In this test one simulation with 600 replications were run on the three PC machines. This test was mainly to see how well the clusters were compared to only having one computer to run the program on.

2. Homogeneous cluster

This cluster consisted of the two fast Celeron machines node1 and node3. The tests were made with openMosix, Ganglia and Ganglia together with openMosix.

3. Heterogeneous cluster with 2 computers

This cluster consisted of the fast Celeron machine node3 and the slow Pentium machine node2. The tests that were made were with openMosix (one test where the script starting the jobs were run on node2 and another where the jobs where started from node3), Ganglia and on Ganglia together with openMosix.

4. Heterogeneous cluster with 3 computers

This test used all the machines (node1, node2 and node3). The tests that were made were with openMosix (one test where the script starting the jobs were run on node2 and another where the jobs where started from node1), Ganglia and on Ganglia together with openMosix.

5.2 Results

In this chapter the results from the different performance tests are presented. The individual data collected during the tests can be found in Appendix A.

5.2.1 No cluster

In table 5-1 the average, variance, standard deviation and 95% confidence interval of the data from the performance tests when no cluster existed on the test computers is presented:

	On node1	**On node2**	**On node3**
Average (s)	469.58	1653.74	462.44
Variance (s^2)	0.06	1.82	0.005
Standard deviation (s)	0.24	1.35	0.07
95% confidence interval (s)	+/- 0.08	+/- 0.48	+/- 0.03

Table 5-1: Statistical values for the results from the performance tests with no cluster.

In figures 5-1 and 5-2 below are graphs of the test data gotten from the tests together with the average and the confidence boundaries.

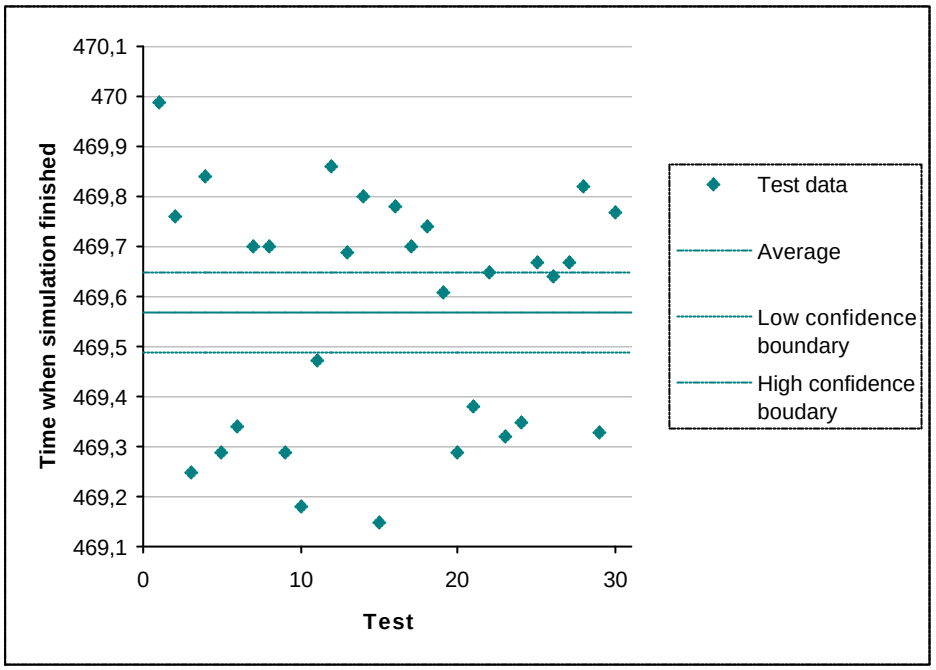


Figure 5-1: Graph of test data with average and confidence boundaries with no cluster on node1.

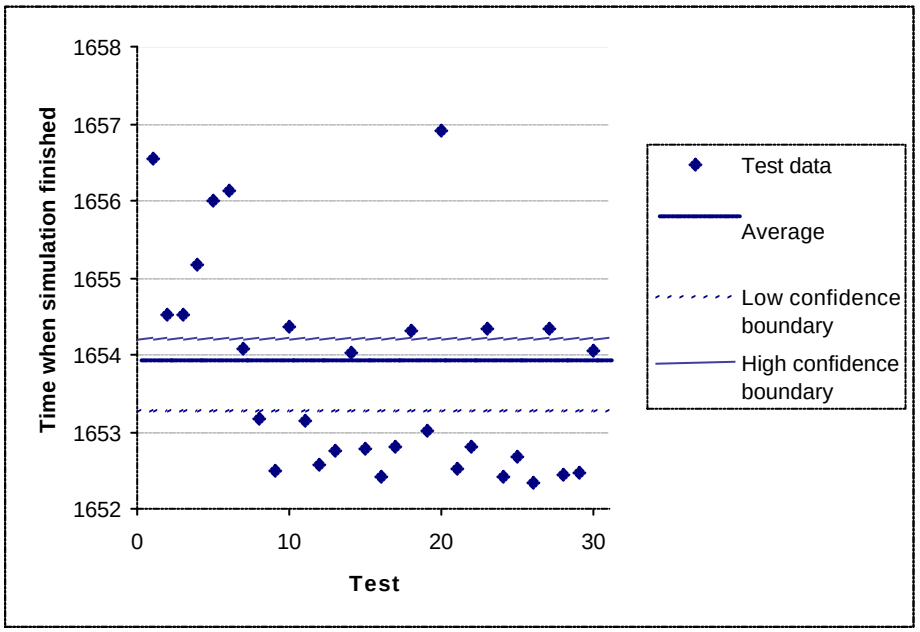


Figure 5-2: Graph of test data with average and confidence boundaries with no cluster on node2.

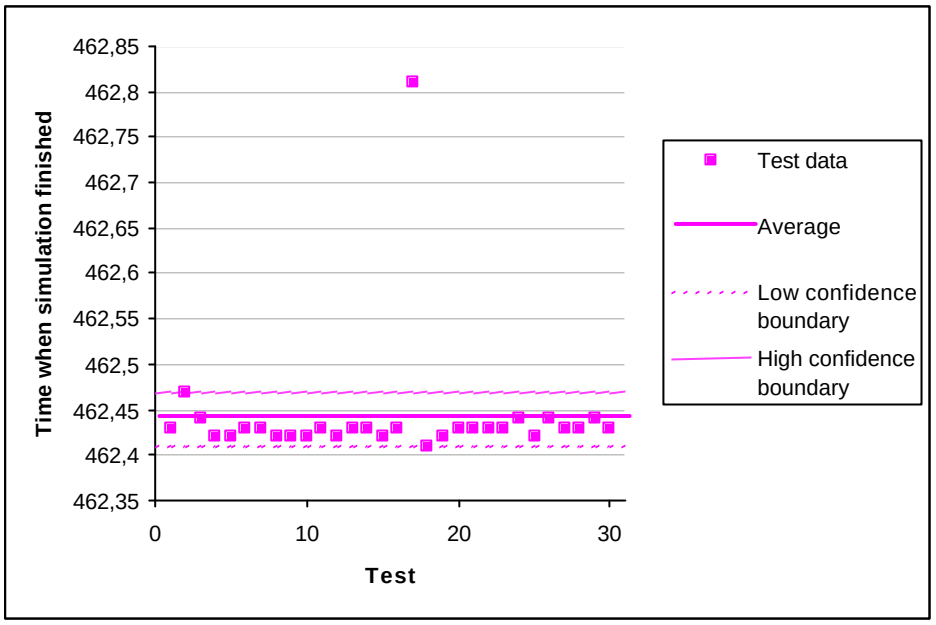


Figure 5-3: Graph of test data with average and confidence boundaries with no cluster on node3.

That the 17th sample time in figure 5-3 is higher (by approximately 3-4 ms) than the rest of the sample times probably depends on some other program on the system that took system resources while the test was running. This is also the explanation to the similar differences in time that will be seen in some of the graphs for the sample data for all the other tests in the project.

The confidence values together with the average time are presented in table 5-2:

	On node1	On node2	On node3
Low confidence boundary	469.5	1653.26	462.41
Average	469.58	1653.74	462.44
High confidence boundary	469.66	1654.22	462.47

Table 5-2: Confidence values (95% confidence interval) for simulation with no cluster.

As seen in table 5-2, the difference in average time for the simulation to finish between the two fast machines (node1 and node3) and the slow machine (node2) is very big. This is mainly because there is such a big difference in hardware between the fast and slow machines. The times for node1 are higher than for node3 even though they have the same hardware. This depends on that node1 is running router and web server software that takes system resources.

5.2.2 Homogeneous cluster

In table 5-3 the average, variance, standard deviation and 95% confidence interval of the data from the performance tests when the two fast machines node1 and node3 were connected in a cluster, is presented:

	OpenMosix	Ganglia	Ganglia together with openMosix
Average (s)	237.3	231.99	232.01
Variance (s^2)	0.28	0.09	0.08
Standard deviation (s)	0.53	0.30	0.28
95% confidence interval (s)	+/- 0.19	+/- 0.11	+/- 0.10

Table: 5-3: Statistical values for the results from the performance tests with no cluster.

In figures 5-3, 5-4 and 5-5 are graphs of the test data gotten from the performance tests together with their average and the confidence boundaries.

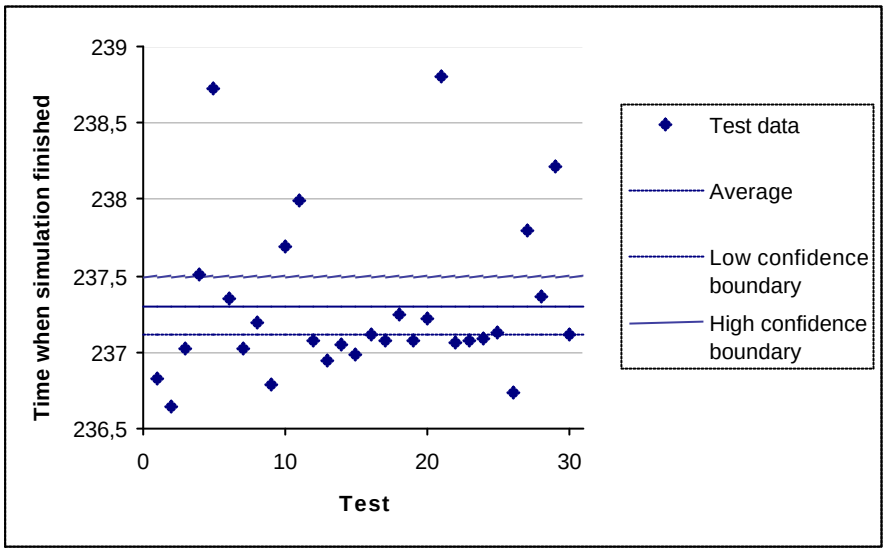


Figure 5-4: Graph of test data with average and confidence boundaries on homogeneous openMosix cluster.

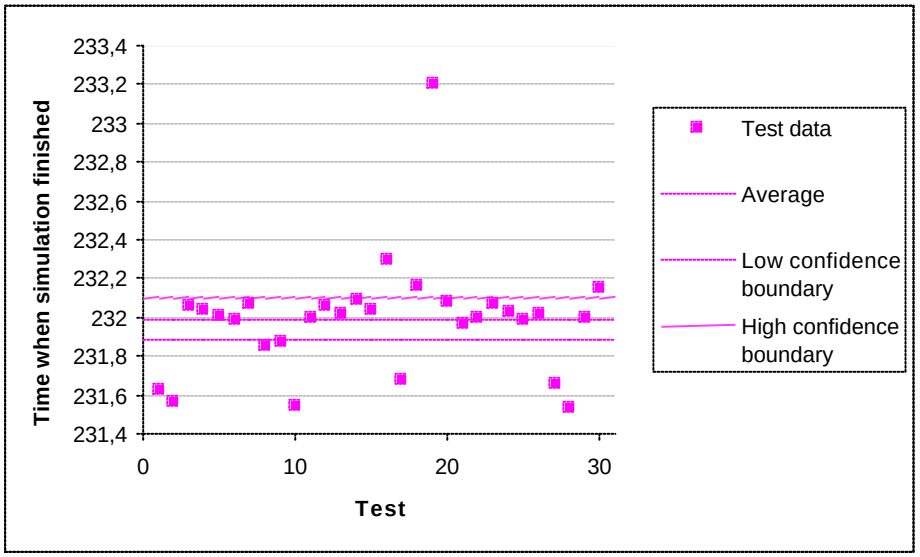


Figure 5-5: Graph of test data with average and confidence boundaries on homogeneous Ganglia cluster.

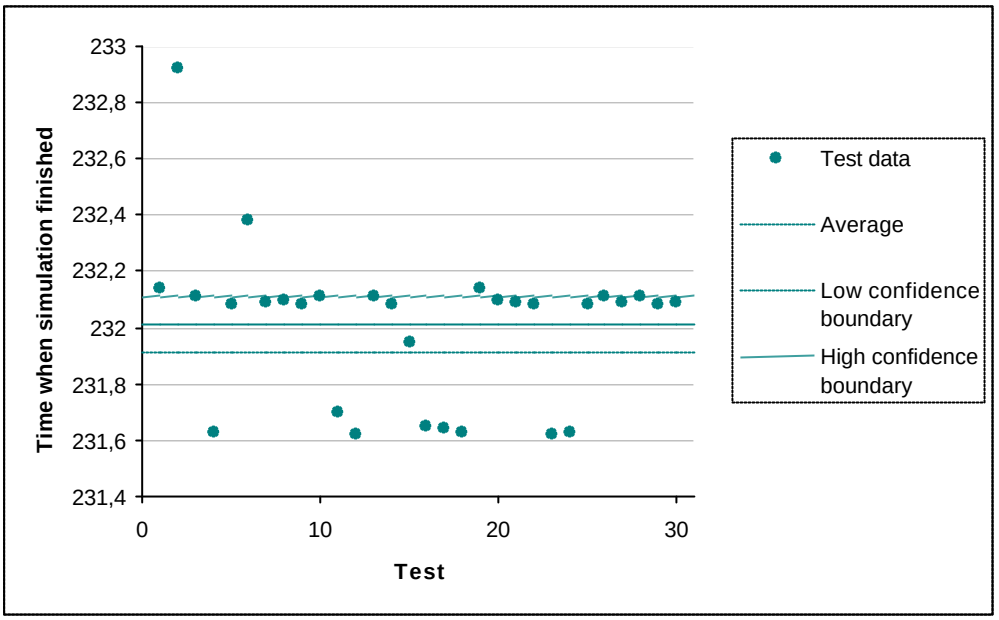


Figure 5-6: Graph of test data with average and confidence boundaries on homogeneous Ganglia & openMosix cluster.

