Expert Linux

MOSIX, POSIX and Linux

In the first of a series of articles on clustering, **David Coulson** takes a look at MOSIX

Clustering is a hot topic at the moment in computing, particularly since companies like DreamWorks are employing the use of Linux clusters for their render farms to produce movies like *Shrek*. Render farms usually distribute individual tasks to each node in the cluster, rather than spreading out the same task between the group, so that they rely on custom job management software.

Other types of cluster can be far more simple, and the most basic is probably a web server and database server used to serve content. Even though each machine has its own unique functionality, and it's certainly not greatly distributed, the load is still spread among more than one machine, so it is just as much of a cluster as a five hundred node farm used for scientific computation.

The first thing that people usually think of when you mention Linux and clustering in the same sentence is Beowulf. Many people are under the misconception that Beowulf is a software package, or configuration, which aids clustering. But, in fact, a Beowulf is a collection of machines which perform process distribution, using either MOSIX, PVM, or customised software. There is a whole host of different applications and utilities that can be used to build a Beowulf. As with anything of this nature, the more network bandwidth that there is available, the better. And if you want a cluster that is going to do its job properly, you really need 100BaseTx and, ideally, a decent switch rather than a hub.

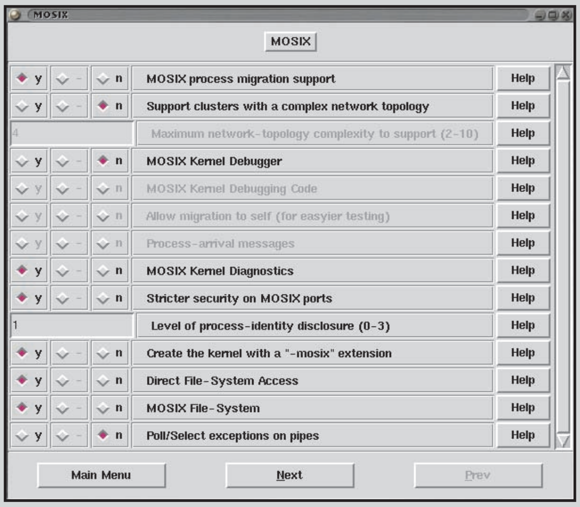
Over the next few months we're going to be looking at the various different clustering concepts under Linux, particularly in the area of high availability and service redundancy. Just to be awkward, though, we're going to start with the MOSIX kernel patch which lacks both of those benefits, but since it enables easy process distribution among a group of Linux machines, it's a fairly simple method of making use of your older machines. PCP

PCPlus SUPER DISC **Related files on SuperDisc 1**
PATH: linux

David Coulson
dcoulson@pcpmag.co.uk

PCPlus
NEXT MONTH
We're going to look at ipvs and high availability networking under Linux

PCPlus SEE YOUR SUPER DISC



MOSIX

- [✓ y] - n — MOSIX process migration support — Help
- y - [✓ n] — Support clusters with a complex network topology — Help
- 4 — Maximum network-topology complexity to support (2-10) — Help
- y - [✓ n] — MOSIX Kernel Debugger — Help
- y - n — MOSIX Kernel Debugging Code — Help
- y - n — Allow migration to self (for easyier testing) — Help
- y - n — Process-arrival messages — Help
- [✓ y] - n — MOSIX Kernel Diagnostics — Help
- [✓ y] - n — Stricter security on MOSIX ports — Help
- 1 — Level of process-identity disclosure (0-3) — Help
- [✓ y] - n — Create the kernel with a "-mosix" extension — Help
- [✓ y] - n — Direct File-System Access — Help
- [✓ y] - n — MOSIX File-System — Help
- y - [✓ n] — Poll/Select exceptions on pipes — Help

MOSIX has quite a few kernel options, although the defaults are perfectly adequate.

Installing MOSIX

Before going to the effort of installing MOSIX, it's worth making sure that it's actually going to be useful for your tasks

There are two types of cluster: where there is a single host node – all processes start on this and migrate out to each node in the cluster; and where you have a set of nodes, each spawning processes and migrating them out to other nodes. In either case, MOSIX relies on the processes to be heavy on computation and low on I/O. I/O can be anything from disc access to network usage, so a process like httpd from Apache isn't going to be of much use to MOSIX, so it will just stay put on the host which started it. More CPU hungry processes, such as gcc or seti@home, will happily migrate out to other nodes using MOSIX. And if you spend much of your time waiting for things to compile, or perform heavy computational tasks, then MOSIX will be of benefit.

