

openMosix, past present and Future.

7. Juni 2004

Note légale

Dieser Beitrag ist lizenziert unter der GNU Free Documentation License.

Zusammenfassung

The openMosix software package turns networked computers running GNU/Linux into a cluster. It automatically balances the load between different nodes of the cluster, and nodes can join or leave the running cluster without disruption of the service. The load is spread out among nodes according to their connection and CPU speeds.

Since openMosix is part of the kernel and maintains full compatibility with Linux, a user's programs, files, and other resources will all work as before without any further changes. The casual user will not notice the difference between a Linux and an openMosix system. To her, the whole cluster will function as one (fast) GNU/Linux system.

openMosix is a Linux-kernel patch which provides full compatibility with standard Linux for IA32-compatible platforms. The internal load-balancing algorithm transparently migrates processes to other cluster members. The advantage is a better load-sharing between the nodes. The cluster itself tries to optimize utilization at any time (of course the sysadmin can affect the automatic load-balancing by manual configuration during runtime).

This transparent process-migration feature makes the whole cluster look like a BIG SMP-system with as many processors as available cluster-nodes (of course multiplied with X for X-processor systems such as dual/quad systems and so on). openMosix also provides a powerful optimized File System (oMFS) for HPC-applications, which unlike NFS provides cache, time stamp and link consistency.

With openMosix you can start a process on one machine and find out it actually runs on another machine in the cluster. Each process has its own Unique Home Node (UHN) where it gets created.

Migration means that a process is splitted in 2 parts, a user part and a system part. The user part will be moved to a remote node while the system part will stay on the UHN. This system-part is sometimes called the deputy process: this process takes care of resolving most of the system calls.

openMosix takes care of the communication between these 2 processes.

Recent improvements in openMosix, contain the initial implementation of Shared Memory Migration, CheckPointing Support, LoadLimitation, Autodiscovery tools, the General openMosix daemon and lots more.

1 Clustering, an overview

But what is clustering all about ? Basically there are 3 types of clusters, the most deployed ones are probably the Fail-over Cluster and the Load-balancing Cluster. High Performance Clusters are the currently lesser used type within the commercial market.

Fail-over Clusters consist of 2 or more network connected computers with a separate heartbeat connection between the 2 hosts. The Heartbeat connection between the 2 machines is being used to monitor whether all the services are still in use, as soon as a service on one machine breaks down the other machine tries to take over.

With load-balancing clusters the concept is that when a request for say a web-server comes in, the cluster checks which machine is the least busy and then sends the request to that machine. Actually most of the times a Load-balancing cluster is also Fail-over cluster but with the extra load balancing functionality and often with more nodes.

The last variation of clustering is the High Performance Computing Cluster, this machine is being configured specially to give data centers that require extreme performance the performance they need. Beowulf's

have been developed especially to give research facilities the computing speed they need. These kind of clusters also have some load-balancing features, they try to spread different processes to more machines in order to gain performance. The underlying mechanism for distributing work is that different routines of a program are executed on different systems. Special programming libraries are used to synchronize the systems and pull together the results.

So where does openMosix fit this picture ?, Actually openMosix is the alternative to the more classical Beowulf, but it's also complementary to this HPC technology.

2 openMosix Internally

Lets have a look inside, openMosix consists of a mechanism called Preemptive Process Migration (PPM) which is a set of algorithms for adaptive resource sharing. A process can be splitted into 2 parts, The system context of a process, called deputy, which stays on the processes home node and the working part of the process that moves to the remote node in the cluster.

Users can run parallel applications by initiating multiple processes in one node, and then allow the cluster to assign these processes to the best available resource at that time. If during execution of the process a new resources becomes available, the resource-sharing algorithms are designed to utilize these new resources by possible reassignment of the process amongst the nodes. This capability to assign and re-assign processes during runtime is particularly important for ease-of-use and to provide an efficient multi-user, time-sharing execution environment.

As we see every process has a home node called UHN ,the user context, called the remote, contains the program code, stack, data, memory-maps and registers of the process. The system context, called the deputy encapsulates the process when it is running in the kernel. While the remote can migrate many times between different nodes, the deputy is never migrated.