Ganglia and Ganglia together with openMosix are better than openMosix though it's not a very big difference (only around 5 seconds).

There is no statistical difference between Ganglia and Ganglia together with openMosix, since the average values for both systems are within each other's confidence intervals.

The average time Ganglia (that was fastest with this configuration) took to finish the entire simulation were 0.5 times the average time it took for the fastest machine (node3) to finish it by itself.

5.2.3 Heterogeneous cluster with 2 computers

In table 5-4, the statistics of the tests when the fast machine node3 and the slow machine node2 were connected in a cluster. *Started on node2/node3* in the table means that the simulation-processes were started on that node and then openMosix migrated them between the machines if the cluster system saw the need for it.

	OpenMosix started on node2	**OpenMosix started on node3**	**Ganglia**	**Ganglia together with openMosix**
Average (s)	469.99	462.81	827.15	827.6
Variance (s^2)	0.97	0.03	0.24	0.2
Standard deviation (s)	0.99	0.16	0.48	0.45
95% confidence interval (s)	+/- 0.35	+/- 0.06	+/- 0.17	+/- 0.16

Table 5-4: Statistical values for the results from the performance tests with no cluster.

In figures 5-6, 5-7, 5-8 and 5-9 are plots of the test data together with their average and confidence boundaries.

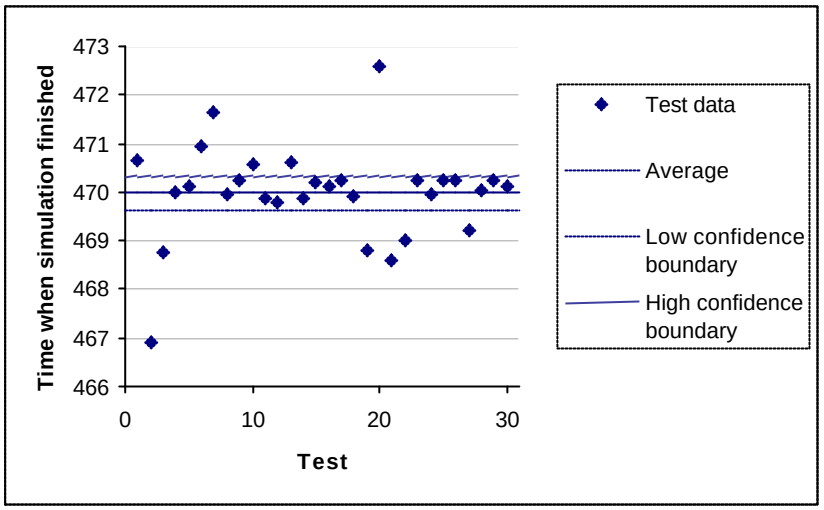


Figure 5-7: Graph of test data with average and confidence boundaries on heterogeneous openMosix cluster with 2 computers with simulations started from node2.

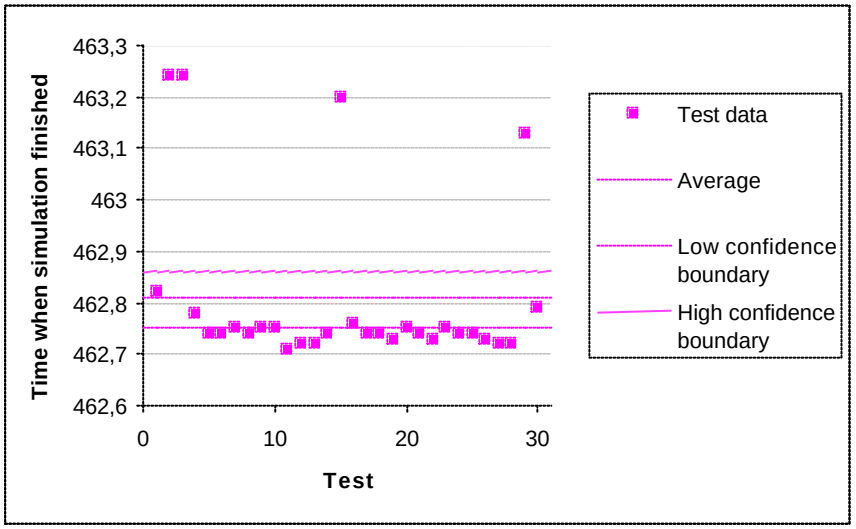


Figure 5-8: Graph of test data with average and confidence boundaries on heterogeneous openMosix cluster with 2 computers with simulations started from node3.

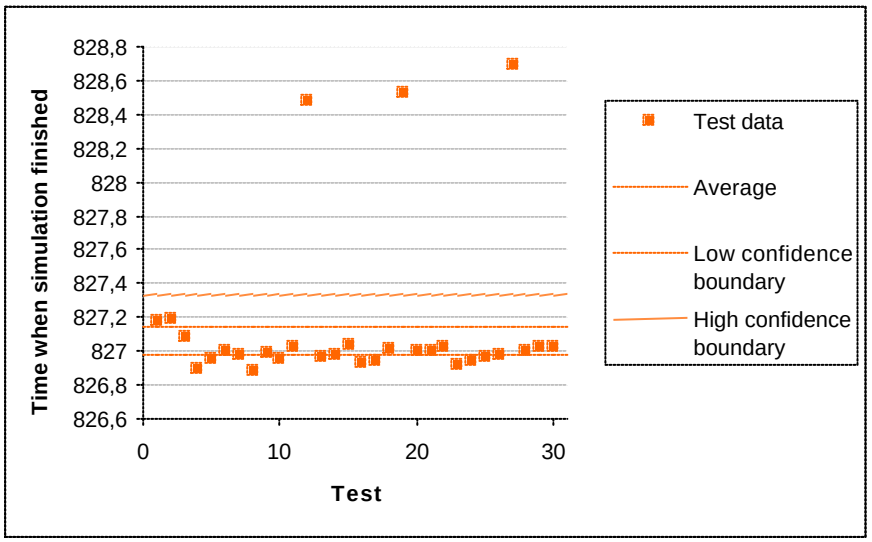


Figure 5-9: Graph of test data with average and confidence boundaries on heterogeneous Ganglia cluster with 2 computers.

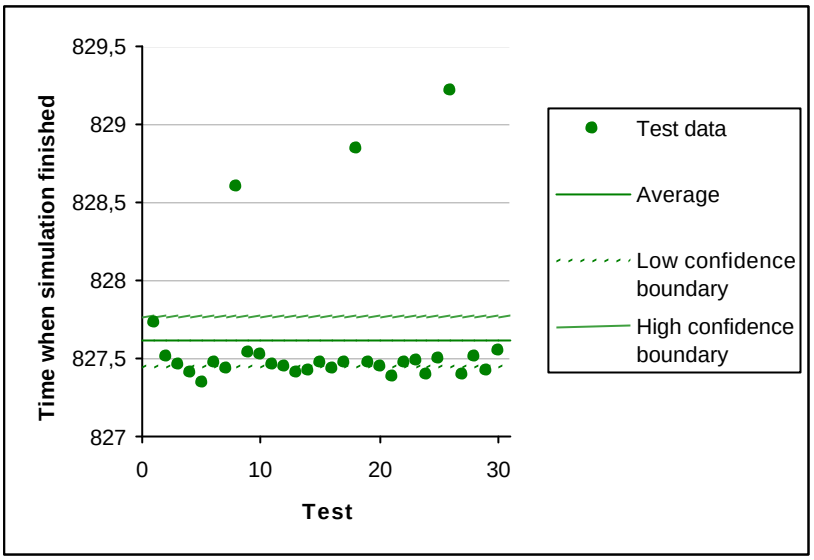


Figure 5-10: Graph of test data with average and confidence boundaries on heterogeneous Ganglia & openMosix cluster with 2 computers.

In figure 5-11 the times to finish the simulation tests are presented in a graph and the confidence values together with the average time are presented in table 5-5:

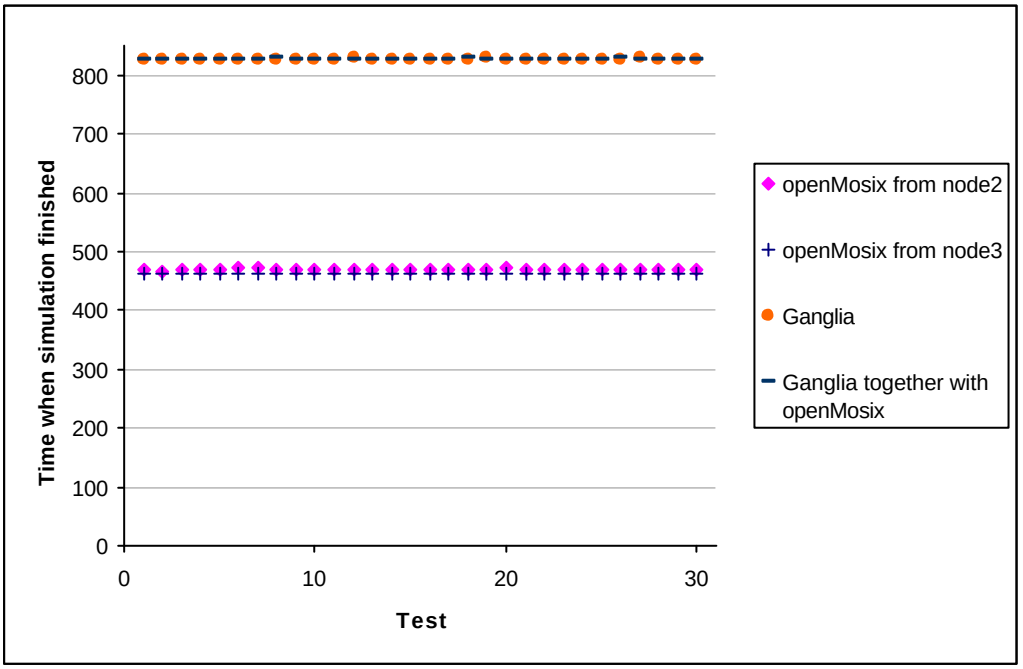


Figure 5-11: Graph of the test data from all the tests on a heterogeneous cluster with 2 computers.

	OpenMosix started on node2	OpenMosix started on node3	Ganglia	Ganglia together with openMosix
Low confidence boundary	469.63	462.75	826.98	827.45
Average	469.99	462.81	827.15	827.61
High confidence boundary	470.34	462.86	827.33	827.77

Table 5-5: Confidence values (95% confidence interval) for a heterogeneous cluster with 2 computers.

As seen in figure 5-11 openMosix is better than Ganglia and Ganglia together with openMosix. OpenMosix from node3 are better than openMosix from node2 but just with a few seconds as shown in figure 5-12 below.

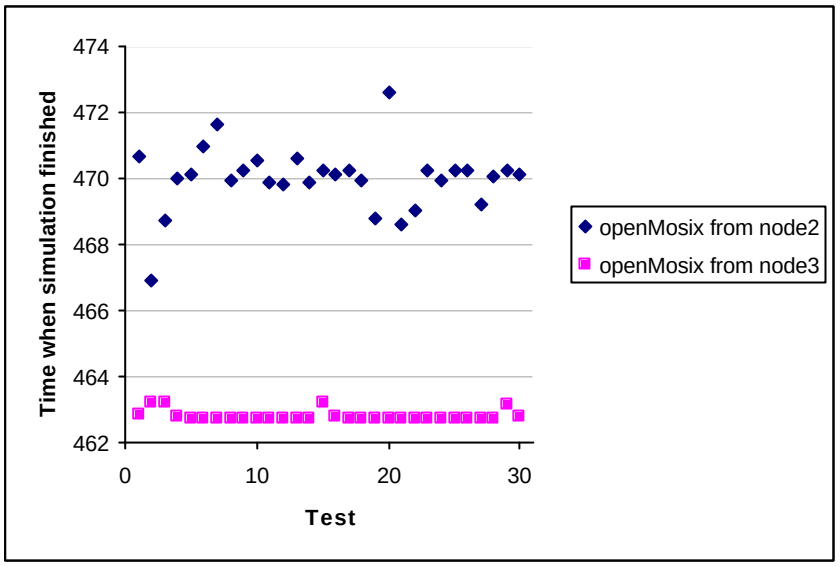


Figure 5-12: Graph test data for openMosix from node2 and node3 on a heterogeneous cluster with 2 computers.

Ganglia are a little better than Ganglia together with openMosix but not much better (less than a second) as shown in figure 5-13.

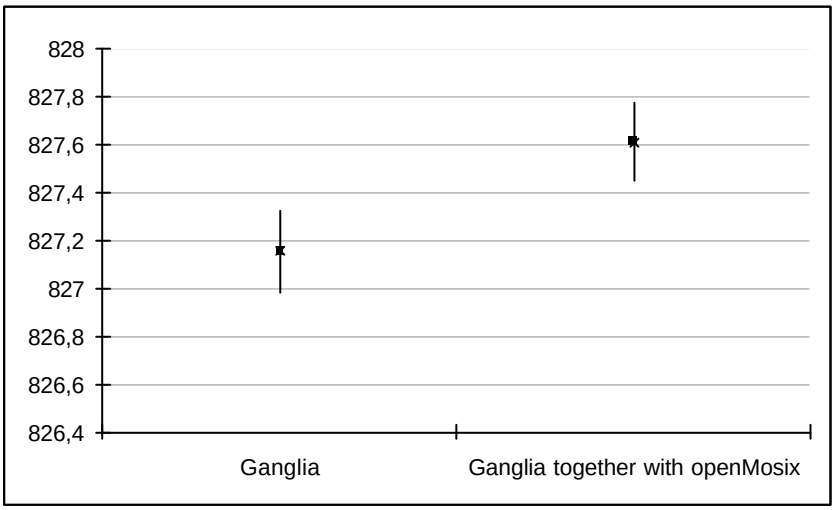


Figure 5-13: Graph of the average and confidence intervals for heterogeneous Ganglia cluster and heterogeneous Ganglia together with openMosix cluster (both with 2 computers).

The time it took for openMosix from node3 (which was fastest in these tests) to finish was more or less the same (though a little slower) than running the entire simulation on node3 by itself.

5.2.4 Heterogeneous cluster with 3 computers

In table 5-6 the average, variance, standard deviation and 95% confidence interval of the data gotten from tests when all the 3 PC machines (the two fast machines node 1 and 3 plus the

slow machine node2) was connected in a cluster, is presented:

	OpenMosix started on node2	OpenMosix started on node1	Ganglia	Ganglia together with openMosix
Average (s)	252.00	246.17	555.3	555.67
Variance (s^2)	26.53	38.15	0.23	0.11
Standard deviation (s)	5.15	6.18	0.48	0.33
95% confidence interval (s)	+/- 1.84	+/- 2.21	+/- 0.17	+/- 0.12

Table 5-6: Statistical values for the results from the performance tests with no cluster.

