Building a Mosix Cluster with SystemImager

By Rick Philbrick of Yozons, Inc.

Also be sure to see the first article in this series, "Image is Everything", in which we discussed using SystemImager to automate the install of nodes in a cluster.

Its a mixed blessing living here in Seattle. On the one hand, your best friends or even your spouse will have some affiliation with Microsoft. But on the other hand, another huge local company that builds 747s and satelites also sells used PCs for pennies on the dollar. What-a-deal!!!

I heard obout MOSIX and really wanted to try it out. Following my primal "Having to try it" urge also gave me the excuse to go to the Boeing surplus store to see if I could scrounge up a couple machines to give this an honest attempt. Much to my liking, the day I went they were dumping the PCs I came to scrounge. I got 8 of them for less then 200 bucks.

Each box has a classic non-MMX Pentium 133 w/64megs of RAM and a 1 gig hard drive. Not exactly the horsepower you'd want in a modern machine, but I was just out to prove the concept and these fit the buget. I also used some inexpensive tulip chipset NICs from my parts box.

As I was driving home, I was thinking, "Where are these going to fit? Will I have enough power? How long is it going to take to setup and install the software on these darn PCs. What am I going to tell my wife when she sees all these boxes? (We have a very modest sized house.)" As it turns out, getting the software setup and installed is the easiest task above all else, thanks to SystemImager.

Now for those of you in the dark about MOSIX please visit the MOSIX homepage. MOSIX is a Linux kernel extension that allows you to run normal (non-cluster aware) applications across a cluster. One feature of MOSIX that I found most intriguing is "process migration". Process migration is just what it sounds like -- processes can migrate from one node to another. For example, when a particular process starts to dominate a machine's load, it gets moved to another node in the cluster that has more idle resources. What you get is SMP-like functionalty out of a bunch of uni-processor machines.

I chose to experiment with MOSIX out of pure curiosity, and whenever I start out on one of these pursuits, I put a lot of time into reading up on the subject and visiting associated web sites. The MOSIX web site is a great resource for anyone who wants to learn about MOSIX or just needs to get some hints on setting it up. I also found some good downloads, a mailing list, and FAQs that seem to flow with you as you're setting up the software.

I took a simple approach to arranging the hardware which was to set up each machine to be as uniform as possible. I gave them matching NICs, CPUs, HardDrives, and memory. MOSIX will do just fine when using PCs of varying horsepower -- I just find it easier to manage a cluster of like machines. If there is any real trick here it is to have the Image Server ready for serving the image. I'll refer you to the first installment of this series to get hints on how to do that. Since I already had the image server set up, I only needed to get one of the nodes installed and configured as the "golden client", get it's image to the image server, and then create a SystemImager boot floppy.

For the preparation of the golden client I started with a base install of RH7.1. My nodes only have a single 1 gig drive, so with that limitation I am forced to conserve disk space. With the aide of a temporarily attached CD-ROM drive, I got the base RH 7.1 installed with minimal Xwindows for a GUI front end and I still have room to do a build of the MOSIX kernel. If you want to install MOSIX from an RPM you can choose to do that instead.

FYI: The MOSIX RPM I found is for kernel version 2.2.19 only. It worked fine for my first swing at this, but for this article I chose the version for the 2.4 kernel as it provides the added benefits of the newer 2.4 kernel and the support of MFS. If you haven't heard of MFS, I am not surprised. It is a file system unique to MOSIX. The advantage of MFS is that it will make all directories and regular files throughout a MOSIX cluster available from all nodes. It is compliant with Direct File System Access (DFSA) standards.

To build the MOSIX kernel I needed to download the kernel 2.4.13 sources from kernel.org and the MOSIX patch ver 1.5.1 from the MOSIX homepage. I placed them both in the /usr/src directory. MOSIX has an automatic build script and it worked cleanly for me. Make sure you have the dev86 and binutils packages installed if you're using RedHat. The MOSIX script will walk you through the kernel config and then do its compile and even edit lilo.conf for you. So far, not much time has been invested, perhaps about the same amount of time had I done one those "Install Everything" types of installs.