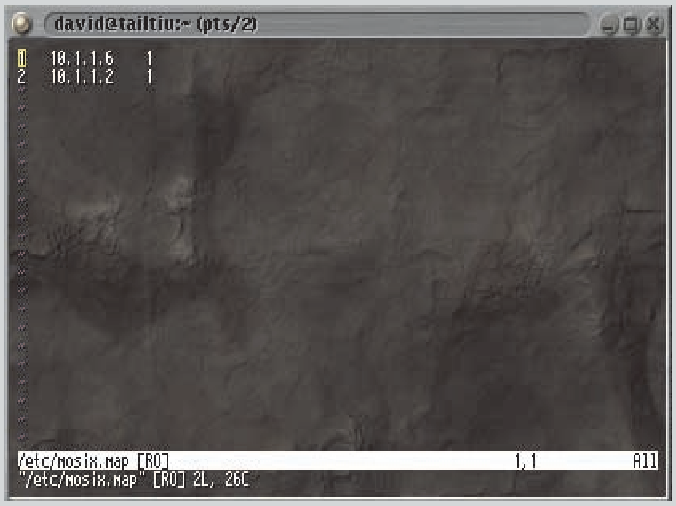
MOSIX requires both a set of command line utilities, as well as a kernel patch, all of which must be installed on any machine which you want to be included in the cluster. It's important that all nodes in the cluster are running the same release of MOSIX, which is 1.3.0 for Linux 2.4.9 at the moment. There is also a release of MOSIX for the 2.2.19 kernel, if you are still using the 2.2 series. The package which you download from mosix.org has both an automated and manual installation, and we're going to look at the latter, since it enables you to understand exactly what is going on.

First, we need a clean 2.4.9 kernel tree. MOSIX patches many different files, and if you try to patch it 2.4.9-ac or 2.4.10-pre, then you'll find you'll get an awful lot of rejected patches and you'll end up in a mess. If you need ext3 or xfs, along with MOSIX, you may find it easier to apply those after the MOSIX patch. Upon untaring MOSIX-1.3.0.tar.gz, there will be a patches.2.4.9 file, which is the kernel patch. This should be applied using the following:

```
cd /usr/src/linux-2.4.9
patch -p0 < ~/MOSIX-1.3.0/patches.2.4.9
```

You can then just build your kernel as you would normally, although you must ensure that support for the /proc filesystem is enabled, but it's quite unlikely that /proc is disabled. If you have an existing kernel source, you can copy over the .config out of there, into the linux-2.4.9 directory and run make oldconfig. It will ask you a few questions about which MOSIX options you need, then will continue to search for any new options.

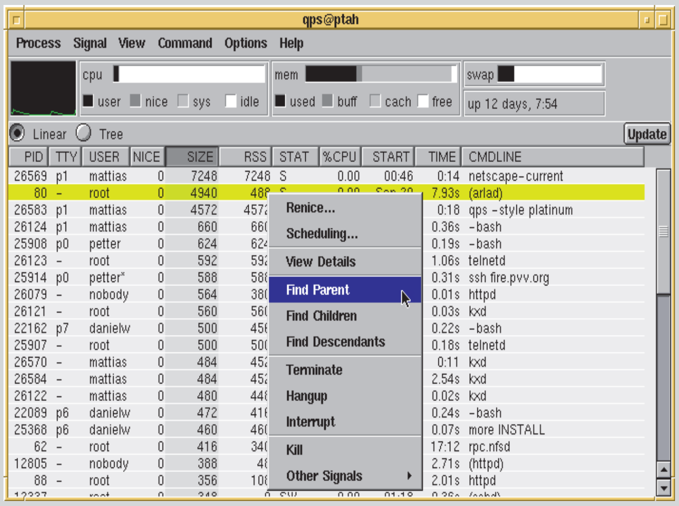
The default MOSIX options should be sufficient, although you may like to use DFSA and MFS, which enable nodes to transfer data more efficiently. The -mosix extension on the kernel is useful so you can easily spot if the machine is using the wrong kernel.



```
1  10.1.1.6    1
2  10.1.1.2    1
```

```
/etc/mosix.map [RO]                      1,1          All
"/etc/mosix.map" [RO] 2L, 26C
```