In figures 5-13, 5-14, 5-15 and 5-16 are graphs of the test data gotten from the performance tests together with the average and their confidence boundaries.

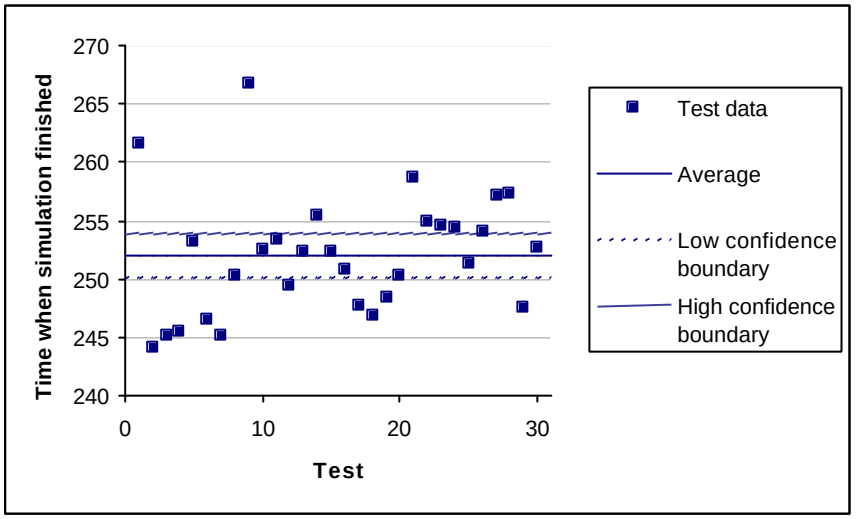


Figure 5-14: Graph of test data with average and confidence boundaries on heterogeneous openMosix cluster with 3 computers with simulations started from node2.

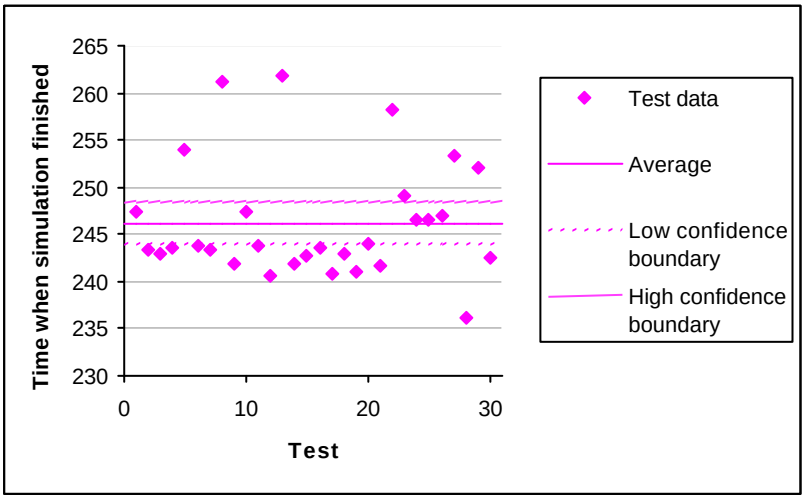


Figure 5-15: Graph of test data with average and confidence boundaries on heterogeneous openMosix cluster with 3 computers with simulations started from node1.

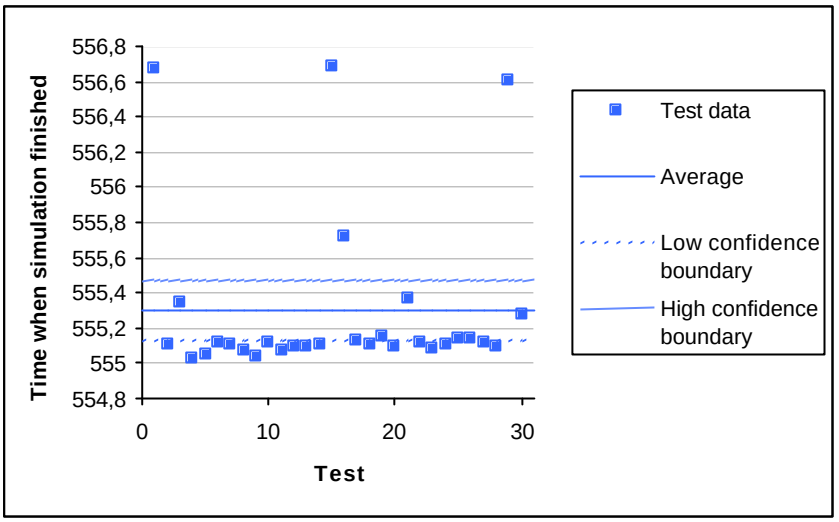


Figure 5-16: Graph of test data with average and confidence boundaries on heterogeneous Ganglia cluster with 3 computers

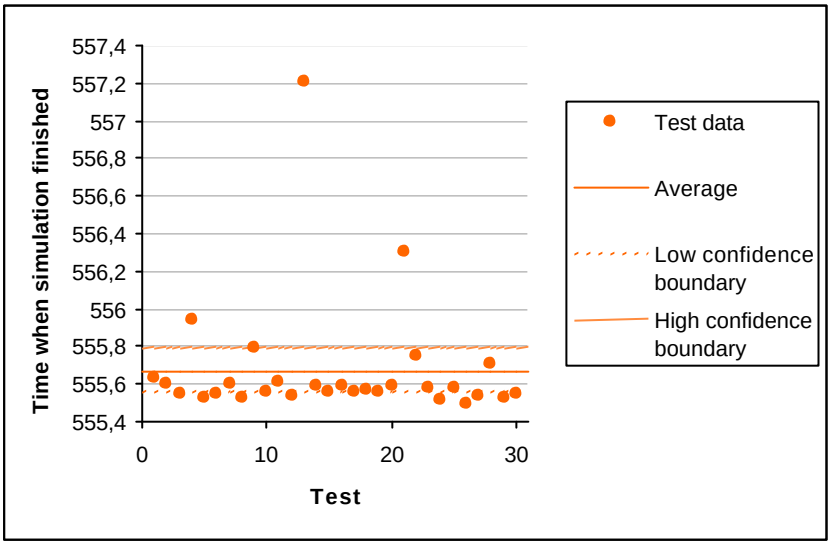


Figure 5-17: Graph of test data with average and confidence boundaries on heterogeneous Ganglia together with openMosix cluster with 3 computers.

The confidence values together with the average times are presented in table 5-7:

	OpenMosix started on node2	OpenMosix started on node1	Ganglia	Ganglia together with openMosix
Low confidence boundary	250.15	243.96	555.13	555.56
Average	251.0	246.17	555.3	555.67
High confidence boundary	253.84	248.38	555.47	555.79

Table 5-7: Confidence values (95% confidence interval) for a heterogeneous cluster with 3 computers.

As seen in table 5-7 and in figure 5-18 openMosix is far better than both Ganglia and Ganglia together with openMosix.

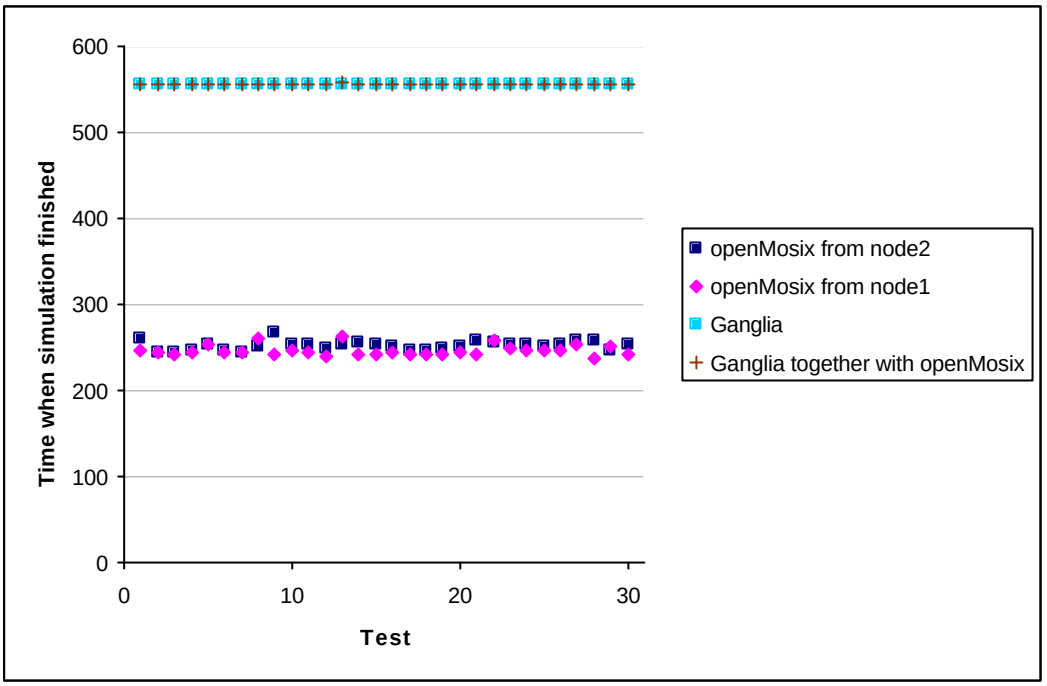


Figure 5-18: Graph of the test data from all the tests on a heterogeneous cluster with 3 computers.

OpenMosix from node1 has better times than openMosix from node2 (though not by much) but a higher variance as seen in figure 5-19.

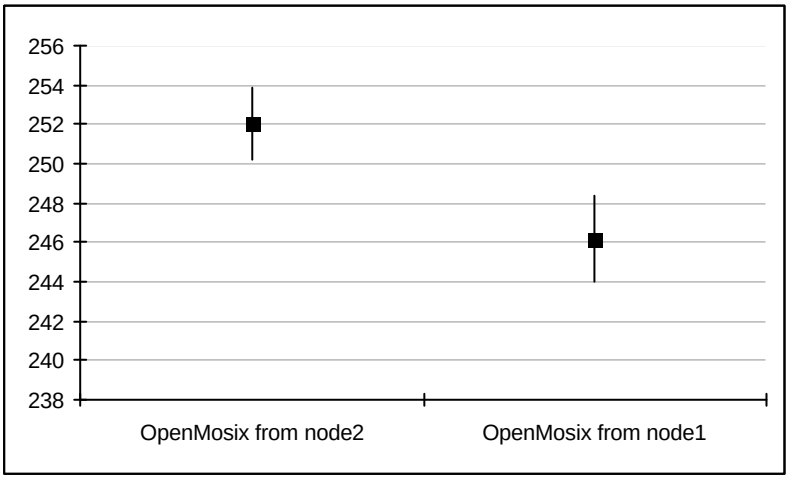


Figure 5-19: Graph of the average and confidence intervals for heterogeneous openMosix cluster (from node1 and node2) with 3 computers node1.

OpenMosix has better times than Ganglia but a very much higher variance. Ganglia is a little better than Ganglia together with openMosix as seen in figure 5-20 below, but not by much.

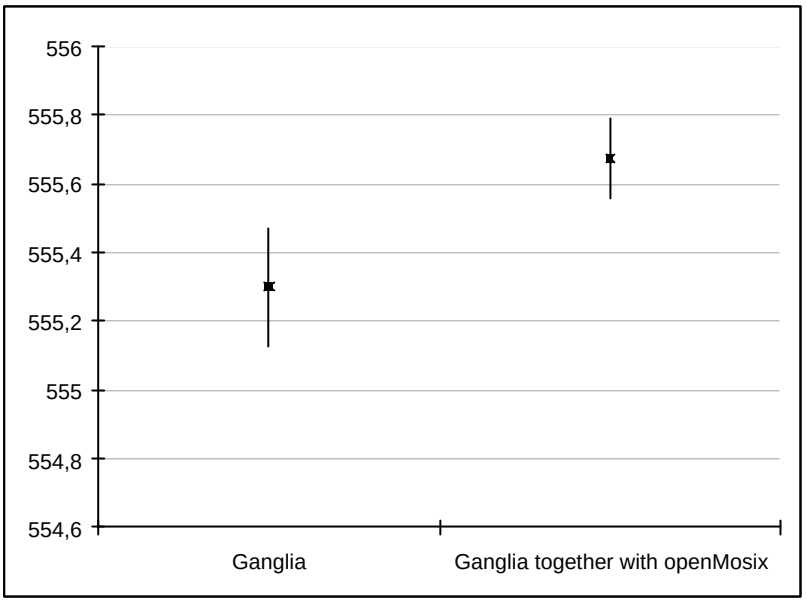


Figure 5-20: Graph of the average and confidence intervals for heterogeneous Ganglia cluster and heterogeneous Ganglia together with openMosix cluster (both with 3 computers).

The average time openMosix from node1 (that was fastest with this configuration) took to finish the entire simulation were 0.53 of the average time it took for the fastest machine (node3) to finish it alone.

5.3 Summary of all the tests

In table 5-8 the average, variance and confidence intervals for the tests when no cluster was used is presented:

	On node1	On node2	On node3
Average	469.58	1653.74	462.44
Variance	0.06	1.82	0.005
95% confidence interval	+/- 0.08	+/- 0.48	+/- 0.03

Table 5-8: Average, variance and confidence interval when no cluster is used.

The best time is when the simulation was run on node3 since this was one of the fast machines with the smallest amount of different daemons running on it. If the time for a simulation on a cluster is faster than the time it took on node3 alone then the cluster system should be considered.

In table 5-9 the average, variance and confidence intervals for all the tests with openMosix is presented.

	Homogeneous cluster	Heterogeneous cluster with 2 computers (node3)	Heterogeneous cluster with 3 computers (node1)
Average	237.3	462.81	246.17
Variance	0.28	0.97	38.15
95% confidence interval	+/- 0.19	+/- 0.06	+/- 2.21

Table 5-9: Average, variance and confidence interval for openMosix clusters.

The best times are from the homogeneous cluster but the heterogeneous cluster with 3 computers isn't much worse. When adding more computers to a heterogeneous cluster the times for the tests to finish on the cluster will get better. That the time got better might also depend on that the extra machine in the heterogeneous cluster was a fast one and therefore the jobs got migrated to it too (instead of the jobs only running on one machine that seemed to be the case with the heterogeneous cluster with 2 computers). If the third machine had been of the same hardware as the slower machine, then the time might not have been much better than for a homogeneous cluster with 2 computers since the slower machines priority in the cluster is much lower than the priority for the faster machines.