After I got the MOSIX kernel built and installed, I determined it was a good idea to edit a couple of files. One is the

cluster configuration file aka /etc/mosix.map and the other is the /etc/rc.local file.

First, /etc/mosix.map is a simple configuration file that is found on each of the nodes. Since mosix.map needs to be consistant throughout the nodes, and because we'll be using SystemImager, it will only have to be edited on one machine, the golden client.

On this small experimental cluster it looks like this.

```
# 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.
#
# MOSIX#  IP             Number-of-Nodes
# ====================================
  1       192.168.0.101  4
```

Next, because the cluster is made up of P133 machines it is necessary to have the cluster calibrated for that CPU at start up. This can be done simply by placing the "mosctl setyard P/133" command on its own line in rc.local. If your're not using a P133 or if your CPU has MMX you'll need to tweak mosctl similarly. There's a good man page for mosctl that will tell you the settings you need. Chips with MMX have a built in bias to report higher than actual speeds. MOSIX has a special way of compensating for this which you can read about in the mosctl manpage.

MOSIX comes with monitoring tools based on ncurses, but I decided to use a GUI based tool called Mosixview. It is a front end to the mosctl command. You'll need to make sure that the mosix.map file is correctly edited or you won't get the desired results from this application. Mosixview is certainly the way to go for an easy to follow visual cluster managemnet tool and is also a well supported Open Source program.

You need to have rsh or SSH set up on the nodes for Mosixview to work. If you want to be able to root to a remote machine without being prompted for a password, here are the steps to do it on a RH 7.1 box.

First, you need to have openssl, openssh-client, and openssh-server installed. As root, go to the /root/.ssh/ directory. I emptied mine, starting fresh. While you're at it, make sure sshd is running. To do that on RH7.1, run "/etc/init.d/sshd status". You should see something like this:

```
sshd (pid 1343 1295) is running
```

To generate key pairs, run the following command: "ssh-keygen -t rsa". I accepted the defaults and left the passphrase blank. So in essence, I just hit <Enter> three times. This is an isolated network, which I have sole access to, so I feel comfortable leaving the passphrase blank.

Now, run ssh-agent (with no arguments) from the /root/.ssh/ directory. Then check to see if your variables are exported. Borrowing from the Mosixview SSH HOWTO, do the following:

```
echo $SSH_AUTH_SOCK
echo $SSH_AGENT_PID
```

Mine were fine... as I had the login-profile tweaked from earlier attempts. At this point, in your /root/.ssh/ directory you should have the two files that were created by ssh-keygen: id_rsa and id_rsa.pub. So add a third, an authorized_keys2 file. Just copy id_rsa.pub to /root/.ssh/authorized_keys2

Now we need to run run ssh-add (with no arguments).

Restart the sshd deamon for good measure, and try logging in with SSH. You should get a response like this:

```
[root@imageserver]# ssh golden-client
The authenticity of host 'golden-client (192.168.0.101)' can't be established.
RSA1 key fingerprint is dd:17:c0:86:a0:62:33:f9:ad:68:30:b5:25:70:1e:f3.
Are you sure you want to continue connecting (yes/no)?
```

Type yes and it will remember your decision. Log in, log out, log in, logout, ...try it a couple times.

HINT: If you do this on the golden-client it will get installed on all the other nodes you build. This is a good thing.

Finally, we'll install the systemimager-client package on this fresh golden client and run the prepareclient command. Then we can go to the ImageServer and run the getimage command. By running those commands, the golden client's image gets saved to the image server. My host and DHCP configurations are already in place, so I'm ready to build out the remaining nodes. If you're looking for tips on how to set up the image server, or a HOWTO on SystemImager, check with my first article or go to the SystemImager documentation. Since the nodes get their names and IP addresses via DHCP from the image server, you should boot/bring up the nodes in the order you want them named. For example, the node that is serving as the golden-client is mosix1. The image server assigns IP addresses, and consequently host names, in order. This means that the first node that gets built from that image will be named mosix2, the next mosix3, etc.