Both parts of the process, deputy and remote, are communicating with each other using a well defined API between the user-context and the system context. Therefore it is possible to intercept every interaction between these context's and forward this interaction across the network. Remote (guest) processes are not accessible to the other processes that run at the same node (locally or originated from other nodes) and vice versa. They do not belong to any particular user (on the remote node, where they run) nor can they be sent signals or otherwise being manipulated by local processes.

openMosix has no central control or master/slave relationship between the nodes. Scalability is archived by randomness in the system control algorithm, where each node does not attempt to determine the overall state of the cluster or any particular node. Each node sends at regular intervals information about its available resources to a randomly chosen subset of other nodes.

At the same time it maintains a small window, with the most recently arrived informations. The mathematical model for scheduling algorithm comes from the fields of economics research. Most important is that the available resources on a cluster of Linux computers are heterogeneous.

In effect, the cost for memory, CPU, process communication etc. are incomparable. They are not even measured in the same units. The latest algorithm employed by openMosix is very interesting because it tries to reconcile these differences based on economic principles and competitive analysis.

The key idea behind this strategy is to convert the total usage of several heterogeneous resources, such as memory, CPU, ... into a single homogeneous "cost".

Jobs are then assigned to the machine where they have the lowest costs, just like a market-oriented economy. At any time openMosix continuously attempts to optimize the resource allocation. Due to its decentralized algorithm , openMosix scales to well over a thousand nodes

Load-balancing is configurable during runtime using the /proc/hpc interface e.g openMosix is using a `fspeed` variable which is calculated at boot up from the openMosix kernel according to the speed of a Pentium 1Ghz. This value can be adjusted easily using either /proc/hpc, the command-line utilities or by the GUI. This directly affects the load balancing of the whole cluster.

3 Before getting openMosix

First of all, you must understand that openMosix is made up of a kernel patch and some user-space tools. The kernel patch is needed to make the kernel capable of talking to other openMosix-enabled machines on the

network. If you download openMosix as a binary package (such as an rpm file), you don't even need to take care about the kernel patch because the kernel has been patched and compiled with the most common default options and modules for you.

The user-space tools are needed in order to make an effective use of an openMosix-enabled kernel. They are needed to start/stop the migration daemon, the openMosix File System, to migrate jobs to certain nodes and other tasks which are usually accomplished with the help of our good old friend: the command line interface. About binary packages: the same as in the kernel patch goes for the user-space tools: if you install an rpm you don't need to care about compiling them or configuring anything; just let them install and run. That's all. Really :)

Once you get to the download page (which we'll talk about in a second), you'll need to get two distinct parts: the kernel and the user-space tools. You can either download two binary packages or get the kernel patch plus the user-space tools' sources. The kernel patch is usually named after this scheme: openMosix-x.y.z-w where x.y.z is the version of the vanilla Linux Kernel against which the patch should be applied and w is the patch revision for that particular kernel release. For the precompiled kernel binaries, please refer to the README-openMosix-kernel.txt file you'll find in the download page. This file also contains updated info about manually compiling a kernel.

About the user-space tools: you'll find those in a package named openMosix-tools. We use the terms user-space tools, userspace-tools and openMosix-tools interchangeably. Updated info about precompiled binaries and manually compiling the tools are also provided in the README-openMosix-tools.txt file. Please note that since version 0.3 of the openMosix-tools, the openmosix.map file is deprecated and the use of the autodiscovery daemon is highly encouraged since it tends to make your life easier.

4 Getting openMosix

You can download the latest versions of openMosix from http://sourceforge.net/project/showfiles.php?group_id=46729> You can either choose the binary (even in rpm) compiled for UP or SMP or download the source code. You will need both the kernel patch or binaries and the userland tools. Alternatively you can get the CVS version:

```
cvs -d:pserver:anonymous@cvs.openmosix.sourceforge.net:/cvsroot/openmosix login
cvs -z3 -d:pserver:anonymous@cvs.openmosix.sourceforge.net:/cvsroot/openmosix co linux-ope
cvs -z3 -d:pserver:anonymous@cvs.openmosix.sourceforge.net:/cvsroot/openmosix co userspace
```

At the password prompt, just type enter since you're doing an anonymous login. Please take care that CVS trees DO BREAK now and then and that it might not be the easiest way to install openMosix ;-)

5 Kernel Compilation

Always use pure vanilla kernel-sources from <http://www.kernel.org/>> to compile an openMosix kernel! Please be kind enough to download the kernel using a mirror near to you and always try and download patches to the latest kernel sources you do have instead of downloading the whole thing.