The /etc/mosix.map file identifies each of your nodes, and tells MOSIX how your network is organised.



```
up 12 days, 7:54

  PID TTY USER    NICE  SIZE   RSS STAT  %CPU   START   TIME CMDLINE
26569 p1  mattias    0  7248  7248 S     0.00   00:46   0:14 netscape-current
   80 -   root       0  4940   488                     7.93s (arlad)
26583 p1  mattias    0  4572  4572                      0:18 qps -style platinum
26124 p1  mattias    0   660   660                     0.36s -bash
25908 p0  petter     0   624   624                     0.19s -bash
26123 -   root       0   592   592                     1.06s telnetd
25914 p0  petter*    0   588   588                     0.31s ssh fire.pvv.org
26079 -   nobody     0   564   380                     0.01s httpd
26121 -   root       0   560   560                     0.03s kxd
22162 p7  danielw    0   500   456                     0.22s -bash
25907 -   root       0   500   500                     0.18s telnetd
26570 -   mattias    0   484   452                      0:11 kxd
26584 -   mattias    0   484   452                     2.54s kxd
26122 -   mattias    0   480   448                     0.02s kxd
22089 p6  danielw    0   472   416                     0.24s -bash
25368 p6  danielw    0   460   460                     0.07s more INSTALL
   62 -   root       0   416   340                     17:12 rpc.nfsd
12805 -   nobody     0   388    48                     2.71s (httpd)
   88 -   root       0   356   108                     2.01s httpd
```

As well as mon, qps (www.d.kth.se/~f91-men/qps) provides you with some MOSIX process information.

The complex network topology option is required to set up separate nodes on different subnets, so the MOSIX IP packets need to be routed. It's important that all nodes in the cluster have this selected, as well as the 'Maximum network topology complexity', which is the number of separate subnets you have. Since you get maximum network performance from a single network, it's unlikely that you will need to use this, but if you have a fairly large network, with bits and pieces spread all over the place, telling MOSIX how the network is organised makes it much quicker.

Building the binaries

Once you've got the kernel built, you can start to build the binary utilities which MOSIX needs to work. You don't need to reboot into your MOSIX kernel at this point, but you do need the patched kernel set up so that the utilities can find the required header files in /usr/include/linux/. They live in the user.tar file, so you'll need to tar xvf user.tar within MOSIX-1.3.0. It untars itself into the MOSIX-1.3.0 directory, so you just need to run 'make' from there, then run 'make install' to install them in the appropriate places. There is a manuals.tar file, which should be untared into /usr/local/man, and contains the manual pages for each of the installed binaries.

The only real configuration file MOSIX has is /etc/mosix.map, which defines which nodes are available. This must be exactly the same on all machines in the cluster, otherwise you'll find things don't work as expected. It's actually quite a complicated little file, particularly if you're using more than one network.

The file has three fields. Firstly, there is the node number, then the IP, then the number of nodes in that range. So, for example, if 10.1.1.1 through to 10.1.1.4 were the MOSIX nodes you would do:

```
1 10.1.1.1 4
```

or

```
1 10.1.1.1 1
2 10.1.1.2 1
3 10.1.1.3 1
4 10.1.1.4 1
```

FORUM www.futureforums.co.uk/pcplus

DFSA and MFS

High I/O processes won't migrate under MOSIX, but DFSA improves file access between nodes

Both DFSA and MFS are useful filesystems when you're running a MOSIX cluster. MFS is a method of providing access to filesystems located on a remote machine, so if a process has been migrated from a host to a separate node, the process does not have to migrate back to the host to perform a filesystem operation. However, MFS requires DFSA to be set up on any cluster node, which is to have its filesystems accessed by another node, so it requires a little bit of configuration on all nodes.

DFSA is handled as a mount option. Any filesystem on the node that you want to enable access to by a remote process needs to be remounted so that DFSA is enabled. You are allowed to enable DFSA on up to eight filesystems, so if you have more than that, a little kernel hacking is required. The best way to enable DFSA on your filesystems is to edit /etc/rc.d/init.d/mosix and have all of your filesystems remounted with the -odfsa=x option, where x is a unique index number for the filesystem. x can be anything from one to eight, and a value of zero removes the DFSA association from the filesystem.