In a short time I had my MOSIX cluster set up and running. With a mouse, monitor, and keyboard connected to my master node, I can start a job and monitor the process as it migrates. There is also a test program written in AWK that can be found on the MOSIX FAQ page. Copy that one liner down and run it. It will show you how processes migrate.

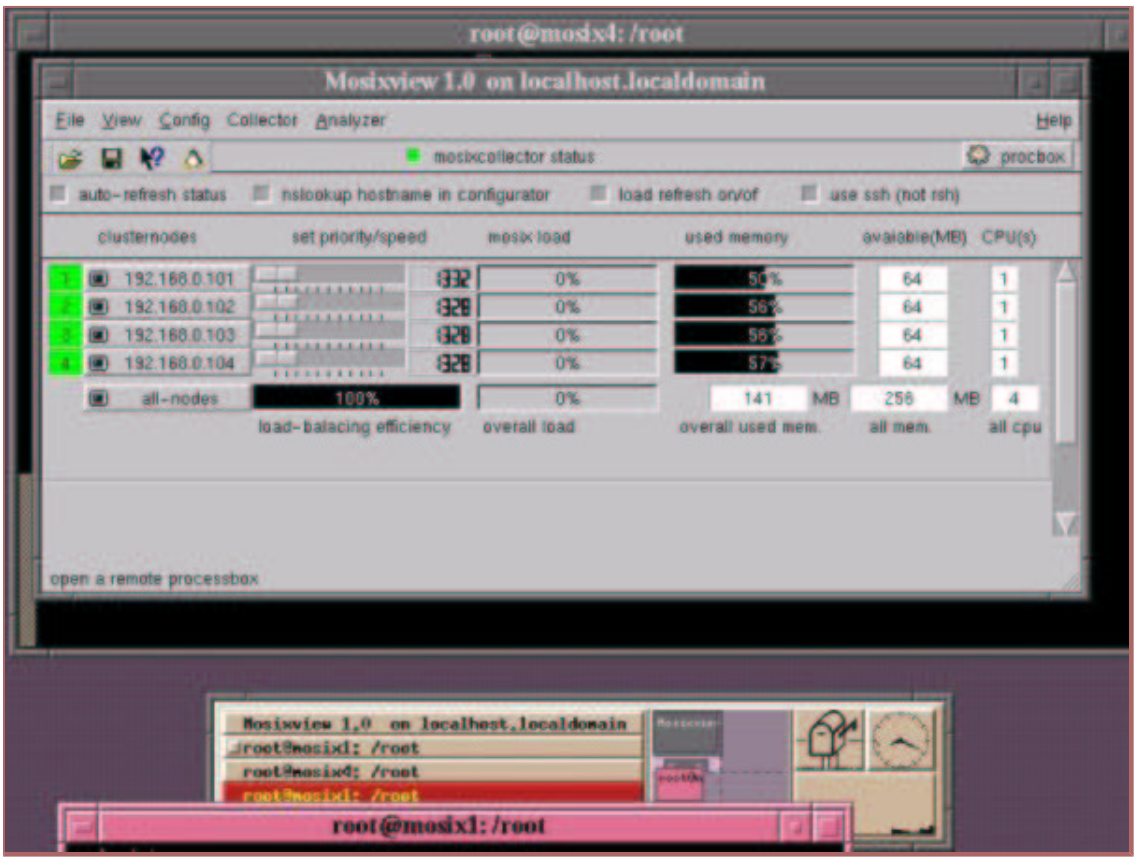
Here is an outline of the entire process:

- Install RH7.1 on what will be node1.
- Install the sources for the 2.4.13 kernel and the MOSIX patch 1.5.1 in /usr/src.
- Run the MOSIX autoinstall script to get the MOSIX kernel built and installed.
- Edit the cluster configuration file /etc/mosix.map.
- Edit the /etc/init.d/rc.local file to calibrate the cluster for the CPU model.
- Install Mosixview and setup SSH for remote root access without authentication prompts.
- On the golden client, install the systemimager-client package and run prepareclient.
- Run the getimage command from the image server.
- Boot a blank box with the SystemImager floppy media.