Be sure to use the right openMosix patch depending on the kernel-version. At the moment I write this, the latest 2.4 kernel is 2.4.22 so you should download the openMosix-2.4.22-3.bz patch, where the "3" stands for the patch revision (ie: the greater the revision number, the most recent it is).

Do not use the kernel that comes with any Linux-distribution: it won't work. These kernel sources get heavily patched by the distribution-makers so, applying the openMosix patch to such a kernel is going to fail for sure! Been there, done that: trust me ;-)

Download the actual version of the openMosix patch and move it in your kernel-source directory (e.g. /usr/src/linux-2.4.22). If your kernel-source directory is other than "/usr/src/linux-[version_number]" at least the creation of a symbolic link to "/usr/src/linux-[version_number]" is required.

Supposing you're the root user and you've downloaded the bzipped patch file in your home directory, apply the patch using the patch utility:

```
mv /root/openMosix-2.4.22-3.bz /usr/src/linux-2.4.22
cd /usr/src/linux-2.4.22
bzcat openMosix-2.4.23-3.bz | patch -Np1
```

The "patch" command should now display a list of patched files from the kernel-sources. Enable the openMosix related options in the kernel-configuration file, e.g.

```
...
CONFIG_MOSIX=y
# CONFIG_MOSIX_TOPOLOGY is not set
CONFIG_MOSIX_UDB=y
# CONFIG_MOSIX_DEBUG is not set
# CONFIG_MOSIX_CHEAT_MIGSELF is not set
CONFIG_MOSIX_WEEEEEEEEEE=y
CONFIG_MOSIX_DIAG=y
CONFIG_MOSIX_SECUREPORTS=y
CONFIG_MOSIX_DISCLOSURE=3
CONFIG_QKERNEL_EXT=y
CONFIG_MOSIX_DFSA=y
CONFIG_MOSIX_FS=y
CONFIG_MOSIX_PIPE_EXCEPTIONS=y
CONFIG_QOS_JID=y
...
```

Now compile it with:

```
make dep bzImage modules modules_install
```

After compilation install the new kernel with the openMosix options within your boot-loader; e.g. insert an entry for the new kernel in /etc/lilo.conf and run lilo after that, or grub or whatever pleases you.

Reboot and your first openMosix-cluster-node is up!

6 Easy Configuration

One node does not make a cluster, so you'll have to install the same kernel on another node. After rebooting the auto-discovery daemon (omdiscd) provides a way to automatically configure an openMosix cluster hence eliminating the need of a /etc/mosix.map or similar manual configurations. Auto-discovery uses multicast packages to notify other nodes that it is an openMosix node. This way adding an extra node to your mosix cluster means that you just have to start the omdiscd on your machine and it will join the cluster.

However there are some small requirements, Like with any openMosix cluster , you need to have networking configured correctly. mainly the routing. Without a default route, you must specify an interface to omdiscd with the -i option.

And have a look at your logfiles you should see something similar to this

```
Sep  2 10:00:49 oscar0 kernel: openMosix configuration changed: This is openMosix #2780 (o
Sep  2 10:00:49 oscar0 kernel: openMosix #2780 is at IP address 192.168.10.220
Sep  2 10:00:49 oscar0 kernel: openMosix #2638 is at IP address 192.168.10.78
Sep  2 10:00:49 oscar0 kernel: openMosix #2646 is at IP address 192.168.10.86
Sep  2 10:00:49 oscar0 kernel: openMosix #2627 is at IP address 192.168.10.67
Sep  2 10:00:49 oscar0 kernel: openMosix #2634 is at IP address 192.168.10.74
```

Congratulations , your openMosix cluster is now working.