In table 5-10 the results for all the tests with Ganglia is presented.

	Homogeneous cluster	**Heterogeneous cluster with 2 computers**	**Heterogeneous cluster with 3 computers**
Average	231.99	827.15	555.3
Variance	0.09	0.24	0.23
95% confidence interval	+/- 0.11	+/- 0.17	+/- 0.17

Table 5-10: Average, variance and confidence interval for Ganglia clusters.

As seen above the best times for Ganglia was on the homogeneous cluster even though it had lesser computing power (2 fast machines) than the heterogeneous cluster with 3 computers (2 fast machines and one slow). This depends on that Ganglia runs one job on each machine even though it would be faster to run the jobs only on the faster machines. The only way to do so that this doesn't happen is that the user tells gexec to run the programs on only the fast machines with the help of the GEXEC_SVRS variable.

	Homogeneous cluster	**Heterogeneous cluster with 2 computers**	**Heterogeneous cluster with 3 computers**
Average	232.01	827.6	555.67
Variance	0.08	0.2	0.11
95% confidence interval	+/- 0.10	+/- 0.16	+/- 0.12

Table 5-11: Average, variance and confidence interval for Ganglia together with openMosix clusters.

As seen in table 5-11 above, Ganglia together with openMosix is more or less the same as only Ganglia. This is because openMosix doesn't seem to automatically migrate the simulation processes when it's started with the help of gexec. To make the processes migrate so that they finish faster there is two ways: Either by setting the processes */proc/[PID]/lock* file to 0 (For example *echo 0 > /proc/1234/lock* if the process-id is 1234.) so that the automatic migration of the process is enabled or by manually migrate the process with the migrate command.

In both the heterogeneous systems, openMosix was a lot better than Ganglia and Ganglia together with openMosix. One thing with the openMosix systems on a heterogeneous computer configuration is that the small jobs used in the performance tests didn't migrate to the slower machine, node2, but went all over to the faster machines. This might be because node2 was a lot worse than the two other machines (1/3 the processor power, 1/4 the

memory) and therefore got a much lower priority in openMosix load balancing procedures. The only time Ganglia and Ganglia together with openMosix were best was on a homogeneous system, but the time difference weren't that big and on the heterogeneous systems they where a lot worse than a heterogeneous cluster running openMosix.

The systems whose test times were better than the times on one single fast machine (node3) was openMosix on a homogeneous cluster, openMosix on a heterogeneous cluster with 3 computers, Ganglia on a homogeneous cluster and Ganglia together with openMosix on a homogeneous cluster. These systems are the only ones that should be considered as a solution since it’s no idea to create a cluster if one machine can finish the simulations as fast as the cluster.

Since the cluster ITS is considering building is going to use machines they don't currently use, it won't probably be a homogeneous cluster. Therefore an openMosix cluster should be best according to these performance tests. Ganglia can still be used on the cluster though since it’s not needed to run programs with gexec.

Chapter 6 Discussion and conclusions

6.1 Comparison of Ganglia and openMosix

When it comes to hardware and operating systems Ganglia's monitoring core works on a lot more systems than Ganglia's Execution environment and openMosix who only work on Intel x86 and Athlon machines running Linux. But since a Ganglia cluster without the execution environment won't be useful for the tasks ITS wants to be done, it doesn't matter what of the systems is chosen. The Ganglia execution environment might be ported in the future but there is no information about when or if this will be done.

When it comes to installation both systems can be installed via RPMs on a machine running RedHat Linux. Since that Linux distribution is the one mainly used at ITS, it probable that the clustering system would be installed in this way on the cluster. Installing the RPMs is just as easy for openMosix as Ganglia.

When it comes to adding nodes Ganglia's auto-discovery is of great help. But since openMosix also have a sort of auto-discovery the difficulty in adding nodes to the systems isn't that different between them either.

When it comes to administration of the systems, all of Ganglia's settings exist in two files (one for gmond and one for gmetad). OpenMosix has a directory structure with administrative information that can either be edited manually or via the tools that exists to openMosix. Monitoring of the systems are taken care of by a web interface on Ganglia, while openMosix uses openMosixview together with openMosixcollector or mosmon if X windows can't be used. Ganglia's monitoring system can monitor more things about the computers than openMosix (and new things to monitor can also be added). There is a possibility to use Ganglia together with openMosix. If the tools in Ganglia's monitoring core are needed they can be used without too much interference in openMosix operations.

When it comes to running programs on the two systems openMosix is better than Ganglia, for parallel execution of programs. Ganglia's execution environment is good for running the same program on several machines at the same time while openMosix can send different sub processes to other machines. There are certain programs that can't migrate to other machines on an openMosix system mainly because of shared memory. Of course if the user really just want to run one instance of the same program on all the machines then Ganglia is better but this is mostly not the case if a system for calculations and simulations are wanted. Another thing with openMosix is that the binary for the program that is going to be executed only needs to be on the machine that the program is started from while in Ganglia the binary needs to be on all the machines that gexec are going to start up the programs on. Finally another positive thing with openMosix is that there are several ways to make (or recode) the programs that the user wants to run parallel on the cluster.

When it came to the parallel performance tests, openMosix was considered be the best as described in chapter 5.3. Ganglia is better than openMosix on a homogeneous cluster but on a heterogeneous cluster openMosix work better.

6.2 Summary of the comparison conclusions

To summarize:

Hardware and operating systems: Both systems works on Intel x86 and Athlon machines running Linux. Ganglia's monitoring core work on even more hardware and operating systems (including SPARC running either Linux or Solaris) but the execution environment doesn't work on those. Therefore there is more or less no difference between openMosix and Ganglia.

Installation and adding nodes: There isn't much of a difference in the difficulty to install the two different systems. If auto-discovery isn't used on openMosix then it's a little more work adding nodes but not much more. So openMosix and Ganglia are more or less the same here too.

Administration and monitoring: Ganglia only have two files to edit, and is probably easier to administrate than openMosix because of this. There is quite a difference on what can be done with the systems though and therefore they are quite hard to compare. Ganglia can be used to monitor more computer statistics than openMosixview. It also has a practical web interface for viewing these. There is always the solution to run Ganglia together with openMosix if Ganglia's monitoring tools are wanted.

Running programs: Here openMosix is the better clustering system when it comes to computations and simulations. There do exist several ways to program the computation or simulation programs so they migrate on the cluster etc.

Parallel performance: Here openMosix is best on heterogeneous clusters (which is probably how the cluster that ITS wants will be) while Ganglia were best on a homogeneous cluster. It was very hard to find a program that could run in parallel on openMosix and Ganglia, which also speak in openMosix favor.

The conclusion made in this project is therefore that openMosix is better than Ganglia for the kind of system that ITS wants. Ganglia could however, be used as an administration tool of the openMosix cluster.

6.3 Other clustering systems

There do exist other clustering systems and toolkits that can be used and that might be better than the two written about in this report.

One of the most known clustering systems is Beowulf[42] that was created by the NASA contractor CESDIS (Center of Excellence in Space Data and Information Sciences) in 1994. The Beowulf Project is now hosted by Scyld Computing Corporation, which was founded by members of the original Beowulf team.

Parallel Virtual Machine (mentioned in chapter 3.5) can be used to connect several machines in a cluster too. The programs that should run parallel on a PVM cluster have to the programmed with the libraries that PVM uses for parallel computing. PVM can be used in conjunction with other clustering systems like for example openMosix.

There also exists different clustering toolkits where a group has collected and in some instances re-coded different clustering tools so they can work together. Some of these toolkits builds on openMosix or Beowulf clusters while others can be used with any kind of cluster.

There is more information on different kinds of clustering systems on [9] and [10].

References

[1] Ganglia Development Team, *Ganglia Toolkit*, http://ganglia.sourceforge.net/
[2] B. Knox, *openMosix, an Open Source Linux Cluster Project*, http://www.openmosix.org/
[3] A. Barak, *MOSIX*, http://www.mosix.org/
[4] *Licenses - Gnu Project - Free Software Foundation (FSF)*, http://www.fsf.org/licenses/licenses.html#GPL
[5] *RedHat Linux*, http://www.redhat.com/
[6] *Debian GNU/Linux*, http://www.debian.org/
[7] H-G. Hegering, S. Abeck, B. Neumair, *Intergrated Management of Networked Systems: Concepts, Architectures, and Their Operational Application*, 1999, ISBN 1-55860-571-1
[8] *The OMG's CORBA Webpage*, http://www.corba.org/
[9] K Railsback, *Linux Clustering in Depth*, 2000, http://www.linux-mag.com/2000-10/clustering_01.html
[10] J. Greenseid, *Linux Clustering Information Center*, http://lcic.org/
[11] R. Shah, *Linux clustering cornucopia*, 2000, http://www.linuxworld.com/linuxworld/lw-2000-03/lw-03-clustering.html
[12] K. Buytaert, *The openMosix HOWTO*, http://howto.ipng.be/openMosix-HOWTO/
[13] M. Bar, *openMosix Internals: How openMosix Works*, http://sourceforge.net/docman/display_doc.php?docid=10390&group_id=46729
[14] *RPM Package Manager*, http://www.rpm.org/
[15] *openMosixview cluster-management GUI*, http://www.openmosixview.com/
[16] *Trolltech - Qt - Overview*, http://www.trolltech.com/products/qt/
[17] *The Linux Kernel Archives*, http://www.kernel.org/
[18] *PVM: Parallel Virtual Machine*, http://www.csm.ornl.gov/pvm/
[19] MPI - The Message Passing Interface Standard, http://www-unix.mcs.anl.gov/mpi/
[20] The Apache Software Foundation, http://www.apache.org/
[21] Maple 8, http://www.maplesoft.com/products/Maple8/
[22] *Mathematica*, http://www.wolfram.com/products/mathematica
[23] *The MathWorks*, http://www.mathworks.com/
[24] *MySQL*, http://www.mysql.com/
[25] *ns version 1 - LBNL Network Simulator*, http://www-nrg.ee.lbl.gov/ns/
[26] *The Network Simulator – ns-2*, http://www.isi.edu/nsnam/ns/
[27] *Octave Home Page*, http://www.octave.org/
[28] *The R Project for Statistical Computing*, http://www.r-project.org/
[29] *openMosix Documentation Wiki*, http://howto.ipng.be/openMosixWiki/
[30] *UC Berkeley Millennium Research Group*, http://www.millennium.berkeley.edu/
[31] Ganglia Development Team, *Ganglia Documentation Pre-2.5.0*, http://ganglia.sourceforge.net/pre2.5.0-docs/
[32] Ganglia Development Team, *Ganglia Documentation*, http://ganglia.sourceforge.net/ganglia_docs/
[33] *RSA Laboratories - Research Center of RSA Security*, http://www.rsasecurity.com/rsalabs/
[34] *PHP: Hypertext Preprocessor*, http://www.php.net/
[35] *Extensible Markup Language (XML)*, http://www.w3.org/xml/
[36] *Python Language Website*, http://www.python.org/
[37] *OpenSSL: The Open Source toolkit for SSL/TLS*, http://www.openssl.org/
[38] *Alien package converter*, http://kitenet.net/programs/alien/
[39] *RRD Tool*, http://www.rrdtool.com/

[40] *netfilter/iptables - Home,* http://www.netfilter.org/
[41] R. Jain, *The Art of Computer Systems Performance Analysis*, 1991, ISBN 0-471-50336-3
[42] *Beowulf.org*, http://www.beowulf.org/

Appendix A Data from the performance tests

No cluster

In this test there was 1 simulation with 600 replications. These tests were made on the Pentium 300 MHz machine, node2, and the two Celeron 1 GHz machines, node1 and node3, without any migration between the machines. These data are to see how much the clustering of computers help when it comes to the performance. If the time of a test on a cluster is lower than the times that the job takes on one machine then the cluster isn't good to use.

	On node1	On node2	On node3
Test	Time	Time	Time
1	469,99	1656.54	462.43
2	469,76	1654.53	462.47
3	469,25	1654.53	462.44
4	469,84	1655.16	462.42
5	469,29	1656.01	462.42
6	469,34	1656.12	462.43
7	469,7	1654.09	462.43
8	469,7	1653.18	462.42
9	469,29	1652.49	462.42
10	469,18	1654.37	462.42
11	469,47	1653.15	462.43
12	469,86	1652.56	462.42
13	469,69	1652.75	462.43
14	469,8	1654.02	462.43
15	469,15	1652.79	462.42
16	469,78	1652.42	462.43
17	469,7	1652.81	462.81
18	469,74	1654.31	462.41
19	469,61	1653.02	462.42
20	469,29	1656.91	462.43
21	469,38	1652.51	462.43
22	469,65	1652.81	462.43
23	469,32	1654.33	462.43
24	469,35	1652.41	462.44
25	469,67	1652.68	462.42
26	469,64	1652.35	462.44
27	469,67	1654.35	462.43
28	469,82	1652.44	462.43
29	469,33	1652.46	462.44
30	469,77	1654.06	462.43

Homogeneous cluster

In this test there was 2 simulations with 300 replications per simulation. These tests were made on the two Celeron 1 GHz machines, node1 and node3.

Test	OpenMosix Time	Ganglia Time	Ganglia together with openMosix Time
1	236.83	231.63	232.14
2	236.64	231.57	232.92
3	237.03	232.06	232.11
4	237.51	232.04	231.63
5	238.73	232.01	232.08
6	237.35	231.99	232.38
7	237.03	232.07	232.09
8	237.2	231.85	232.1
9	236.79	231.87	232.08
10	237.69	231.54	232.11
11	237.99	232	231.7
12	237.08	232.06	231.62
13	236.95	232.02	232.11
14	237.05	232.09	232.08
15	236.99	232.04	231.95
16	237.12	232.3	231.65
17	237.08	231.68	231.64
18	237.24	232.16	231.63
19	237.07	233.2	232.14
20	237.22	232.08	232.1
21	238.8	231.97	232.09
22	237.06	232	232.08
23	237.08	232.07	231.62
24	237.09	232.03	231.63
25	237.13	231.99	232.08
26	236.74	232.02	232.11
27	237.79	231.66	232.09
28	237.36	231.53	232.11
29	238.21	232	232.08
30	237.12	232.15	232.09

Heterogeneous cluster with 2 computers

In this test there was 2 simulations with 300 replications per simulation. These tests were made on a cluster consisting of the Pentium 300 MHz machine, node2, and the Celeron 1 GHz machine, node3.