I leave the ImageServer running so the machines will always have their given DHCP identiy and for easily updating the nodes to new images. SystemImager is a real time saver when it comes to adding new files, or updates to existing files on the nodes. When it comes to upgrading the software on all the nodes, you'll find it much easier to use the updateclient feature than having to perform the task of manually upgrading each node. Now that we have this much going we can check in with the MOSIX homepage and look for any updates. The MOSIX team is very good about releasing early and often. Thanks guys!

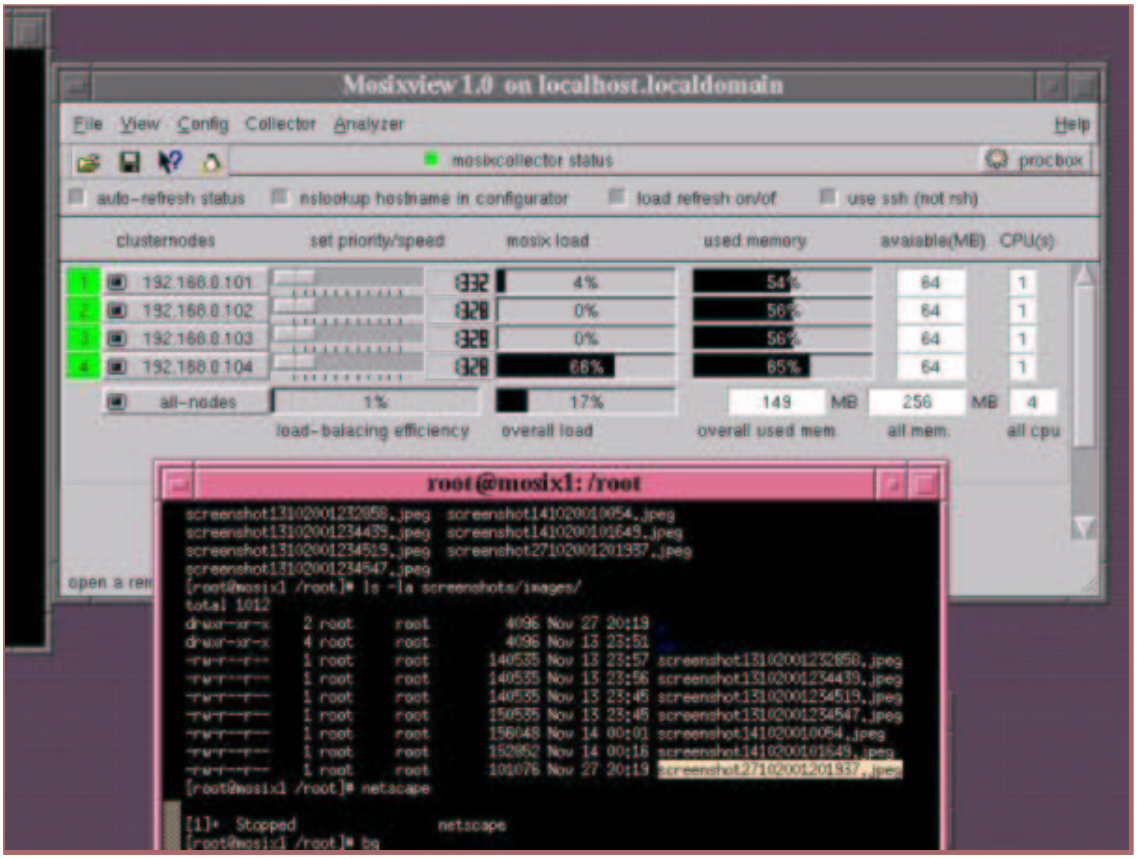
So, with all the machines running MOSIX and networked into a 100Mb ethernet switch, let's step through a quick test to find out how well process migration works. Because these machines are networked, and Mosixview is installed properly, we only need to attach the monitor, keyboard, and mouse to the one box we will use as the command station. I will start several CPU bound processes using the one line awk script that I found on the MOSIX FAQ page, and watch the processes migrate. If I want to manually move processes around I have that capability as well.

Login to the command station node (on my cluster that would be mosix1). Launch an X session and a couple xterms. Then, using an availabe xterm, start Mosixview. At the very bottom of the next slide you can barely see the xterm for mosix1, but its there.