Now lets still have a short look at the other tool , it's showmap. This tool will show you the newly auto generated openMosix map.

```
[root@oscar0 root]# showmap
My Node-Id: 0x0adc
```

Base Node-Id	Address	Count
0x0adc	192.168.10.220	1
0x0a4e	192.168.10.78	1
0x0a56	192.168.10.86	1
0x0a43	192.168.10.67	1
0x0a4a	192.168.10.74	1

Auto-discovery has some other features not listed here such as a routing mechanism for clusters with more than one network. More detailed information can be found in the README and DESIGN files in the user-land tools source tree.

More recent versions of the openMosix rc scripts will first verify whether an /etc/openmosix.map file or similar exists before trying to use autoconfiguration.

7 Basic Administration

openMosix provides the advantage of process migration to HPC-applications. The administrator can configure and tune the openMosix-cluster by using the openMosix-user-space-tools or the /proc/hpc interface.

Up till openMosix version 2.4.16 the /proc interface was named /proc/mosix ! Until openMosix version 2.4.17 it was named /proc/hpc.

The values in the flat files in the /proc/hpc/admin directory presenting the current configuration of the cluster. Hence the administrator can write its own values into these files to change the configuration during runtime, e.g.

```
echo 1 > /proc/hpc/admin/block
```

locks the arrival of remote processes

```
echo 1 > /proc/hpc/admin/bring
```

brings all migrated processes home ...

Next to changing values you can also get more information about your cluster from this interface.

```
cat /proc/hpc/nodes/[openMosix_ID]/CPUs
```

shows you how many CPU's node «openMosix_ID» has

```
cat /proc/hpc/nodes/[openMosix_ID]/load
```

shows the openMosix load of this node. Other files include mem,rmem,speed,status,tmem,util etc ..

Within the /proc/\$PID/ subsystem you can find important information about the status of a process, whether it locked (/proc/\$PID/lock) or send to another node (/proc/\$PID/where) , or even why that specific process can't move to another node (/proc/\$PID/cantmove) , how many times it migrated (/proc/\$PID/migrate)

As we will see later there are also graphical user interfaces to see or modify these values, but in a scripted environment these may come in handy !

8 the userspace-tools

We also have a bunch of command line tools that are providing easy administration to openMosix clusters. *migrate* [PID] [openMosix_ID] sends process \$PID to openmosix node \$openMosix_ID. *mon* is a ncurses-based terminal monitor several informations about the current status are displayed in bar-charts.

The main openMosix configuration utility is *mosctl*, with this tool you can define weather processes should stay or leave the current node, you can set a new speed for a certain node, you can ask migrated processes to come home again and much other features that are documented in the howto.

mosrun can be used to run a special configured command on a chosen node

```
mosrun [-h|openMosix_ID| list_of_openMosix_IDs] command [arguments]
```

The *mosrun* command can be executed with several more commandline options. To ease this up there are several preconfigured run-scripts for executing jobs with a special (openMosix) configuration.

Last but not least is the *setpe* tool. It is required for the configuration of a node. It is used to enable or disable openmosix on a node (*setpe -f /etc/openmosix.map* or *setpe -off*)

Additional to the /proc interface and the commandline-openMosix utilities (which are using the /proc interface) there is a patched "psänd" "topäavailable" (they are called "mpsänd" "mtop") which displays also the openMosix-node ID on a column. This is useful for finding out where a specific process is currently being computed.

This actually summarized the command line tools, but have a look at *openMosixview* which is a GUI for the most common administration tasks.

9 openMosixView

The aim of the openMosixview project is to provide an intuitive GUI for managing and monitoring an openMosix cluster. All common used actions which can be executed on an openMosix cluster are implemented into a nice QT based user interface. openMosixview has become one of the most used management-GUI's for openMosix cluster, now, and it contains 8 useful applications to monitor and administrate openMosix cluster by just a few mouse-clicks.

