

Project Survey

Distributive Memory benchmarking with openMosix Clustering

Prepared for: Professor Kaugars, CS 454, Spring 2006

Prepared by: Lucas Holt & Swaroop Atre

February 7, 2006

Distributive Memory benchmarking with openMosix Clustering



An introduction to openMosix

openMosix is a project based on work done in Israel to allow linux processes to be distributed through a cluster environment, but in a manner that simulates SMP process execution. It uses kernel patches and user-land utilities to control the cluster and provide the scheduling and monitoring of nodes. The focus is exclusively on giving processes 100% CPU utilization on a host. Without patches, the system can not run shared memory applications, networked applications like the Apache HTTPD server, or multithreaded applications. Ideal workloads are those which use a series of forked processors or when a series of processes needs to be executed like encoding mp3's in a batch.

The openMosix website is located at <http://openmosix.sourceforge.net>. A brief review of the shared memory extensions for the Linux kernel and openMosix can be found at <http://www.unixreview.com/documents/s=8989/ur0404/>.

From the openMosix website:

openMosix is a Linux kernel extension for single-system image clustering. This kernel extension turns a network of ordinary computers into a supercomputer for Linux applications.

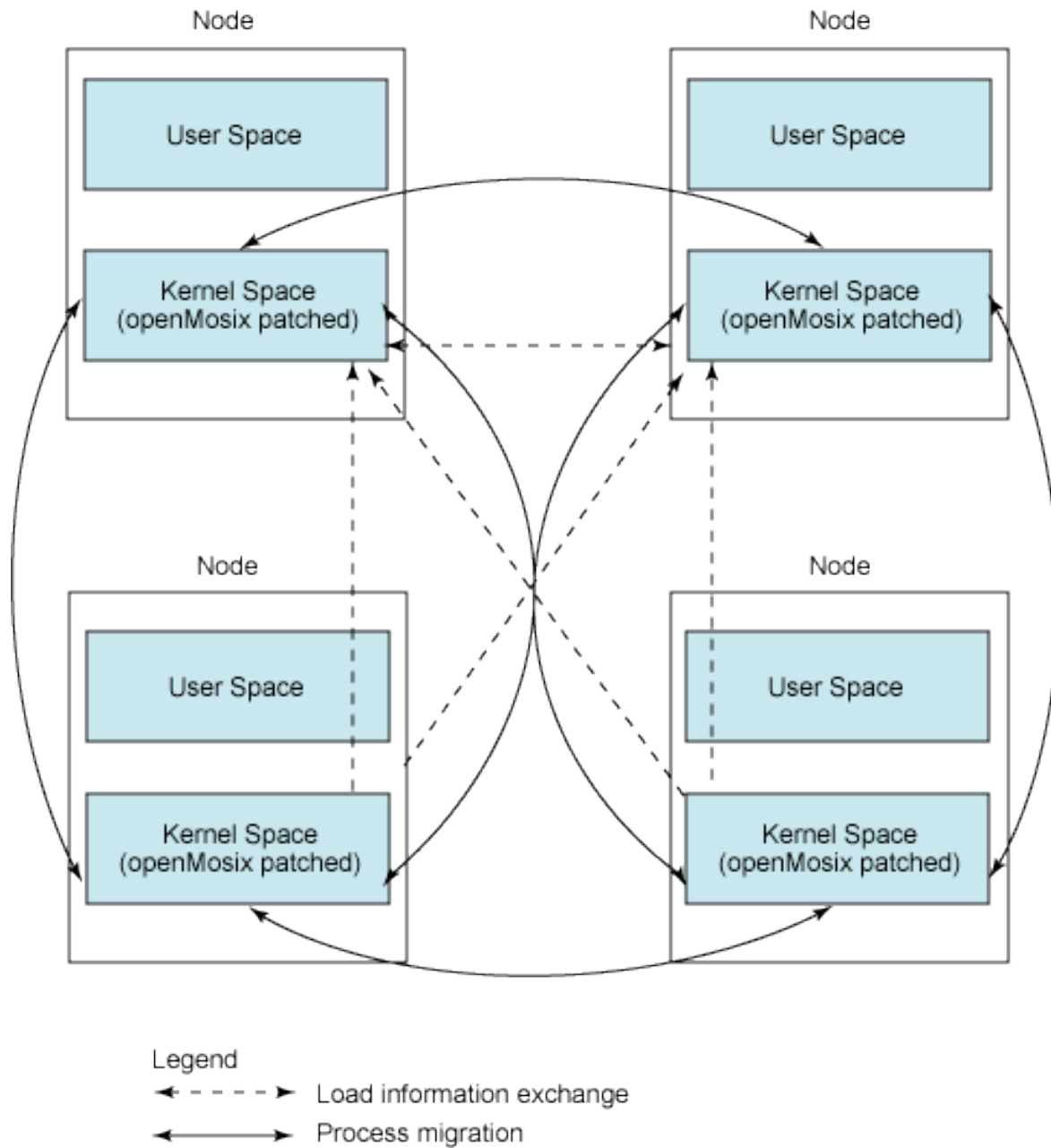
Once you have installed openMosix, the nodes in the cluster start talking to one another and the cluster adapts itself to the workload. Processes originating from any one node, if that node is too busy compared to others, can migrate to any other node. openMosix continuously attempts to optimize the resource allocation.