Test	OpenMosix from node2 Time	OpenMosix from node3 Time	Ganglia Time	Ganglia together with openMosix Time
1	470.64	462.82	827.18	827.73
2	466.9	463.24	827.19	827.51
3	468.74	463.24	827.08	827.46
4	469.99	462.78	826.9	827.41
5	470.1	462.74	826.95	827.34
6	470.94	462.74	827	827.48
7	471.66	462.75	826.98	827.43
8	469.94	462.74	826.88	828.6
9	470.23	462.75	826.99	827.54
10	470.56	462.75	826.95	827.53
11	469.88	462.71	827.02	827.46
12	469.79	462.72	828.48	827.45
13	470.61	462.72	826.97	827.41
14	469.86	462.74	826.98	827.42
15	470.22	463.2	827.04	827.47
16	470.12	462.76	826.93	827.43
17	470.26	462.74	826.94	827.47
18	469.93	462.74	827.01	828.84
19	468.81	462.73	828.53	827.47
20	472.59	462.75	827	827.45
21	468.58	462.74	827	827.38
22	469.01	462.73	827.02	827.48
23	470.23	462.75	826.92	827.49
24	469.95	462.74	826.94	827.4
25	470.23	462.74	826.97	827.5
26	470.23	462.73	826.98	829.22
27	469.2	462.72	828.69	827.4
28	470.04	462.72	827	827.51
29	470.26	463.13	827.02	827.42
30	470.13	462.79	827.03	827.55

Heterogeneous cluster with 3 computers

In this test there was 3 simulations with 200 replications per simulation. These tests were made on a cluster consisting of the Pentium 300 MHz machine, node2, and the two Celeron 1 GHz machines, node1 and node3.

Test	OpenMosix from node2 Time	OpenMosix from node1 Time	Ganglia Time	Ganglia together with openMosix Time
1	261.53	247.3	556.68	555.63
2	244.03	243.37	555.11	555.6
3	245.11	242.98	555.35	555.55
4	245.57	243.57	555.03	555.94
5	253.21	254.04	555.05	555.53
6	246.55	243.72	555.12	555.55
7	245.09	243.3	555.11	555.6
8	250.29	261.11	555.07	555.53
9	266.67	241.97	555.04	555.79
10	252.48	247.35	555.12	555.56
11	253.42	243.75	555.07	555.61
12	249.39	240.51	555.09	555.54
13	252.35	261.76	555.1	557.21
14	255.48	241.91	555.11	555.59
15	252.33	242.79	556.69	555.56
16	250.75	243.65	555.72	555.59
17	247.64	240.84	555.13	555.56
18	246.81	242.91	555.11	555.57
19	248.4	241.04	555.15	555.56
20	250.29	244.03	555.09	555.59
21	258.67	241.65	555.37	556.3
22	254.94	258.18	555.12	555.75
23	254.54	249.14	555.08	555.58
24	254.32	246.48	555.11	555.52
25	251.29	246.44	555.14	555.58
26	254.01	247.06	555.14	555.5
27	257.19	253.35	555.12	555.54
28	257.32	236.14	555.1	555.71
29	247.61	252.11	556.61	555.53
30	252.62	242.55	555.28	555.55

Appendix B

openMosix How-to for ITS

Emma Roos
emma@fukt.bth.se

Table of Contents

1 Brief introduction

This how-to describes how to install and administrate openMosix plus how to use some of the tools that comes with it. For an introduction on this clustering system see the report.

2 Requirements

The basic hardware requirements are at least 2 network-connected computers, the faster the network the better performance you will get. The computers also have to have either Intel x86 or Athlon processors.

The system needs a basic Linux installation of any distribution though this how-to is written with RedHat in mind. The network cards needs to be properly configured and you need a quite a lot of swap-space. If you are compiling openMosix you have to have the source for a 2.4.* kernel. The openMosix tool omdiscd also needs IP multicast support enabled in the kernel.

OpenMosixview needs X, QT 2.3.0 or above, ssh (or rlogin and rsh) on all the nodes (not only the one with openMosixview on it) and the User-space tools installed.

3 Downloading

The openMosix kernel and user-space tools can be found at:
http://sourceforge.net/project/showfiles.php?group_id=46729.
Download the openMosix kernel you want (you have to have the same version on all the machines). When this how-to was written the most stable openMosix version was 2.4.19-6. If you download the rpm version don't forget to choose the one that fits best for the nodes hardware (athlon for AMD Athlon/Duron/K7 processors, i686 for Pentium-Pro/Celeron/Pentium-II/newer Pentium processors and i386 for other Intel processors or if you're not sure (and don't need SMP support). If you have more than one processor on the machine you should choose a RPM compiled with SMP support. You can check what you should use by typing *uname -a* at the prompt on the node. This will tell you what the current kernel is compiled for and if you have used the standard installation then this should be the correct kernel-type to use.
If you are going to compile openMosix download the openMosix patch with the same version number as the kernel source version you are going to compile.
Download also the User-space tools. There are two RPMs (i385 and source) and a tar.gz file with the source. Choose the version you like (the latest version that existed when this how-to was written was 0.2.4).

You will find openMosixview on *http://www.openmosixview.com/download.html.* The RPMs that exists are for RedHat 7.2 and 7.3 plus for SuSE 7.2. There is also a tar.gz with the source code. The source for QT (that is needed if openMosixview is compiled from source) can be found on *ftp://ftp.troll.no/pub/qt/source*.

4 Installation with RPMs

4.1 Introduction

The files mentioned in the installation instructions are the versions that were the latest when this how-to was written. This section only describes the installation on a RedHat system with RPMs. How to compile openMosix, the user-space tools and openMosixview is described in chapter 5 of this how-to. For information on how to install openMosix on other Linux distributions see Kris Buytaert's openMosix HOWTO.

4.2 Installing the openMosix kernel and the user-space tools

As root go to the directory where you downloaded the openMosix and user-space tools RPMs. Then use the rpm command to install the kernel and the user-space tools at the same time. For example if you are installing openMosix 2.4.19-6 on an ordinary Intel machine with one processor then type:

rpm -vih openmosix-kernel-2.4.19-openmosix6.i386.rpm openmosix-tools-0.2.4-1.i386.rpm

During this installation a script in the kernel rpm will try to add your new kernel to the boot-loader. If you have RedHat 7.2 or later with Grub as boot-loader this will work automatically otherwise you will have to add the new kernel to */etc/lilo.conf* manually.
A soft-link named */boot/vmlinuz-openmosix* always points to the new openMosix-kernel, so you don't have to edit */etc/lilo.conf* each time you upgrade the openMosix-kernel. A soft-link */boot/initrd-openmosix.img* is also created, pointing to the new openMosix-initrd. Put it in your *lilo.conf* if you need some of the modules in order to boot. Just copy the lines in */etc/lilo.conf* that lists the current kernel and change image, label and initrd.

Take and check the /etc/hosts file so that it the node's IP is listed properly in the file. If the machines IP is 192.168.0.1 and its hostname om1.itslabb.bth.se) then the host should be listed in /etc/hosts as:

192.168.0.1 om1.itslabb.bth.se
127.0.0.1 localhost

When this is done there is one of two things that should be done depending if you've decided to use the auto-discovery tool or not.

If you are not using auto-discovery then you have to edit your */etc/mosix.map* on each node. In the file each line should contain 3 fields that maps IP addresses to openMosix node-numbers:

1) The first openMosix node-number in the IP range used.
2) The IP address of the above node
3) The number of nodes in the IP range used

For example if you got 5 computers in the IP range of 192.168.0.1-192.168.0.5 the end of

/etc/mosix.map (the beginning of the file contains comments on how the file should look like) on each machine should look like this:

1 192.168.0.1 5

If you are going to use the auto-discovery daemon (omdiscd) you don't need the */etc/mosix.map*. Disable the */etc/init.d/openmosix* script with the command:

chkconfig --del openmosix

Now run *omdiscd* on each node (openMosix must be down). If you want to run omdiscd automatically when the machine reboots you have to create a startup script in */etc/init.d* (for example called *omdiscd*) and since no such script comes with omdiscd you have to create this file manually. Below is an example of how this file could look like (does only work on RedHat systems (possibly also in SuSE Linux)):

```
#!/bin/bash
#
# chkconfig: 2345 95 5
# description: omdiscd is a tool to automatically discover openMosix nodes.
#
# omdiscd           Script to stop/start omdiscd

# Source function library.
[ -f /etc/rc.d/init.d/functions ] || exit 0
. /etc/rc.d/init.d/functions

RETVAL=0

#
# The pathname substitution in daemon command assumes prefix and
# exec_prefix are same.  This is the default, unless the user requests
# otherwise.
#
case "$1" in
  start)
    echo -n "Starting omdiscd: "
       daemon /sbin/omdiscd  # default interface is eth0 if another network inteface should
                             # be used use the -i option (ex: /sbin/omdiscd   -i eth1)
    RETVAL=$?
    echo
    [ $RETVAL -eq 0 ] && touch /var/lock/subsys/omdiscd
    ;;
  stop)
    echo -n "Shutting down omdiscd: "
    killproc omdiscd
    RETVAL=$?
    [ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/omdiscd
    echo
    ;;
```

```
    status)
        status omdiscd
        ;;
    restart)
        $0 stop
        $0 start
            ;;
    *)
        echo "Usage: $0 {start|stop}"
        exit 1
        ;;
esac
```

Then add this new script so that it loads at reboot by typing the command:

chkconfig --add omdiscd

Now it's time to reboot the machines, choose the openMosix kernel in the boot-loader and the cluster should work.

4.3 Installing openMosixview

First you should install openMosixview on the nodes that you want to use for monitoring the cluster. As root go to the directory where the downloaded openMosixview RPM is. Install it with the rpm command. For example if you are going to install openMosixview 1.2 for RedHat 7.3 type:

rpm -vih openMosixview-1.2-rh73.rpm

On the rest of the nodes only openMosixprocs is needed. Copy */usr/bin/openmosixprocs* from one of the machines that has openMosixview installed to */usr/bin* on all the nodes that doesn't have openMosixview installed on them.

After that openMosixview/openMosixprocs has been installed on all the nodes it's time to configure SSH so that passwords aren't needed when different features in openMosixview is used. First you have to create a RSA key-pair on all the nodes this you do with the command ssh-keygen.
If you use SSH1 you have to write *ssh-keygen -t rsa1* and if you use SSH version 2 you just change *rsa1* to *rsa*. The program will prompt you for what files you want the keys to be in (use the default) and then it will ask you for a pass phrase. You don't need a pass phrase but if you decide to have one you need to use the program *ssh-agent* before you run openMosixview.

The key pairs will be saved in two files in the */root/.ssh* directory: *identity* (private key) and *identity.pub* (public key) for SSH1 or *id_rsa* (private key) and *id_rsa.pub* (public key) for SSH2. Do **not** give out the private key to anyone!

Now copy the entire content in the public key files on the machines that openMosixview has been installed on into */root/.ssh/authorized_keys* on all the nodes in the cluster. In the end all

the nodes (both the ones with openMosixview and the ones with just openMosixprocs) should have the public keys of all the nodes with openMosixview in */root/.ssh/authorized_keys*. If the machine has openMosixview on it then its own key should also be in this file.

5 Installing from source

5.1 Introduction

The files mentioned in the installation instructions are the versions that were the latest when this how-to was written.

5.2 Compiling the openMosix kernel

You'll always have to use a pure vanilla kernel-sources to compile an openMosix kernel. The kernel source also needs to be the same version as the openMosix kernel parch you are going to use. For example if you are using openMosix 2.4.19-6 you have to use the 2.4.19 kernel.

Download the correct kernel source from *ftp://ftp.kernel.org/pub/linux/kernel/v2.4/* and unpack the kernel with: *tar -xzvf linux-2.4.19.tar.gz*

Apply the patch by going into the kernel source directory and then type the following command (if the kernel patch is in the same directory as where you downloaded the kernel source): *zcat ../openMosix-2.4.16-6.gz | patch -p1*

Now type: *make menuconfig* and enable the openMosix-options in the configuration programs. If the auto discovery daemon is going to be used then IP multicast also has to be enabled. Do also look through the other options shown and configure the kernel the way you want it.

When you exited the configuration utility its time to compile the kernel via:

make dep
make bzImage
make modules

And then install the kernel with:

make modules_install
make install

Edit your boot-loader and reboot.

There might show up problems when you try to reboot the kernel. There are some problems with certain settings in the vanilla kernel on RedHat systems. If the new openMosix kernel doesn't boot when reboot the machine with the old kernel and experiment by removing different options in the kernel. These problems are the reason why installing with RPMs are so much easier.

5.3 Compiling openMosix userland tools

Unpack the openMosix userland source: *tar -xzvf openMosixUserland-0.2.4.tgz*
Go into the directory that is created and edit the file configuration. The lines you only need to touch (most of the times) are (for paths use only **absolute** paths):

OPENMOSIX	Shows where the openMosix kernel is.
INSTALLDIR	Which is where the system base directory is (usually /).
INSTALLEXTRADIR	Where the applications are installed (usually /usr/local or /usr).
INSTALLMANDIR	Is where the man pages should be (usually $INSTALLEXTRADIR/man).
CC	What C compiler should be used (usually gcc).
MONNAME	The filename that openMosix monitoring tool should be called. The recommended name is mosmon (the name used for the program in the rpm).

When this is done it's time to compile the programs. You do this by typing: *make all*

After this copy the file *openmosix* in the scripts directory to */etc/init.d* and if you're not going to use the auto discovery tool make so that openMosix is started on reboot by the command:

chkconfig --add openmosix

If auto discovery is going to be used then the openmosix script shouldn't be added with chkconfig. Instead install the auto discovery by going into the *autodiscovery* directory that lies in the directory with the userland tools source, then remove the line "#define ALPHA" from the files *openmosix.c* and *showmap.c*. After that run the commands: *make clean* and *make*

To make the auto discovery daemon to run on reboot, create a script like the one described in the chapter 4.2 (call the file for example *omdiscd*) in */etc/init.d* and then add it so it gets started at reboot with the command:

chkconfig --add omdiscd

5.4 Compiling openMosixview

When you compile openMosixview on the machine you want to run it on you will not only need the QT libraries (that comes with the ordinary QT RPM) but you also need the QT source files available on the system. To install QT from source download the source tar.gz and then unpack it:

tar –xvzf qt-x11-free-3-0.5.tar.gz

After this you should move the entire directory created to */usr/local* with the new name *qt-x.y.z* (there x.y.z is the QT version):

mv qt-x11-free-3-0.5 /usr/local/qt-3.0.5

When this is done go into the new directory:

cd /usr/local/qt-3.0.5

Before you run the configure script you have to set the QTDIR variable so that the script find the source files.

export QTDIR="/usr/local/qt-3.0.5"

Thereafter run the configure script and compile (*make* will take quite a while):

./configure
make
make install

Now you will be able to compile openMosixview. Begin by go ing to the directory where you downloaded the openMosix tar.gz and unpack the downloaded openMosixview source code:

tar -xzvf openmosixview-1.2.tar.gz

Then go to the directory created and execute the automatic setup-script that comes with the source:

./setup [your_qt_2.3.x_installation_directory]

When openMosixview is installed copy *openmosixprocs* to */usr/bin/* on all the nodes and then fix SSH on all the nodes as described in the chapter 4.3.