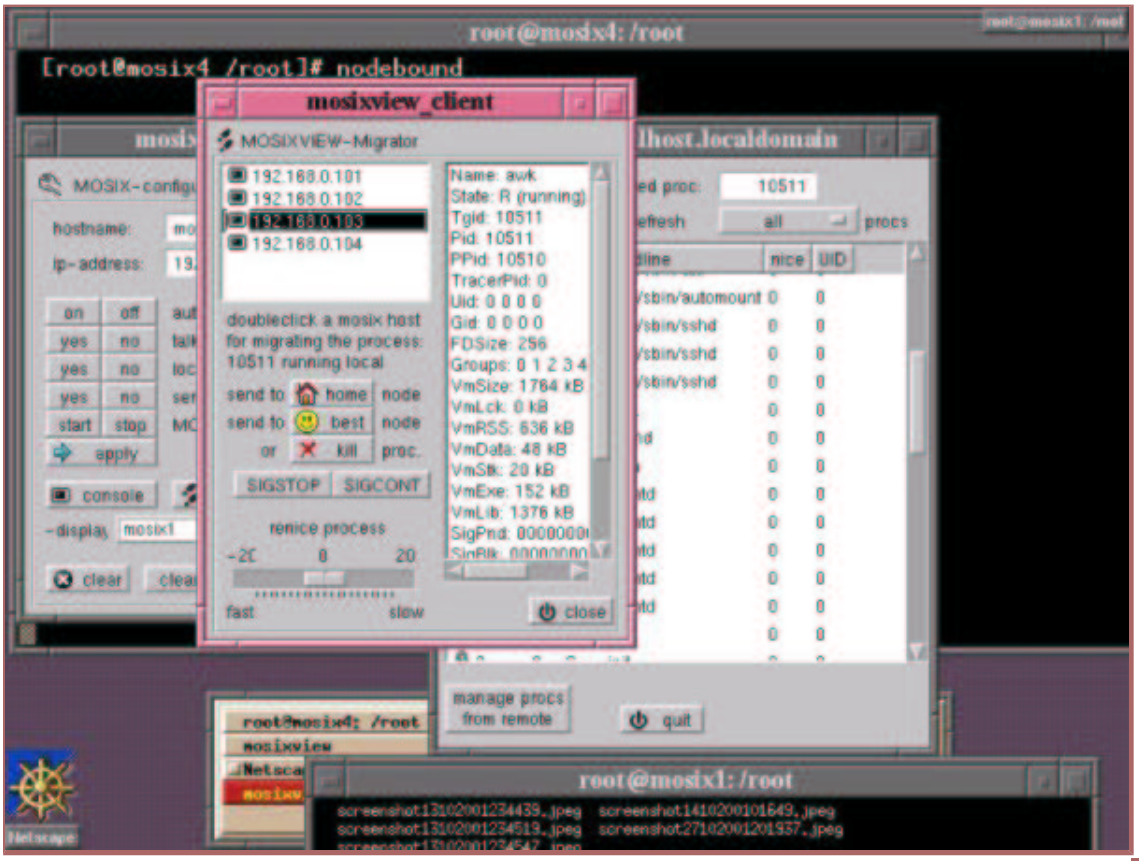


The images for this article were screenshots taken with a perl script called slapshot.