There is a full HTML-documentation about openMosixview included in every package. You find the start-page of the documentation in your openMosixview installation directory: `openmosixview/openmosixview/docs/en/index`

Installation of openMosixview: Download the latest version of openMosixview and unzip+untar the sources. Then copy the tarball to e.g. `/usr/local/`

```
gunzip openmosixview-1.5.tar.gz
tar -xvf openmosixview-1.5.tar
```

Automatic setup-script:

Just cd to the openmosixview-directory and execute

```
./setup [your_qt_2.3.x_installation_directory]
```

You can now execute mosixview

```
openmosixview
```

openMosixview displays all serious openMosix related informations of each node and the overall cluster state in the main application window. All other components included in the openMosix application suite are directly accessible from this main gui. Here an overview about the included openMosixview components :

openMosixprocs gives an overview what is running where

The second column of the "top-like" process monitor displays the openMosix-node ID of each process. 0 means local, all other values are remote nodes. Migrated processes are marked with a green icon and unmovable processes have a lock.

the openMosixanalyzer

The openMosixanalyzer provides a non-stop openMosix-history of your cluster. It displays the log-files created by openMosixcollector in a graphically way The openMosixanalyzer can analyze the current online-logfiles Older backups of your openMosixcollector history logs can be opened easily by the filemenu.

openMosixhistory displays the processlist from the past

openMosixhistory gives a detailed overview which process was running on which node. The openMosix-collector saves the processlist from the host the collector was started. This historic-processlist logs are browseable with openMosixhistory.

the famous openMosixmigmon

The openMosixmigmon is a monitor for migrations in openMosix-clusters. It displays all nodes as little penguins sitting in a circle.

The main penguin is the node on which openMosixmigmon runs and around this node it shows its processes also in a circle of small black squares.

If a process migrates to one of the nodes the node gets an own process-circle and the process moved from the main process-circle to the remote process-circle. Then the process is marked green and draws a line from its origin to its remote location to visualize the migration.

Drag'n Drop! The openMosixmigmon is fully Drag'n Drop enabled. You can grab (drag) any process and drop them to any of your nodes (those penguins) and the process will move there. If you double-click a process on a remote node it will be send home immediately.

Next to the openMosixView gui some other tools exist to monitory an openMosix luster

3dmosmon was developed by Johnny Cache and ported to openMosix by myself. It is a smart replacement for mosix's boring text based mon program. It includes an openGL client monitor and a server daemon and it does not require root privileges. The whole cluster is floating in a 3d space-environment and you can even "fly into it with the mouse.

openMosixwebview contributed by Ramon Ponds is an add-on to openMosixview. Its php-pages are reading the logfiles from the openMosixcollector and display cluster informations and nice statistics in a web-browser.

10 Checkpointing with openMosix + chpox

Since checkpointing is a very interesting and useful issue in high performance computing here now short introduction and example of how to use chpox on openMosix.

openMosix is not suitable for HighAvailability projects, but sometimes people want to implement a way of redundancy, in order to make sure that the results of a long-running process don't get lost due due a crash. Or even a student that reboots a machine in a multifunctional pc-lab.

With Checkpointing implemented, we can take a process , checkpoint it at regular intervals and continue the process at its last checkpoint in the unlikely event of a crash

1. *install chpox*: Unzip the chpox source and compile the chpox modules fitting to the openMosix kernel running on the cluster-nodes. (if you would like to test it use clusterKNOPPIX)
2. *install/inmod the chpox_mod into the running kernel*

```
insmod chpox_mod
```

3. register processes + their needed libraries (can be done automatically)

There are two flavors, /proc-interface or chpox-commandline tools. To register a process you just need to write to the /proc/chpox/register file e.g.

```
echo "[PID]:31:1:/tmp/proc-dump" > /proc/chox/register
```

The same registration can be also executed by the "chpoxctlülil:

```
chpoxctl add [PID] 31 1 /tmp/proc-dump
```

This registers PID and enables the possibility to checkpoint it.

4. *Add required libs for your process*

Do not forget to register the required libs for your process(es). Restoring the registered and checkpointed process will only work if you tell chpox which libraries are required for restoring, starting and running the process.

```
chpoxctl addlib [filename]
```

5. *Checkpoint the processes*

To checkpoint a process, just use the "kill" command and send signal 31:

```
kill -31 [PID]
```

This will "dump" the current state of the process PID to the /tmp/proc-dump file which will be used by the "restore" later.