6 Administrating openMosix

The openMosix cluster can be configured by using the */proc/hpc* interface and the user-space tools. OpenMosixview can also be used to configure certain settings and to monitor the cluster.

6.1 The /proc/hpc interface

In */proc/hpc* there is a number of sub directories with different openMosix settings and statistics. Some of the files in these directories can be set manually others just show statistics and is changed by the system.

The directories in */proc/hpc* are:

admin - that presents the current configuration of the system.
decay - shows the decay statistics (load balancing information settings).
info - shows information about the computer (in binary format).
nodes - has information about the different nodes in the cluster.
remote - has information about the remote processes that has migrated to the computer

There are also files with openMosix information for a specific process in the */proc/[PID*] directories (where [PID] is the process ID for the process). For more information about the different files in the sub-directories to */proc/hpc/* and */proc/[PID]* see Kris Buytaert's openMosix HOWTO.

To view the setting in any of the existing files just use cat, more, less or similar UNIX command. The only directories where the files can be manually edited is */proc/hpc/admin* and */proc/hpc/decay*. You edit these files with the echo command. For example if you want to block the machine, so it doesn't accept remote processes migrating to it, you type:

echo 1 > /proc/hpc/admin/block

6.2 User-space tools

The user-space tools have a number of commands that can be used to administrate the cluster. The commands aren't fully described here, for more information read the commands man file with *man [command]* or have a look in Kris Buytaert's openMosix HOWTO.

/etc/init.d/openmosix is the script that is used to start and stop openMosix. The options to this script are *start*, *stop* and *status*.

The command *migrate* tries to send a process to another machine. The command's syntax is:

migrate [PID] [MOSIX_ID | home | balance]

The last option can be either a nodes openMosix ID, if the process should go to the machine with the lowest load (balance) or if it should migrate to its home node (home).

The command *mosmon* is a small monitor program that displays a bar chart, normally showing the loads on the various openMosix nodes. Alternatively, it can display the various processor speeds, or the amount of available vs. total memory. The program has an online help that you can get if you type h in the program, to quit just type q. Some of the keys that can be pressed in mosmon to display different variables are:

s	shows processor speeds
m	shows memory (used out of total)
r	shows memory (raw used/free out of total)
u	shows utilizability percentage
l	go back to showing loads
d	show also dead (configured but not-responding) nodes
D	stops showing dead nodes
y	shows the yardstick in use (e.g. speed of a standard processor)

If all the nodes doesn't fit one screen you can use the left/right arrow keys to move one node to the left or right and the n or p keys to move one screen to the left or right.

The command *mosctl* is the main configuration tool. With this command you can set and read the different settings that exists in the */proc/hpc/* interface. The settings are changed by that

you type the command mosctl with different options. The syntax for mosctl is:

mosctl { stay | nostay | lstay | nolstay | block | noblock | quiet | noquiet | nomfs | mfs | expel | bring | gettune | getyard | getdecay }

When the command is used with this syntax certain settings (mainly in */proc/hpc/admin*) is set or in the case of the 3 last ones shown. The options mean:

- *stay* sets so that there won't be any automatic process migration (is cancelled with *nostay*).
- *lstay* makes so that local processes doesn't migrate to other nodes but remote processes do (is cancelled by *nolstay*).
- *block* blocks arriving guest processes (is cancelled by *noblock*).
- *quiet* disables gathering of load-balancing information (this is enabled again by *noquiet*).
- *nomfs* disables the MFS (this is enabled again by *mfs*).
- *expel* sends away all the guest processes.
- *bring* brings all the migrated processes home
- *gettune* shows the current overhead parameters used by the kernel to estimate the "I/O factor" in its load-balancing.
- *getyard* shows the current yardstick (the processor speed of the most typical openMosix node).
- *getdecay* shows the current decay parameter (controls the gradual decay of old process statistics for the use of load-balancing).

mosctl whois [OpenMosix-ID | IP-address | hostname]

Resolves openMosix ID, IP-addresses and hostnames of the cluster.

mosctl { getload | getspeed | status | isup | getmem | getfree | getutil } [OpenMosix-ID]

Displays different status parameters on a certain node (defined by its openMosix ID).

- *getload* displays the current (openMosix-) load
- *getspeed* displays the current (openMosix-) speed.
- *status* displays the current status and configuration (stay, lstay, block, quiet).
- *isup* tells if a node is up or down.
- *getmem* shows the logical free memory.
- *getfree* shows physical free memory.
- *getutil* displays the utilization.

mosctl setyard [processor-type | number | this]

Sets a new yardstick value (can be good if the majority of the nodes are very different from the standard yardstick).

mosctl setspeed [numeric-value]

Overrides the node's idea of its own speed.

mosctl setdecay [interval] [slow] [fast]

Sets a new decay interval: *interval* in seconds, how much of 1000 to keep for *slow*-decaying processes and how much of 1000 to keep for *fast*-decaying processes. The interval must be within the range of 1-65535 and the slow and fast parameters in the range of 1-1000 (there the slow parameter is greater or equal to the fast parameter).

For more information on the different settings read mosctl's man page.

The command *mosrun* is used to execute jobs on the cluster with special openMosix settings on a specific node (or nodes). There are also several scripts that can be used to execute a job with a special pre-configured openMosix configuration. The scripts are: *nomig*, *runhome*, *runon*, *cpujob*, *iojob*, *nodecay*, *slowdecay* and *fastdecay*. Mosrun isn't needed to run processes that uses the openMosix migration but is mainly there if you want to control how the process you are starting should behave.

The command *setpe* is used to configure what nodes are in the cluster by reading a specified file (*setpe -w -f [filename]*). You can also read the current configuration with the program (*setpe -r*) plus shutting down openMosix by removing the configuration (*setpe -off*).

The commands *mtop* and *mps* are openMosix aware versions of the standard Linux commands *top* and *ps*. These commands have an extra column that shows what node the programs are running on (in a column marked *n#*). Only the programs that started on the node that mtop/mps is running on is shown and programs that hasn't migrated to another node has node number 0. The options used by these two programs are the same as the ones used for ordinary ps/top.

The tool *omdiscd* is an auto discovery daemon that can be used instead of */etc/mosix.map*. If you start omdiscd openMosix has to be shutdown:

/etc/init.d/openmosix stop

Then you start the auto discover daemon with the command *omdiscd*. If the machine has more than one network interface the -i option should be used. You can also start the daemon so it runs in the foreground sending messages and debugging output to standard error. The option is -n and without it all output should go to the syslog.
For example if you only want omdiscd to run in the foreground and also to search for openMosix nodes on the network that the eth1 interface is connected too then type:

omdiscd -n -i eth1

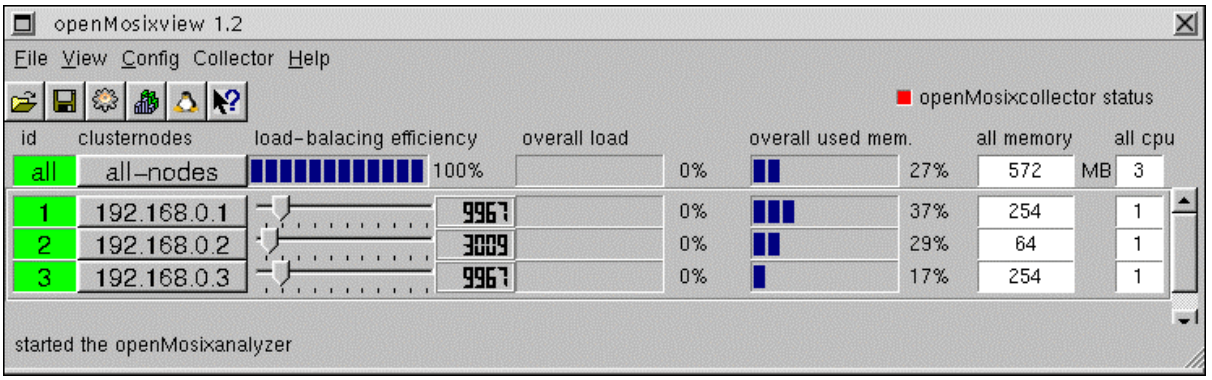
The auto discovery daemon comes with another command called *showmap*. This command shows the current nodes in the cluster.

6.3 openMosixview

The tool openMosixview is a graphical interface for doing certain administration and monitoring tasks on an openMosix cluster. OpenMosixview consists of five programs there among a main program (called just openMosixview) that is used to access the other four. When you run the command *openmosixview* do this as root (so you can change the different

settings) and don't start the program in the background since it wants to use the console that the program started from for error messages etc.

When openMosixview has started you will get a window that shows the nodes on your cluster and their status.



The status information is shown in several columns, with the top row showing the entire clusters status, these columns are:

- **Id:** The node number with a background color that shows if the node is up (green) or down (red).
- **Clusternodes:** A button with the nodes IP-address on, if the button is pressed a window shows up. This new window can be used to set some of the nodes settings (the window that shows up if the button *all-nodes* are pushed sets these settings for all the nodes). In this window you can:
  - Turn auto migration on and off.
  - Tell the node if it should talk to other nodes.
  - Set the node so local processes should stay on it.
  - Tell the nodes guest processes to leave the node.
  - Start and stop openMosix.
  - Open a console on the node.
  - Open the program openMosixprocs on the node.
  - Tell openMosixview on what machine the console and the openMosixprocs program should be displayed on (preferably the machine where you sit at the monitor and run X windows).
- **Load-balancing efficiency:** On the first row where the entire cluster information is shown displays how effective the load balancing is. On the rows that show information about specific nodes a slider exists that shows the current speed of the node and if the slider is moved this speed is changed. Next to the slider is the numerical value of the speed.
- **Overall load:** The load on the entire cluster and the different nodes.
- **Overall used mem:** The overall used memory on the entire cluster and the different nodes.
- **All memory:** The physical memory on the entire cluster and the specific nodes in megabytes.
- **All CPU:** The first row shows the number of CPU's on the entire cluster. The other rows show the number of CPU's on a specific node.

There are five drop-menus at the top of the program these are:

- **File:** In this menu you can either choose to quit openMosixview (Exit) or to get up the advanced execution window (run programm). If you choose the advanced execution window you'll first get to choose what program you want to run on the cluster. When the program has been chosen you'll get a window where you can set the options that the program should have and how you want it to run on the cluster (for example if you want the program to run on a specific node). These settings are similar to the ones that can be set by the program mosrun. When you have set the settings you should press the execute button that's in the window and the program will start.
- **View:** There you can set if you want a status bar at the bottom of the main window that shows what feature was last executed in openMosixview (statusbar).
- **Config:** Here you can set if you want to use SSH when openMosixview talks to the different nodes (use ssh). Here you can also save the configuration (save conf).
- **Collector:** Here you control the openMosixcollector. In this menu is a submenus (openMosixcollector) there you can *start*, *stop*, put in a *checkpoint* and save the openMosix history (*save history*). The status of the openMosixcollector is shown by a little status display (green for on and red for off) in the main window.
- **Help:** In this menu you can get up a window with an online help (*Help*), a mouse pointer that you can use to click on different parts of the windows to get a description what part in the program does (*What is this*) and to get a window with information about the current version of openMosixview (*About*).

In the main window there is also six buttons that represents different features in the program these are:

 This button has the same feature as the menu choice: File -> Run programm.

 This button saves the rsh/ssh configuration (same as Config -> save conf)

 Opens up openMosixprocs on the node that openMosixview is running on.
In openMosixprocs there is a list of all the processes on the current node (with a dropdown box to choose if you want to list a specific user's processes). If you double click on a listed process a window with information about the process is shown plus that you can tell the process to migrate, die and some other things. In the main openMosixprocs window there is also a button that brings up a window there you can manage the remote processes that have migrated to the node. In this window there is a tab for each remote process with information about the process and two buttons, one for migrating the process to its home node and one to migrate the process to the node with the lowest load.

 Opens the program openMosixanalyzer that shows a graphical display of the data collected with openMosixcollector.
In openMosixanalyser you can view a graph over the load () and memory () during the time openMosixcollector has been on by clicking the two icons to choose each graph. The window shows one graph per node in the cluster plus one for the entire cluster. Next to each graph are two icons one that shows the minimum, maximum and average load and memory () and one for printing out the graph on a printer (). A third icon () next to the ones that pull up the load and memory graphs starts up the program openMosixHistory that shows the processes on the node during certain times (chosen by

a slide bar on the top of the window).

 This button has the same feature as the menu choice: Help -> Help.

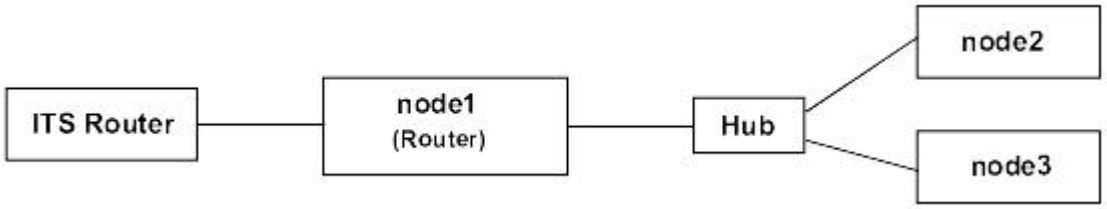
 This button has the same feature as the menu choice: Help -> What is this.

7 Links to more information

The official web page for openMosix: *http://www.openmosix.org*
The official web page for openMosixview: *http://www.openmosixview.com*
Kris Buytaert's openMosix HOWTO: *http://howto.ipng.be/openMosix-HOWTO*
OpenMosix Documentation Wiki: *http://howto.ipng.be/openMosixWiki/*

Appendix C openMosix configuration during the project

How the system was physically connected is presented in the picture below. Data about the nodes are listed in the table.



Hostname	IP on network card connected to hub	Hardware
node1	192.168.0.1	Celeron 1GHz, 256 MB RAM
node2	192.168.0.1	Pentium 300MHz, 64 MB RAM
node3	192.168.0.1	Celeron 1GHz, 256 MB RAM

When the performance tests was made the auto-discovery daemon hadn't been released yet and therefore the file /etc/mosix.map file was the one that was edited to get the cluster configured. This file was also the only file that was edited when openMosix was configured. Below it can be seen how that file looked like for the different cluster types that were used in the performance tests.