We achieve this with a kernel patch for Linux, creating a reliable, fast and cost-efficient SSI clustering platform that is linearly scalable and adaptive. With openMosix' Auto Discovery, a new node can be added while the cluster is running and the cluster will automatically begin to use the new resources.

There is no need to program applications specifically for openMosix. Since all openMosix extensions are inside the kernel, every Linux application automatically and transparently benefits from the distributed computing concept of openMosix. The cluster behaves much as does a Symmetric Multi-Processor, but this solution scales to well over a thousand nodes which can themselves be SMPs.

The openMosix Community is very active, contributing add-on applications and sharing helpful information with all users. The openMosix Add-Ons and Community page lists these shared applications. And, it is all GPL'd.

A visual representation of openMosix in action



As you can see, process migration happens at the kernel level utilizing the patches, just a conventional scheduling algorithm would run in the kernel.

Source; IBM

<http://www-128.ibm.com/developerworks/linux/library/l-colinux/>

Benchmarking openMosix

While several benchmarks have been performed on openMosix in specific situations, the general benefits of openMosix to standard computing, and conventional networks have not been expressed. Most benchmarks in existence use a specific test like compiling the linux kernel with -j 4 flag or doing highly computational work on weather patterns. The openMosix wiki FAQ only specifies doing vanilla linux kernel builds, but also points out that a 2.0 ghz system with 1gb of ram can build the kernel so quickly that it is difficult to put enough load to justify openMosix.

<http://howto.x-tend.be/openMosixWiki/index.php/Additions%20to%20the%20FAQ> (see benchmark question)

Another benchmarking test spawns a series of tests that are CPU bound and model work patterns for scientific loads. This test does not force processes on other machines, but actually tests the algorithms to move processes automatically across machines. <http://www.xyz-network.com/ombenchmark>

Benchmarks have also been performed on thread implementations. According to official documentation, threading is not supported by the openMosix patches as processes can not be migrated without the shared memory patches taking into account for threading libraries. Several libraries were tested which use user-land pthread style implementations. Also linux threads were benchmarked. http://filibusta.crema.unimi.it/openmosix/fsu_threads_on_om/benchmark.htm

I did not find user oriented benchmarks or benchmarks targeted to popular server scenarios. Our benchmarking will focus on these two areas. We hope to determine if desktop processes can be migrated to other hosts such as firefox, KDE or gnome applications, etc. While we can not benchmark specific performance, it would serve as a nice definition to openMosix boundaries.

Benchmarking audio and video encoding applications running concurrently should allow us to test how well openMosix performs. Multiple instances of lame encoding large mp3 files, or compressing MPEG video streams will allow us to test the CPU load and how openMosix handles IO boundaries.

We also plan to benchmark a few custom written c applications that do number crunching both to test the cluster and get a pure CPU bound test. Another option being explored is testing memcached, which is a c based object caching system based on hashing. It is used by sites like livejournal.com to cache user data to avoid MySQL lookups. Since it is entirely memory and CPU bound, we feel this might test the distributed shared memory model.

Why is openMosix useful?

openMosix was developed to save money on SMP hardware and to make use of older machines lying around. It allows for a cheap cluster to be built, while maintaining the simplicity of a traditional SMP based machine. The limitations stated on some sites include the lack of sockets support, general issues with IO operations across hosts, and lack of threading support. These limitations would make it useless for many tasks, however quite a few benchmarks show threading and many IO operations working correctly. If it is possible to overcome these issues, openMosix could increase the performance of home computing environments for power users who could run several tasks simultaneously, system administrators setting up business logic tiers in web farms, or the obvious research environments. Simulations could be run for engineers, computer scientists, DNA testing for bio research and the list goes on. It would not be useful with web-servers which are always IO bound both in network traffic and file IO. Web applications can be separated into logical areas including the data tier, business tier and presentation tier. It might be possible to get the data and business tiers to take advantage of openMosix. Stored procedures often run on a thread pool and the business tier includes all the processing necessary to enforce policies and format data for output. That is often a cpu bound operation. We hope to create tests to benchmark and prove the usefulness of openMosix in many of these areas.

Tasks

- Setup a host with Linux and install an openMosix kernel. Verify the host is stable and functional.
- Setup and configure additional linux hosts with a similar configuration and identical kernel version.
- Create a series of tests we can run to prove the functionality of the cluster.
- Run the tests on the cluster and compare them to the results running individually without openMosix distributing the load. Also compare to an SMP machine without openMosix active (but still in the kernel) as a baseline.
- Formalize a series of benchmarks and determine the environmental needs and attempt to create a consistent environment for the tests. Run the tests as many times as possible in the time available.
- Draw conclusions from the results and present them.

Progress

1. A Linux machine was setup with the latest Gentoo 2005.1 release. Portage includes the openMosix kernel patches and source to the 2.4.30 kernel. The 2.6.x release is masked do to lack of user-land tools for it.
2. The openMosix kernel was installed and booted on the host.
3. Currently scrounging for more hardware to do a cluster. At least two more machines are available.
4. Currently researching possible tests we can run on the cluster. The lame encoder will be included in testing.
5. Expect to have at least 3 computers simultaneously setup for openMosix testing, benchmarks run individually and together on the hosts by the project deadline. Ideally would like to run at least 10 different tests to verify results focusing on memory and CPU intensive operations. The results should prove interesting both in terms of scheduling and SMP operation.