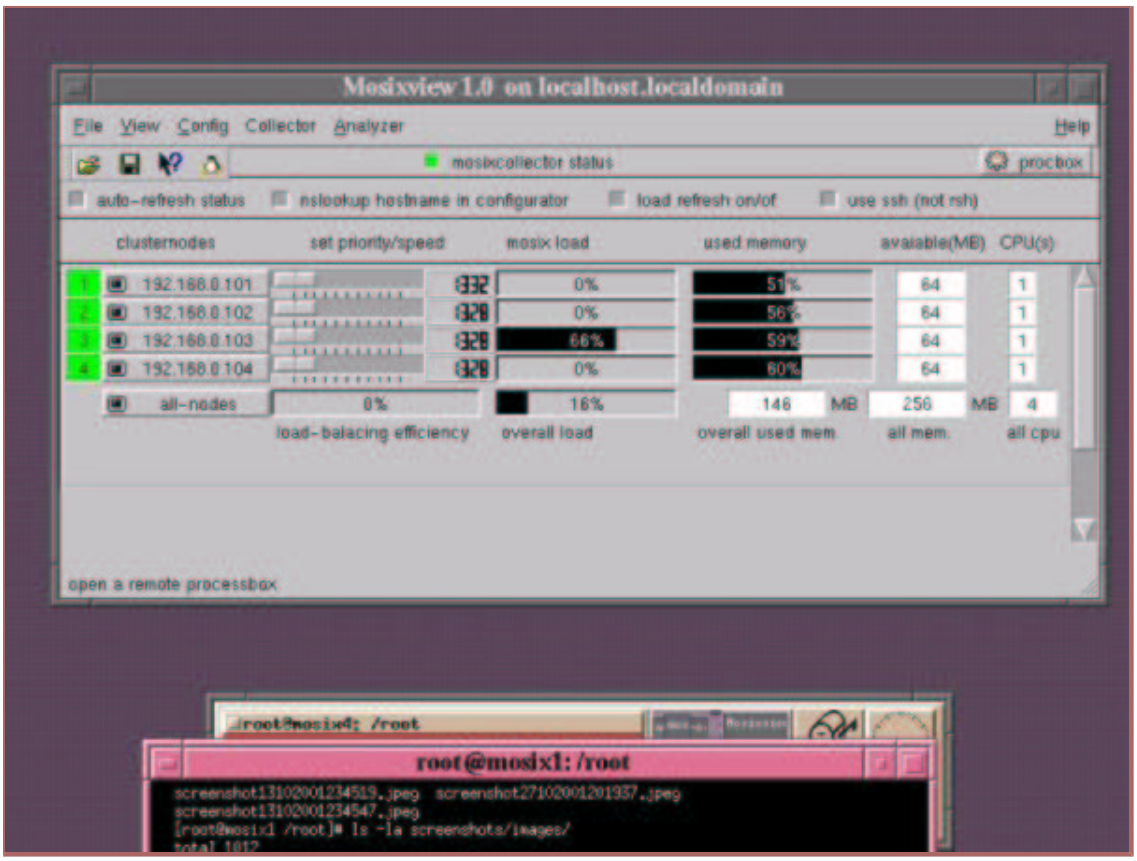
As you can see, each node's number is in green, indicating it is up. Next to the node's number is it's IP address. If the node was offline, its number would appear in red text to indicate non-participation. From the slide above you can see that there is no load on the cluster at all. Lets change that. Since I have an open ssh session to mosix4, I'll start pouring on the steam to that node. By running nodebounder , the awk script that I mentioned earlier, I begin to see the load on mosix4 increase steadily as illustrated below. What you should see, is that the load on mosix4 will top out and the process will go to mosix3 and then to mosix2 and then to mosix1, Ceteris Paribus. But in our example we make mosix1 more equal than the others, so it does not see any topping out or visits from migrating processes. Nice work if you can find it, that is ... to be mosix1.



An alternative way of dealing with processes is to manually assign them to a node. Say that job we started on mosix4 should be placed on mosix3. We can use the mosixview_client to list the processes on any node. Here we need to move process 10511 from mosix4 to mosix3. Double click the process and you'll get the Mosixview-Migrator. Then just double click on the node you want to send PID 10511. It's that easy!!!



Here is what the main Mosixview window looks like after PID 10511 was migrated to mosix3.



The combination of SystemImager and MOSIX is very compelling. By combining the two, anyone can automate the installation of an automated cluster.

The availablity of quality Open Source software such as Mosixview is the perfect icing on this two layered cake.

Acknowledgments: Special thanks Brian Elliott Finley for writing SystemImager and to the MOSIX team for writing MOSIX.

Rick Philbrick (rickphilbrick@hotmail.com) is a Sr. Systems Engineer with Yozons, Inc. and lives in Bellevue Washington. He has been working with Linux since 1996, being introduced to it while on the IT staff at E*Trade.