Homogeneous cluster

When doing the performance tests on a homogeneous cluster openMosix on node2 was shut down and the text in the file /etc/mosix.map on the two nodes (node1 and node3) was:

```
# MOSIX CONFIGURATION
# ===================
#
# Each line should contain 3 fields, mapping IP addresses to MOSIX node-
# numbers:
# 1) first MOSIX node-number in range.
# 2) IP address of the above node (or node-name from /etc/hosts).
# 3) number of nodes in this range.
#
# Example: 10 machines with IP 192.168.1.50 - 192.168.1.59
# 1        192.168.1.50     10
#
# MOSIX-#  IP  number-of-nodes
# ============================
1       192.168.0.1     1
2       192.168.0.3     1
```

Heterogeneous cluster with 2 computers

When doing the performance tests on a heterogeneous cluster openMosix on node1 was shut down and the text in the file /etc/mosix.map on the two nodes (node2 and node3) was:

```
# MOSIX CONFIGURATION
# ===================
#
# Each line should contain 3 fields, mapping IP addresses to MOSIX node-
numbers:
# 1) first MOSIX node-number in range.
# 2) IP address of the above node (or node-name from /etc/hosts).
# 3) number of nodes in this range.
#
# Example: 10 machines with IP 192.168.1.50 - 192.168.1.59
# 1        192.168.1.50     10
#
# MOSIX-#  IP  number-of-nodes
# ============================
1     192.168.0.2     2
```

Heterogeneous cluster with 3 computers

When doing the performance tests on a heterogeneous cluster the text in the file /etc/mosix.map on the three nodes (node1, node2 and node3) was:

```
# MOSIX CONFIGURATION
# ===================
#
# Each line should contain 3 fields, mapping IP addresses to MOSIX node-
numbers:
# 1) first MOSIX node-number in range.
# 2) IP address of the above node (or node-name from /etc/hosts).
# 3) number of nodes in this range.
#
# Example: 10 machines with IP 192.168.1.50 - 192.168.1.59
# 1        192.168.1.50     10
#
# MOSIX-#  IP  number-of-nodes
# ============================
1     192.168.0.1     3
```

Appendix D

Ganglia How-to for ITS

Emma Roos

emma@fukt.bth.se

Table of Contents

1 Brief introduction

This how-to describes how to install, administrate and use the clustering toolkit Ganglia. For an introduction about this clustering system see the report. The instructions for the Ganglia monitoring tools in this how-to (Ganglia Monitor Daemon (gmond), Ganglia Meta Daemon (gmetad) and Ganglia Metad Web Interface) is for version 2.5.0 and above, while for the execution environment (Authentication Daemon (authd) and Ganglia Execution system (gexec)) there hasn't been any new releases lately and therefore the instructions are for the latest version that existed by the time this how-to was written.

2 Requirements

The monitoring core can run on several sorts of architectures: Linux (Intel x86, Athlon, SPARC, Alpha, PowerPC and some others), Solaris, FreeBSD, AIX, IRIX, Tru64, HPUX, MacOS X and Windows (cygwin beta). To install with RPMs, a RedHat system running on an Intel x86 or Athlon machine is needed though. If the monitoring core should be installed on any other kind of system then it needs to be compiled from source.

The monitoring core needs multi cast enabled on the machines it is going to be installed on. You can see if multi cast is enabled by checking if the file */proc/net/igmp* exists. Multi cast is usually enabled in the standard kernel for Linux but if its not then you have to recompile the kernel with it enabled.

The web frontend needs to be installed on a machine that runs a web server. This machine also needs RRD Tool (*http://www.rrdtool.com/*) installed and the web frontend RPM currently only works on Intel x86 machines.

The execution environment currently only seem to work on Intel x86 and Athlon machines. The packages for xinetd and openSSL also need to be installed on the system.

3 Downloading

The different parts of Ganglia can be downloaded from:
http://ganglia.sourceforge.net/downloads.php.
If you are going to install via RPMs then there are two RPMs for the monitoring core, one with gmond and another with gmetad (Intel 386), that should be downloaded. There is one RPM for the web interface (Ganglia Gmetad Web Frontend (for no specific architecture)). When it comes to the execution environment you need the Authd and Gexec RPMs (either for Intel 386 or Athlon).

If you are going to compile Ganglia then you should choose the tar.gz files instead. The difference in files when it comes to the tar.gz's is that the monitoring core only has one tar.gz that includes both gmond and gmetad. The execution environment needs LibE and therefore this tar.gz should be downloaded too plus that, if you want gexec to talk to the monitoring

core, you also need the Monitoring Core Library (libganglia), this library only exists as an RPM (for Intel x86 and Athlon).

4 Installing with RPMs

4.1 Introduction

The files mentioned in the installation instructions are the versions that were listed on the Ganglia download web page at the time this how-to was written. This section only describes the installation on a RedHat system with RPMs. How to compile Ganglia is described in the next chapter in this how-to.

The latest versions of the monitoring core's RPMs doesn't seem to have gexec support enabled so if you want to run gexec on the machine you are currently installing the monitoring core on you have to compile it manually with gexec enabled as described in the next chapter. This problem might be fixed in the future.

4.2 Installing the monitoring core

4.2.1 Gmond

First install gmond on all the machines that you want to monitor with Ganglia with the rpm command (If you are updating the package change the i option to the U option.):

rpm -ivh ganglia-monitor-core-gmond-2.5.1-1.i386.rpm

The configuration for gmond exists in a file called */etc/gmond.conf*. The default configuration works on most clusters but sometimes its necessary to change the settings in this file. The different settings are described in the file just above the specific setting. To use a setting just remove the comment tag (#) in front of it and change it to the value you want. For example if you want to change the name of the cluster remove the # from:

 # name "My Cluster"

And change the *My Cluster* to what you want the cluster to be called. When this is done the line should look like this:

name "The name I've chosen".

If you like to test a configuration file you've created you can use the gmond command's --conf option. For example if you've created a file named *gmond-test.conf* that lies in the directory */etc* and wants to start the gmond daemon with these settings, then first shut down the currently running gmond daemon with */etc/init.d/gmond stop* and then start it with the new configuration with the command:

gmond --conf=/etc/gmond-test.conf

4.2.2 Gmetad

After that gmond is installed it's time to install gmetad on the machine/machines that you want to use to monitor the system (there among the machine that you want to run the web interface on) with the rpm command (If you are updating the package change the i option to the U option.):

rpm -ivh ganglia-monitor-core-gmetad-2.5.1-1.i386.rpm

The configuration of gmetad is in the file */etc/gmetad.conf* and just like */etc/gmond.conf* a description of the different settings exists above the said setting. Before gmetad can run properly you will need to define the "data sources" in the */etc/gmetad.conf* file to tell gmetad which gmond daemons to monitor. The format of the data_source lines should look like this:

data_source "my cluster" localhost host1 host2:port

There "my cluster" is the name of the cluster defined in */etc/gmond.conf* and the options after that are the different hosts running gmond that you want to use to check the cluster. Gmetad will check the first host in the list and if that host doesn't respond it will try the second host listed etc. If you don't define a port then gmetad will try the default port defined by the configuration files.

After that the data_source setting has been defined start gmetad with */etc/init.d/gmetad start*.

4.2.3 Web frontend

Install the web frontend RPM on the machine that you want it to run on, don't forget to have a web server on the machine, with the rpm command:

rpm -ivh ganglia-webfrontend-2.5.1-1.noarch.rpm

Test your installation by visiting the URL: *http://webserver/ganglia-webfrontend/index.php* in a web browser where *webserver* is the host name or IP of the machine that you installed the web frontend on.

4.3 Installing the execution environment

Before authd and gexec are installed on the nodes a pair of SSL keys needs to be created. This is done with the commands:

openssl genrsa -out auth_priv.pem
chmod 600 auth_priv.pem
openssl rsa -in auth_priv.pem -pubout auth_pub.pem

After you have done this copy auth_pub.pem to the */etc* directory on all the nodes that will receive gexec requests (most likely all the nodes on the cluster). This can be done with the command scp:

scp auth_pub.pem node1:/etc/auth_pub.pem
scp auth_pub.pem node2:/etc/auth_pub.pem

Then copy auth_priv.pem to all the nodes that are going to launch gexec requests (the machines in the cluster that users can log on to and run programs on). This can also be done with the scp command:

scp auth_priv.pem node1:/etc/auth_pub.pem

The machines that you will launch gexec requests on will need to have both auth_priv.pem and auth_pub.pem.

Now authd needs to be installed on all the machines that you want to run gexec on. This you do with the rpm command:

rpm -ivh authd-0.2.1-1.i386.rpm

Then its time to install gexec in the same way:

rpm -ivh gexec-0.3.4-1.i386.rpm

After that add gexec in */etc/services* under *# Local services*, the line should look like this:

gexec 2875/tcp # Caltech gexec

5 Installing from source

5.1 Introduction

The files mentioned in the installation instructions are the versions that were listed on the Ganglia download web page at the time this how-to was written.

5.2 Compiling the monitoring core

First unpack the tar.gz with the source that you've downloaded:

tar -xvzf ganglia-monitor-core-2.5.1.tar.gz

Then go into the directory that got created:

cd ganglia-monitor-core-2.5.1

If you are going to compile gmetad then you have to make sure that the configure script will find RRD Tool's library file *librrd.a* and header file *rrd.h* first. For the script to find these files they have to be found in */usr/local/lib* (for *librrd.a*) and */usr/local/include* (for *rrd.h*), this can be done by doing a soft-link to these files in these directories. For example if RRD Tool is installed in */usr/local/rrdtool* you should do something like this:

ln -s /usr/local/rrdtool/include/rrd.h /usr/local/include/
ln -s /usr/local/rrdtool/lib/librrd.a /usr/local/lib/

Now you should run the configuration script. If you want to install gmetad then add the --with-gmetad flag as an option and if you want gexec to run on the machine (and have the monitoring core aware of it) you also have to add the --enable-gexec flag. The command line should look something like this:

./configure -- with-gmetad --enable-gexec

Now run the command *make* and the programs will compile. Then run the command *make install* and after that copy the configuration file to the directory */etc/*:

cp ./gmond/gmond.conf /etc/

Edit the file if necessary and then it's time to do so that gmond is started at reboot:

cp ./gmond/gmond.init /etc/init.d/gmond
chkconfig --add gmond
/etc/init.d/gmond start

If you are installing on the machine that you are planning to run the web frontend on then you also have to install gmetad. First copy the configuration file to */etc/*:

cp ./gmetad/gmetad.conf /etc/

Edit the different settings in the file if necessary and add the *data_source* line in the same way as described in chapter 4.2.2. Then make so that gmetad can run and is started at reboot:

mkdir -p /var/lib/ganglia/rrds
chown -R nobody /var/lib/ganglia/rrds
cp ./gmetad/gmetad.init /etc/init.d/gmetad
chkconfig --add gmetad
/etc/init.d/gmetad start

5.3 Installing the web frontend

See that gmetad, a web server and RRD Tool exists on the machine that you want the web frontend on and if they aren't there install them. Also make sure that the web server knows how to handle PHP (for Apache the mod_php module needs to be enabled).

Copy the tar.gz with the web frontend files that was downloaded into your website tree (usually */var/www/html/*) and then unpack the file there:

cp ganglia-webfrontend-2.5.1.tar.gz /var/www/html/
cd /var/www/html/
tar -xvzf ganglia-webfrontend-2.5.1.tar.gz

Test your installation by visiting *http://webserver/ganglia-webfrontend/* in a web browser (*webserver* is the hostname/IP of your web server).

5.4 Compiling the execution environment

You first need to install libE. Unpack the tar.gz with the libE source you've downloaded:

tar -xvzf libe-0.2.2.tar.gz

Then go into the library that got created:

cd libe-0.2.2

Now run the configure script:

./configure

Then run the command *make* and after that install libE with *make install*.

When libE had been installed it's time to compile authd. First the SSL keys should be created and put in */etc/* as described in chapter 4.3. After this it's time to compile authd. First unpack the tar.gz with the authd source that you've downloaded:

tar -xzvf authd-0.2.1.tar.gz

After that go into the directory created (authd-0.2.1) and run the configure script:

./configure

Then run the command *make* and after that install authd with *make install*.

When authd has been installed it's time for gexec. First libganglia needs to be installed, this library only exists as a RPM package so install the RPM packages you've downloaded:

rpm -ivh ganglia-monitor-core-lib-2.5.1-1.i386.rpm

Now unpack the gexec source tar.gz:

tar -xvzf gexec-0.3.4.tar.gz

Now go into the directory created by the command:

cd *gexec-0.3.4*

Before you run the configure command and if you are going to run the monitoring core on the machine (and want gexec to interact with it) you first have to copy the directory *lib/ganglia/* from the monitoring core source directory into the directory created by the gexec tar.gz. For example if both the monitoring core (version 2.5.1) and gexec (0.3.4) tar.gz-files were

unpacked in the same directory (for example */var/tmp*) you should copy the directory like this:

cp –R /var/tmp/ganglia-monitor-core-2.5.1/lib/ganglia /var/tmp/gexec-0.3.4/

This will create a directory called ganglia in the gexec-0.3.4 directory with the same contents as the *ganglia-monitor-core-2.5.1/lib/ganglia* directory.

Now its time to run the configure script (with the --enable-ganglia flag if you are going to run the monitoring core on the same machine):

./configure --enable-ganglia

After this run the command *make* to compile gexec and the install it with *make install*. Last you should edit */etc/services* and add under *# Local services* the line:

gexec 2875/tcp # Caltech gexec

6 Administrating Ganglia

6.1 Configuration files

As described under the installation instructions the configuration for gmond and gmetad exists in the files */etc/gmond.conf* and */etc/gmetad.conf*. In these files the different settings are also described quite well.

When it comes to the web frontend most of its configuration options exists in the file *conf.php* that lies in the *ganglia-webfrontend* directory that lies under the web server's main directory for web pages (commonly */var/www/html/*). There you can for example change the location of gmetad and RRD Tool if they are installed elsewhere, you can also change how the different graphs should be displayed and some other settings. All these settings are described with comments in the file.

6.2 Command line tools

The monitoring core has two command line tools: The Ganglia Metric client and The Ganglia Status Client.

The **Ganglia Metric Client** (gmetric) is used to get different metric values from the different machines running gmond (that broadcasts on the same multi cast channel). The program formats and sends a special multi cast message to all machines with gmond that are listening.