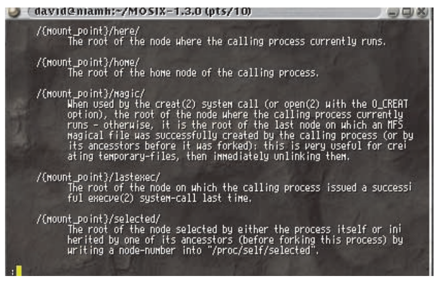
You might want to put something similar to this in init.d/mosix:

```
mount -o remount / -odfsa=1
mount -o remount /usr -odfsa=2
mount -o remount /usr/local -odfsa=3
mount -o remount /var -odfsa=4
mount -o remount /home -odfsa=5
mount -o remount /tmp -odfsa=6
```

Then, add this in the section of init.d/mosix where MOSIX is disabled:

```
mount -o remount / -odfsa=0
mount -o remount /usr -odfsa=0
mount -o remount /usr/local -odfsa=0
mount -o remount /var -odfsa=0
mount -o remount /home -odfsa=0
mount -o remount /tmp -odfsa=0
```

MFS is easily set up from there, and is generally mounted under /mfs:



```
/{mount_point}/here/
       The root of the node where the calling process currently runs.

/{mount_point}/home/
       The root of the home node of the calling process.

/{mount_point}/magic/
       When used by the creat(2) system call (or open(2) with the O_CREAT
       option), the root of the node where the calling process currently
       runs - otherwise, it is the root of the last node on which an MFS
       magical file was successfully created by the calling process (or by
       its ancestors before it was forked): this is very useful for crei
       ating temporary-files, then immediately unlinking them.

/{mount_point}/lastexec/
       The root of the node on which the calling process issued a successi
       ful execve(2) system-call last time.

/{mount_point}/selected/
       The root of the node selected by either the process itself or ini
       herited by one of its ancestors (before forking this process) by
       writing a node-number into "/proc/self/selected".
```

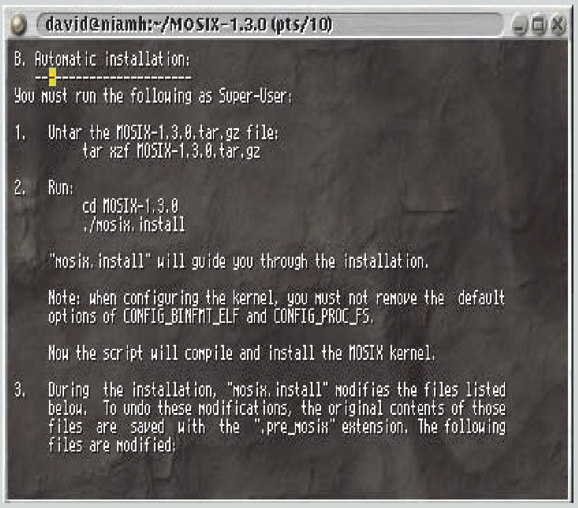
MFS has a number of special directories, which you can use to find out node information.

```
mount -t mfs mfs /mfs
```

or add an entry to /etc/fstab:

```
mfs /mnfs mfs defaults 0 0
```

You will also need to add mfs to the list of remote filesystems in /etc/cron.daily/slocate.cron. Once mfs is mounted, you can access the remote filesystems of each node by accessing /mfs/node_number/. Of course, for it to work correctly, you need to ensure that the UID and GID entries are exactly the same on all nodes, so the use of NIS would be very beneficial.



```
david@niamh:~/MOSIX-1.3.0 (pts/10)

B. Automatic installation:
   ----------------------
You must run the following as Super-User:

1.   Untar the MOSIX-1.3.0.tar.gz file:
          tar xzf MOSIX-1.3.0.tar.gz

2.   Run:
          cd MOSIX-1.3.0
          ./mosix.install

     "mosix.install" will guide you through the installation.

     Note: when configuring the kernel, you must not remove the  default
     options of CONFIG_BINFMT_ELF and CONFIG_PROC_FS.

     Now the script will compile and install the MOSIX kernel.

3.   During  the installation, "mosix.install" modifies the files listed
     below.  To undo these modifications, the original contents of those
     files  are  saved  with  the  ".pre_mosix" extension. The following
     files are modified:
```

↑To make things that bit easier, MOSIX also has a automatic installation script. But where's the fun in that?