6. *Restore processes*

To restore a process just pick its latest checkpoint-dump file of the registered process and execute:

```
ld-chpox [process-dump-file]
```

... and the process is running/working again

It might be problematic for parallel applications which are pawning and running process on remote hosts. chpox is (currently?) limited to working with non-interactive applications only. The chpox developers are working on support for sockets, shared-memory, IPC and threads. <http://www.cluster.kiev.ua/tasks/chpx_eng.html>

11 other useful contributions from the community

The openMosix community is a growing and active group of contributors to different parts of the openMosix project. Below we list a short section of interesting projects

The openMosixApplet, (by JP) is a cool Java applet which displays load statistics in your browser (on the way to be ported to the gomd)

The KludgeKollection, a smart set of script by Aaron Freed e.g to setup ssh access.

wmomload is a small openMosix load applet for windowmaker.

For quick testing openMosix there are some great bootable cd-distributions available which are providing a complete openMosix cluster out-of-the-box. No need to install or configure anything, just boot your nodes with those CD's.

Namely there is : ClusterKNOPPIX, PlumbOS, Quantian, Linux-live cd, CHAOS, BCCD and dyne:bolic. The latter one is even running on any Xbox which means you can make your Xbox join your openMosix cluster. What a fun.

Then there is openMosixLoaf which is an openMosix cluster on a single floppy disk.

Gentoo and Debian are providing easy-to-use openMosix packages for their distributions.

Even a small game for openMosix, called the openMosixblaster is available. You can shoot on your processes to migrate them. of course just for "funny administrators"

And certainly let's not forget GOMD, gomd is a daemon which executes commands and gets information from the nodes of an openMosix cluster. It has to run on every node in order to collect data, and it waits for commands to execute. gomd stands for «general openMosix daemon»

12 Running sequential and parallel applications on openMosix

First we will look at "running parallel applications" and then we will tell you how running sequential applications will take advantage on an openMosix cluster.

A parallel application, consists of more than one process, starts processes on every cluster-nodes and uses its own communication mechanism

To give the program-developers a helping hand there are lots of different parallel libraries that care e.g. about initializing the cluster, starting processes, inter-process communication etc. Two widely used libraries are used for this PVM and MPI. Lets look how openMosix optimizes applications created with those libraries

12.1 PVM applications on openMosix

PVM, in full words : parallel virtual machine Is a software which connects a number of networked systems as a parallel virtual machine. Typically those clusters and their PVM-applications are master-slave based. The "main" program is started on the master which starts a bunch of worker-tasks on the remote nodes by the rsh or ssh protocol as configured.

PVM has a master/slave concept of applications it has a static load/process assignment by PVM

By having dynamic load-balancing done by openMosix we minimize the configuration needs for PVM

Generally a PVM application running "natively" in the above described way on an openMosix cluster takes additional advantages of the dynamic load-balancing PVM tasks are distributed to the worker-nodes by the PVM-cluster init + spawn function (spawn is a kind of remote-fork) openMosix balances this statically assigned task during runtime to optimize the performance.

openMosix can also minimize the PVM-cluster configuration requirements to exactly one node *the master node*. To archive this goal the PVM cluster and application is just installed and configured on one node in the openMosix cluster. The result is a single node PVM cluster within a multi-node openMosix cluster. PVM spawns its tasks just on the PVM-master node and openMosix takes care of the full load + computing distribution and load-balancing. In this scenario PVM is just used to parallelize the application. By using the PVM library-functions the "problem to solve" is splitted into multiple sub-tasks which are then balanced by the openMosix single system cluster.

All yet used PVM applications are taking advantages by running on openMosix. PVM and openMosix are "good friends" and PVM is quite easy to learn. If PVM is used as a single-node PVM cluster within an openMosix cluster there is no additional configuration requirements at all, most of the times PVM is even pre-installed + configured by default.

To give some examples of programs using PVM :

1. octave, a high-level language, primarily intended for numerical computations
2. pvm gmake, a distributed gnu make
3. pvm-povray, a parallel povray renderer
4. parallel lil-gp, a parallel version of the genetic programming kernel "Lil-gp"

The one and only negative aspect of PVM is that the further development is mostly stopped. PVM is still widely used, there are lots of applications for PVM, documentation and active news.groups are still available.