All the metrics have: a name (option: -n or --name), value (option: -v or --value), type (option: -t or --type) and optionally units (option: -u or --units).
The name is what the metric should be called, value is the value of the metric generated by a program that exist on the machine, type defines what kind of data type the metric is in (string, 8, 16 and 32 bit integer (int8 etc), 8, 16 and 32 bit unsigned integer (uint8 etc), float and

double) and units print what the unit of the metric should be called. For example if there is a program called cputemp that shows the temperature of the machine's CPU in Celsius, then to send this metric to all the listening gmonds type:

gmetric --n temperature --v 'cputemp' -t int16 -u Celsius

To continuously check this metric you have to add it to the systems cron table (Read the man pages for crontabs with *man crontab* and *man 5 crontab* for more information how to do this).

There are also other options to the gmetric command. What these are and what they do can be seen by typing *gmetric --help*.

With the **Ganglia Status Client** (gstat) you can get a status report for the cluster. If you only type gstat you will get a small status report on the cluster looking something like this:

CLUSTER INFORMATION
Name: Exjobbskluster
Hosts: 2
Gexec Hosts: 0
Dead Hosts: 0
Localtime: Wed Dec 4 14:02:12 2002

CLUSTER HOSTS
Hostname LOAD CPU Gexec
CPUs (Procs/Total) [1, 5, 15min] [User, Nice, System, Idle]

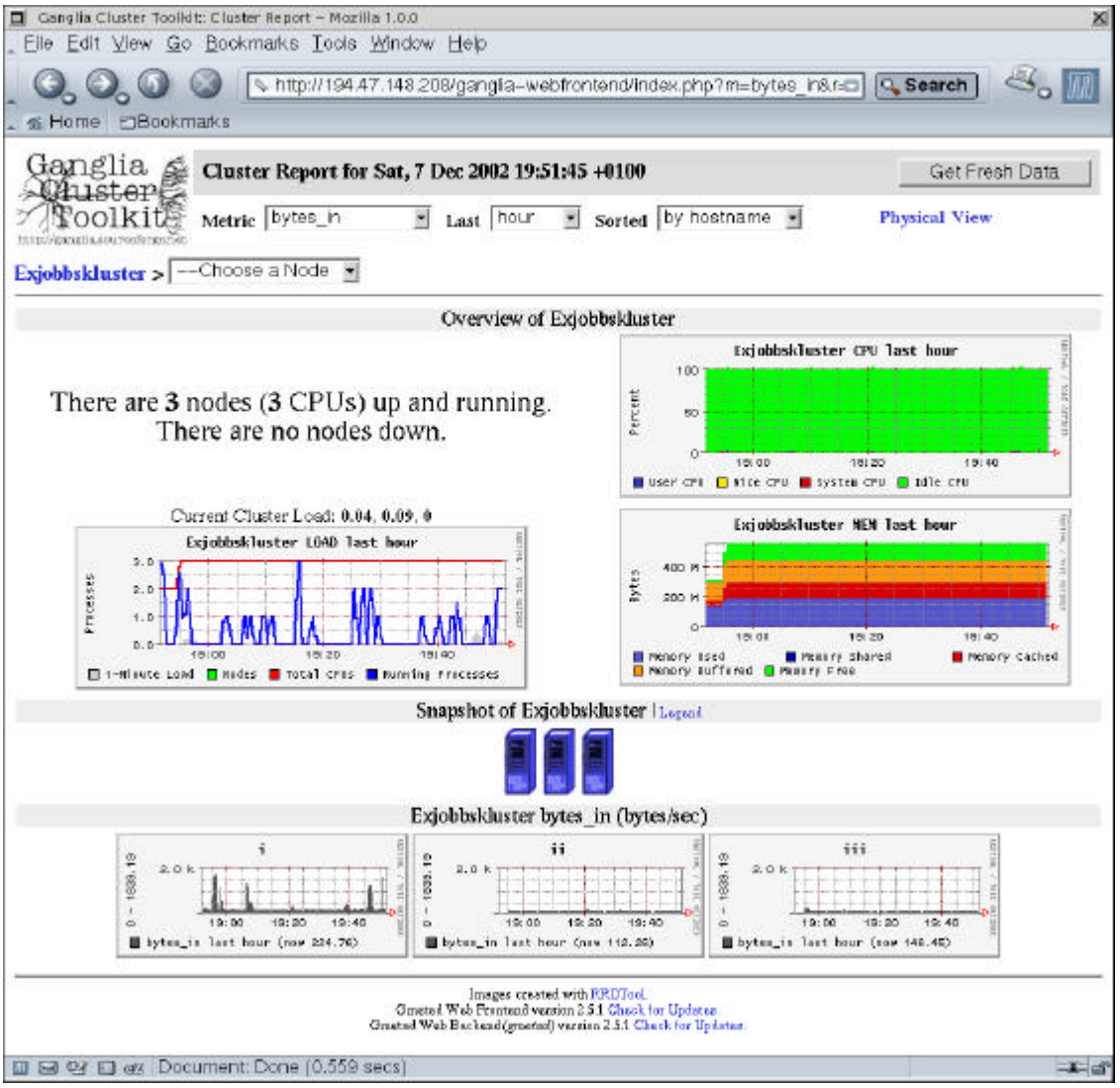
ii
1 (0/ 50) [0.00, 0.00, 0.00] [0.1, 0.0, 0.2, 100.0] ON

To get the syntax for the command use the help option: *gstat --help.* Some of the common options are:

-a (or --all) lists all the hosts (not just the ones running gexec)
-d (or --dead) lists only the hosts that are dead
-l (or --list) prints only the host list (the hosts running gexec)

6.3 The web frontend

The web frontend is a website that show different statistics and graphs of the cluster. On the main web page (index.php) you get an overview of the entire cluster:



The three graphs highest up on the web page shows the load, CPU usage and memory usage on the entire cluster the latest hour. The link "Physical View" just under the "Get Fresh Data" button goes to a page that lists the different hosts with some hardware information about them.

The graphs at the bottom of the web page shows the metric chosen, in the metric drop box at the top of the page, for all the nodes in the cluster. The other two drop boxes on the same row as the metric drop box defines over what time period (last hour, day, week, month, year) the graphs should be and in what order. The fourth drop box is used to choose a node and go to a web page that shows specific metrics and settings for that node, you can also get to this page by clicking on the metric graph for the specific machine or the small computer picture above the graphs.

On the web page with information about a specific node that you get to by doing as described above shows all the different metrics that gmond is monitoring. Some of the metrics are show as text ("Time and String Metrics" and "Constant Metrics") while the rest of them are shown in several graphs on this web page. There is also a drop box that you can use to change over what time period (last hour, day, week, month, year) the different graphs should be shown.

There is also a link, Host view, that goes to another page that show the hosts hardware- and software- configuration plus some other information.

7 Running programs with gexec

To run a program on several nodes the gexec command can be used. The main syntax for the command is: *gexec -n <number of nodes> command* where: *number of nodes* is the amount of nodes that the *command* you want to run should spawn on (where 0 is all nodes). For example if you want to see the uptime of the three first nodes that gexec know about you can do this with the command:

gexec -n 3 uptime

You will then for example get the output (the number in front of the output of the command is the virtual node number (vnn)):

0 10:26am up 16:38, 2 users, load average: 0.01, 0.01, 0.00
2 11:32am up 5 days, 21:52, 1 user, load average: 0.00, 0.00, 0.00
1 10:40am up 5 days, 21:54, 0 users, load average: 0.36, 0.28, 0.26

For gexec to know what nodes to use its supposed to interact with gmond to get a list of hosts running gexec. Unfortunately this doesn't seem to work with the latest versions of the monitoring core but hopefully it will be fixed in the future. Because of this the hosts running gexec (or the hosts that you want at the moment to run the programs you start) needs to be defined via the environment variable GEXEC_SVRS. To set this variable use the command export (or setenv if you use tcsh):

export GEXEC_SVRS="node1 node2 node3"

If the communication between gexec and gmond had worked properly (this might work with future releases and it does work with version 2.4.* of the monitoring core) you can also manually define the gmond servers that you want gexec to ask for nodes running gexec. This environment variable is GEXEC_GMOND_SVRS. You also set this variable with the export command:

export GEXEC_GMOND_SVRS="node1 node2:port"

Gexec also has a few other options the most useful of these are:

-h (--help)
: Prints the syntax of the command

-d (--detached)
: Gexec will spawn the jobs and then detach. This option is good if you for example are starting up daemons or other programs that should be running in the background.

-p (--prefix-type) [none | ip | vnn | host]
: This is the string that is before the output from gexec. The default is vnn (the unique identification number the node has), IP is the IP address of the node and host is the node's host name.

-P (--port)

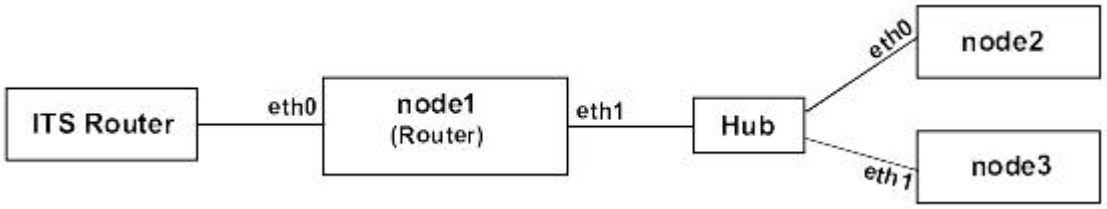
This option is used if gexec on the different machines is running on another port than the one described in */etc/services* on the machine that the command is run on.

8 Links to more information

The official Ganglia web page: *http://ganglia.sourceforge.net/*
The authd web page: *http://www.theether.org/authd/*
The gexec web page: *http://www.theether.org/gexec/*
The RRD Tool web page: *http://www.rrdtool.com/*

Appendix E Ganglia configuration during the project

How the system was physically connected is presented in the picture below. Data about the nodes are listed in the table.



Hostname	Network interface connected to the hub	IP on the network interface connected to the hub	Hardware
node1	eth1	192.168.0.1	Celeron 1GHz, 256 MB RAM
node2	eth0	192.168.0.2	Pentium 300MHz, 64 MB RAM
node3	eth1	192.168.0.3	Celeron 1GHz, 256 MB RAM

When the performance tests were made version 2.4.1 of the monitoring core was used and therefore the only file that was edited when Ganglia was configured was /etc/gmond.conf. The file on the different nodes looked like below (though node2 had mcast_if to eth0 instead of eth1):

```
# This is the configuration file for the Ganglia Monitor Daemon (gmond)
# Documentation can be found at http://ganglia.sourceforge.net/docs/
#
# To change a value from it's default simply uncomment the line
# and alter the value
######################
#
# The name of the cluster this node is a part of
# default: "unspecified"
name  "Exjobbstestkluster"

#
# The multicast channel for gmond to send/receive data on.  i
# NOTE: Must be in the multicast range from 224.0.0.0-239.255.255.255
# default: 239.2.11.71
#mcast_channel 239.2.11.71
#
# The multicast port for gmond to send/receive data on
# default: 8649
#mcast_port    8649
#
# The multicast interface for gmond to send/receive data on
# default: the kernel decides based on routing configuration
mcast_if  eth1
#
# The multicast Time-To-Live (TTL) for outgoing messages
# default: 1
# mcast_ttl  1
#
# The number of threads listening to multicast traffic
# default: 2
# mcast_threads 2
#
```

```
# Which port should gmond listen for XML requests on
# default: 8649
# xml_port     8649
#
# The number of threads answering XML requests
# default: 2
# xml_threads   2
#
# Hosts ASIDE from "127.0.0.1"/localhost and those multicasting
# on the same multicast channel which you will share your XML
# data with.  Multiple hosts are allowed on multiple lines.
# default: none
# trusted_hosts 1.1.1.1 1.1.1.2 1.1.1.3 \
# 2.3.2.3 3.4.3.4 5.6.5.6
#
# The number of nodes in your cluster.  This value is used in the
# creation of the cluster hash.
# default: 1024
 num_nodes  64
#
# The number of custom metrics this gmond will be storing.  This
# value is used in the creation of the host custom_metrics hash.
# default: 16
# num_custom_metrics 16
#
# Run gmond in "mute" mode.  Gmond will only listen to the multicast
# channel but will not send any data on the channel.
# default: off
# mute on
#
# Run gmond in "deaf" mode.  Gmond will only send data on the multicast
# channel but will not listen/store any data from the channel.
# default: off
# deaf on
#
# Run gmond in "debug" mode.  Gmond will not background.  Debug messages
# are sent to stdout.  Value from 0-100.  The higher the number the more
# detailed debugging information will be sent.
# default: 0
# debug_level 10
#
# If you don't want gmond to setuid, set this to "on"
# default: off
# no_setuid  on
#
# Which user should gmond run as?
# default: nobody
# setuid     nobody
#
# If you do not want this host to appear in the gexec host list, set
# this value to "on"
# default: off
# no_gexec   on
#
# If you want any host which connects to the gmond XML to receive
# data, then set this value to "on"
# default: off
# all_trusted on
```

When it came to specifying what computers gexec should start the programs on the variable GEXEC_SVRS were set. For example when the homogeneous cluster was tested then the the variable was set by *export GEXEC_SVRS=”node1 node2”*.

When the version 2.5.0 of the monitoring core came out and it was installed (this was after the performance tests) the file /etc/gmetad.conf also needed to be changed. This file looked like below on node1:

```
# This is an example of a Ganglia Meta Daemon configuration file
#                http://ganglia.sourceforge.net/
#
# $Id: gmetad.conf,v 1.3 2002/09/19 18:56:42 sacerdoti Exp $
#
# Setting the debug_level above zero will make gmetad output
# debugging information and stay in the foreground
# default: 0
# debug_level 10
#
# The data_source tag must immediately be followed by a unique
# string which identifies the source then a list of machines
# which service the data source in the format ip:port, or name:port.
# If a # port is not specified then 8649 (the default gmond port) is
# assumed.
# default: There is no default value
# data_source "my box" localhost  my.machine.edu:8655  1.2.3.5:8655
# data_source "another source"  1.3.4.7:8655  1.3.4.8
#
data_source "Exjobbskluster" localhost 192.168.0.2 192.168.0.3

# List of machines this gmetad will share XML with
# default: There is no default value
# trusted_hosts 127.0.0.1 169.229.50.165
#
# If you don't want gmetad to setuid then set this to off
# default: on
# setuid off
#
# User gmetad will setuid to (defaults to "nobody")
# default: "nobody"
# setuid_username "nobody"
#
# The port gmetad will answer requests for XML
# default: 8651
# xml_port 8651
#
# The number of threads answering XML requests
# default: 2
# server_threads 4
#
# Where gmetad stores its round-robin databases
# default: "/var/lib/ganglia/rrds"
# rrd_rootdir "/some/other/place"
```