You can only set a range great than one if the IPs are sequential, so 10.1.1.5 and 10.1.1.6 could be set as:

```
1 10.1.1.5 2
```

But if you were only using 10.1.1.5 and 10.1.1.7, you would need to use two separate entries, each with a range of one. It gets a little more confusing once you mix in a separate network, and if you wanted to add 10.1.2.3 and 10.1.2.4 to your cluster, which were routed through 10.1.1.4 that also had the IP 10.1.2.1 you would do:

```
1 10.1.1.1 3
4 10.1.1.4 1
4 10.1.2.1 ALIAS
5 10.1.2.3 2
```

Here, we tell MOSIX that 10.1.1.4 and 10.1.2.1 are really the same machine. In a case where you have to route between nodes, you need to edit /etc/mosgates to tell MOSIX the maximum number of hops to another node. If there are two networks, with a single gateway machine between them, there there is one hop to the other network. However, if you've got two networks branching off one, then for the middle network there is one hop to each other network, but for the outer two there will be two hops, so the /etc/mosgates file will have to be slightly different. MOSIX will try to figure out which entry in /etc/mosix.map belongs to the local machine by matching its hostname against an IP. But if it doesn't like the way your machine is set up, you can set the node number in /etc/mospe, which should only contain the node number which belongs to the local machine, as defined in /etc/mosix.map.

Set the options

Now, we just need to make sure all the MOSIX options are set up when the machine reboots, which requires a init.d file. You can just copy mosix.init out of MOSIX-1.3.0 into /etc/rc.d/init.d or, if you use Debian, /etc/init.d/ as mosix and then set up MOSIX for the various run levels. It's very important, in fact it is critical, that MOSIX comes up after your network comes up, and goes down before the network. Under Red Hat, the network initialisation scripts usually have a priority of ten for up and 90 for down, and MOSIX is generally set up with an up priority of 95 so that it's the last thing, bar maybe linuxconf, to come up. It also has a down value of five so that it's the first thing to go down.

Once you are sure that everything is set up satisfactorily, you can go ahead and reboot your nodes into the MOSIX kernel. It doesn't make much difference about the order of the nodes coming up, as MOSIX is a cooperative system, and there isn't a controlling node, that is, until processes start migrating away from the host node. If you want to see what your cluster is up to, and where specific processes have migrated to, you can use the mon utility to view node load information.

→Setting up MOSIX nodes

Once you've got your MOSIX kernel up and running, it takes a little tweaking of the system before it will run happily

It's important that some processes don't migrate away from the host node, otherwise all sorts of nasty things happen. The easiest way to do this is to disable migration for everything started by init.d, or via inittab, then individually enable them to migrate away from the host.

The main things to stop are rc.sysinit, shutdown and update, all of which are launched out of /etc/inittab at various times. We stop these migrating by running them through mosrun -h, so

```
ca::ctrlaltdel:/sbin/shutdown
-t3 -r now
```

becomes

```
ca::ctrlaltdel:mosrun -h
/sbin/shutdown -t3 -r now
```

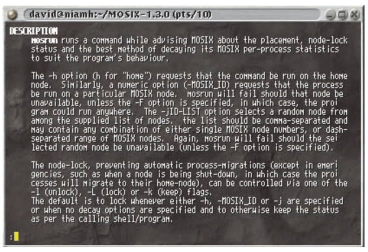
This needs to be repeated for everything else mentioned, as well as each of the run level entries, such as:

```
l3:3:wait:/etc/rc.d/rc 3
```

However, since we need some things to migrate, we have to edit their init.d scripts. atd can be migrated, so we need to add the following to the top of /etc/rc.d/init.d/atd:

```
echo 0 > /proc/$$/lock
```