12.2 MPI applications on openMosix

MPI is a standard for parallel computing and stands for "message passing interface". There are two mainstream implementations of MPI which are LAM, (Local Area Multicomputer) and MPICH, a freely available, portable implementation of MPI, the Standard for message-passing libraries.

Both MPI implementations are using the same MPI so that MPI programs are usually compileable + runnable on LAM and MPICH.

There are also other MPI implementations available with additional or enhanced functionality which are not further explained here.

The concept of the master-slave programming model is similar to that of PVM. MPI provides a lot specialized functions for passing messages from task to tasks, or better from processes to processes like for example broadcast, reduce, gather,

A common execution model of MPI is MPSD, multiple program single data. It means that the same program is running on multiple processors which are MPI cluster-nodes. The program task running on each node synchronizing variables and exchanging data by e.g. broadcasting + gathering. Each program task computes a part of the data and the sub-results are being combined afterwards.

Basically MPI and openMosix are friends and MPI applications are benefiting from the dynamic load-balancing on openMosix. MPI is just very aggressive in using the network-connections between the MPI-tasks to exchange data. Often MPI applications requiring synchronization of all running tasks which can soon become a bottleneck in the cluster -> this is a well know limitations of master-slave-based clustering-concepts. In this scenario having the MPI task migrated by openMosix on remote nodes other than their MPI-home-node will cause an additional network-utilization which won't be a benefit for the MPI application.

Please notice that this is NOT an openMosix limitation but a that of some "fancy programmed" MPI-application. In most cases those bottle-necks can be solved by increasing the network speed, bandwidth and latency. Also having enough memory on all nodes (including swap) is a serious issue since the MPI-program footstamp in the nodes memory can be quite big dependent on the application and data to compute.

the openMosix patch for MPICH For the MPICH implementation there is a small patch available which exchanges the use of "rsh/ssh" by "mosrun". This has the great advantage to reduce the overhead of the remote execution calls. The MPICH processes won't be spreaded to the remote nodes by rsh/ssh but starting as "local" but directly migrated to a remote node by the mosrun util.

12.3 sequential applications on openMosix

Even pure sequential/regular applications take advantages by running them on the openMosix platform. Those regular application task are performing computation on a data-section. To archive parallelization one has to think of how the "problem to solve by the application" can be divided into sub-tasks, or better sub-problem-solvers. In most cases this can be done by simply splitting the data to compute into several parts and run multiple instances of the program on each data-part.

This is known as MPMD, multiple program multiple data.

To ease up starting of those sub-tasks to gain parallelization the openMosix API provides functions and examples e.g. for dynamically process creation according the number of nodes We can now have a small wrapper script to start the sequential application which e.g. finds out how many nodes are available, then splits the data according the number of nodes and forks a sub-task on each part automatically.

some sequential application examples are : seti -> seti@home openMosix group , povray, number-cruncher applications, password cracker, mp3 converter, shell scripts, compiling, mostly anything ;)

Also users can benefit from openMosix if their workstation is part of the cluster. Every user can simply start his computation tasks and openMosix will balance all tasks on the whole cluster. Using the loadlimit feature each user or the administrator can decide how many resources each single workstation should spend to the cluster.

13 Future Directions in openMosix

People often ask what the future developments in openMosix will be. A lot of new features have already been implemented during the past year. We currently have autodiscovery (omdisc), clustermask, loadlimit support etc, available as default stable options.

Some other options are more experimental and not yet in the main openMosix three, the most tested and well known feature is the MigSHM patch, which is currently undergoing thorough revisions, and will enter the main openMosix three as soon as it is stable.

Keeping up with the kernel releases is also a major part of openMosix development. As I write this latest stable openMosix release is 2.4.22-om3, the vanilla kernel is at 2.6.5 and in 2.4 series we are at 2.4.26. We are working on porting the 2.4.22-om3 patch to 2.4.23 and up. New features are being implemented in 2.4.25-6. And we are also working at an openMosix port for the 2.6 series, where we are targeting at 2.6.3 as our first test release.

Migratable Sockets, a replacement for th oMFS , a port to Opteron, and someday maybe to other architectures, are the future directions of openMosix