Most services spawned out of inetd shouldn't be migrated, but telnet, rsh



```
david@niamh:~/MOSIX-1.3.0 (pts/10)

DESCRIPTION
       mosrun runs a command while advising MOSIX about the placement, node-lock
       status and the best method of decaying its MOSIX per-process statistics
       to suit the program's behaviour.

       The -h option (h for "home") requests that the command be run on the home
       node.  Similarly, a numeric option (-MOSIX_ID) requests that the process
       be run on a particular MOSIX node.  mosrun will fail should that node be
       unavailable, unless the -F option is specified, in which case, the pro
       gram could run anywhere.  The -jID-LIST option selects a random node from
       among the supplied list of nodes.  the list should be comma-separated and
       may contain any combination of either single MOSIX node numbers, or dash-
       separated range of MOSIX nodes.  Again, mosrun will fail should the se
       lected random node be unavailable (unless the -F option is specified).

       The node-lock, preventing automatic process-migrations (except in emer
       gencies, such as when a node is being shut-down, in which case the pro
       cesses will migrate to their home-node), can be controlled via one of the
       -l (unlock), -L (lock) or -k (keep) flags.
       The default is to lock whenever either -h, -MOSIX_ID or -j are specified
       or when no decay options are specified and to otherwise keep the status
       as per the calling shell/program.

:
```

↑The 'mosrun' utility is used to set the migration options on processes on the local host.

and ssh, if you are running them, can be migrated.

If you want in.telnetd to migrate, you need to have it run through mosrun -l -z and the /etc/inetd.conf entry would look something like:

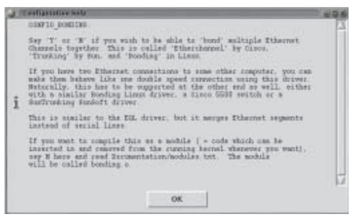
```
telnet stream tcp nowait root
/bin/mosrun mosrun -l -z
/usr/sbin/tcpd in.telnetd
```

/etc/xinetd.conf is slightly different, and /bin/mosrun has to be considered as a server, and everything else is an argument for it:

```
service telnet
{
  socket_type = stream
  protocol = tcp
  wait = no
  user = root
  server = /bin/mosrun
     server_args = -l in.telnetd
}
```

→Need more bandwidth?

Sometimes 100Mbit isn't enough, but Linux has a rather useful kernel module to help



```
CONFIG_BONDING

Say 'Y' or 'M' if you wish to be able to 'bond' multiple Ethernet
Channels together. This is called 'Etherchannel' by Cisco,
'Trunking' by Sun, and 'Bonding' in Linux.

If you have two Ethernet connections to some other computer, you can
make them behave like one double speed connection using this driver.
Naturally, this has to be supported at the other end as well, either
with a similar Bonding Linux driver, a Cisco 5500 switch or a
SunTrunking SunSoft driver.

This is similar to the EQL driver, but it merges Ethernet segments
instead of serial lines.

If you want to compile this as a module ( = code which can be
inserted in and removed from the running kernel whenever you want),
say M here and read Documentation/modules.txt.  The module
will be called bonding.o.
```

←The Kernel Ethernet bonding module enables you to combine two or more Ethernet interfaces into a single, faster, interface.

MOSIX is network intensive, so if you have a 10BaseT network you may want to invest in some new kit. But, even for some small clusters 100BaseT-FDx is too slow, and often the expense of upgrading to Gigabit hardware, be it fibre or copper, puts a lid on upgrade opportunities. Linux has a nice way of making use of 100BaseT hardware, or any Ethernet network interfaces to improve bandwidth. It only works between two machines, so daisy chaining is the easiest way to set up a network using it.

The Linux kernel has a bonded network device, where you can bond two or more interfaces into a single bond0 interface, and use that for transferring data. It requires the ifenslave utility, which is difficult to locate on the net, but the source code can be copied from **www.geocrawler.com/archives/3/423/2001/4/0/5704810**

Once you've built ifenslave, with gcc -o ifenslave ifenslave.c and put it in /usr/sbin, you can set up your bond0 interface:

```
ifenslave bond0 eth0
ifenslave bond0 eth1
ifconfig bond0 10.1.1.1
netmask 255.255.255.252 up
```

The machine on the other end will need to be set up in exactly the same way, but using 10.1.1.2 instead. There are a number of NICs available with more than one port, such as the quad port D-Link 570TX, so you don't need to fill your machines up with single port cards and waste slots.