Adventures in Cheap (i.e. Student) Clustering

Alapan Arnab & Carl Hultquist

Overview

- Introduction
- Set Up
- Mosix vs. Open Mosix
- Dist CC

Introduction

- Tutorial as part of Honours’ Cluster Computing Course
- Group of 7 students; 4 groups in total
  - Set up
  - Software
  - Benchmarking
- Group decided to build and test a variety of cluster systems
  - MPI
    - LAM-MPI vs MPICH
  - Open Mosix
  - Mosix

Equipment

- Borrowed machines from the Science Faculty
- 4 machines per group
  - Pentium III 550 MHz
  - 10/100 Mbps Network Card
  - 6 Gb Hard disks
  - USB
  - 1 x 3.5” Floppy and 1 x CD-Rom drives
- 16 Port Switch (shared between all the groups)
- No Internet connection
- Debian GNU/Linux 3.0r1 (woody)

Set Up

- Traditional network of computers
- Master Node + 3 slave nodes
- Master Node
  - DHCP, DNS
  - SystemImager
  - NFS
  - NTP

SystemImager

- Cloning nodes
- Created images of desired systems and stored on Master
  - ○ Images essentially the full directory structure
  - ○ Requires a client node to be imaged
  - ○ Simpler, Better, Faster than individual installations
- Uses rsync to update
  - ○ Only updates data that has changed
  - ○ New nodes can be imaged through boot disk/CD from the master node

SystemImager (continued)

- Uses SSH and “sudo” for security
  - Unnecessary in our environment
  - Modified scripts to increase speed
- One bug in *pushupdate*:
  - Command has a range argument to update multiple machines
  - However, script did not save the parameters, thus halted after the first update
- Incomplete documentation for SystemImager

MPICH vs LAM-MPI

- Intention was to install and benchmark both
- Both can co-exist on same system; but we also had images with separate copies
- MPICH does not *fully* support MPI 2.0 specs
  - Our MPI Benchmarks were all MPI 2.0 specs
    - PMB-MPI 1
    - PMB-EXT
  - Did not carry on using MPICH

MOSIX and OpenMOSIX

- Automatic process migration across nodes in the cluster
  - Nodes periodically notify other nodes of their status regarding number of processes, processor usage and memory usage
  - When a node determines that another node has more resources to handle a process, the process is *migrated*, and then runs on the other node

Migrated processes: the details

- As far as process is concerned, it is still running on the originating node
  - Includes any kind of I/O access, memory access, network access, and process control functions
  - All of these operating system calls are relayed back to the originating node, which executes the relevant function and sends any output back to the other node

A user’s view of a MOSIX/OpenMOSIX system

- When a process is migrated, the destination node does not reflect any information about this process
- Furthermore, the information about the process is still maintained on the originating node
  - Complete transparency: user’s on the originating node still perceive the process as running locally, whilst users on the destination node are unaware that the process is running on that node

MOSIX/OpenMOSIX components

- Two components: a kernel patch, and user-space tools
    - For both MOSIX/OpenMOSIX, the kernel patch is GPL licensed
    - For MOSIX, the user-space tools are not GPL’ed
    - OpenMOSIX user-space tools are GPL’ed
- Kernel patch handles details related to maintaining user transparency and redirection of specific OS function calls
- User-space tools handle communication between nodes; these tools then interface with the local kernel functions exposed by the kernel patch

Issues with MOSIX/OpenMOSIX

- Processes that do lots of I/O run slowly, since all I/O needs to be communicated over the network between the destination and originating nodes
  - ○ Partially solved using DFSA (Direct File System Access) and MFS (MOSIX File System)
    - MFS allows each node to see the entire directory structure of every node in the cluster, promoting data sharing which can also be cached by MOSIX
- In many scientific applications, many parallel processes need to access the same large datafile. Doing this over the network is very infeasible!
  - ○ Solution: MOPI (MOSIX Parallel I/O). Instead of “bringing the file to the process”, MOPI suggests “bringing the process to the file”. The datafile is split over the nodes in the cluster, and when a process needs to access a certain portion of the file it is migrated to the appropriate node.

How do MOSIX and OpenMOSIX differ?

- Essentially, MOSIX has a restrictive license on the user-space tools whilst OpenMOSIX is completely GPL’ed.
- Conducted a test to really try and see which was “better”
  - Basic idea behind test was to create a setup which would intentionally try to “break” MOSIX/OpenMOSIX, and see which handled this test best. Tests that stress the *benefits* of MOSIX/OpenMOSIX weren’t conducted.

The “Carl-test”

- Create several processes that perform mind-numbing calculations (additions, multiplications, etc)
- Insert random delays for random periods into each process
  - The delay causes the load on the executing machine to decrease, hopefully “tricking” MOSIX/OpenMOSIX into thinking that it could migrate a more needy process to that node

Final details of the “Carl-test”

- Program written in C that:
  - Creates 10 child processes
  - For each process:
    - Seeds the random number generator (where the seeds are different for each process, but the same on consecutive runs on the program – i.e. program is deterministic)
    - Executes an outer loop of 1000000 cycles, which:
      - Executes an inner loop of 1000 mathematical operations
      - Randomly decides whether the process should be delayed; if so the process is delayed for a random number of seconds between 1 and 10
  - Waits for all its children to finish their processing

Results of the “Carl-test”

Configuration	Time taken to complete test
Single CPU (no MOSIX, no OpenMOSIX)	3m 1.052s
MOSIX (4-node cluster)	2m 30.254s
OpenMOSIX (4-node cluster)	3m 4.054s

MOSIX/OpenMOSIX conclusions

- Useful to abstract load balancing
- Good for ad-hoc clusters that simply runs lots of unrelated jobs
- Bad for programmatic load balancing, since the user has no control over how load balancing is achieved
- AFAIK, requires a 2.4 series kernel! IMHO 2.6 offers some good performance enhancements, and a 2.6 MOSIX/OpenMOSIX patch would be useful

Recent adventuring: DistCC

- Compiling large projects can take *forever*
- Would be useful to parallelise compilation, balancing a batch of compilations over the cluster
  - Could simply spawn lots of compilation processes on a MOSIX/OpenMOSIX system and let it’s implicit load balancing handle things
    - However, compilations are typically quick enough such that MOSIX/OpenMOSIX does not have a chance to do any balancing
  - Better bet is to directly distribute compilations to specific nodes

DistCC specifics

- Simply used by using the “-j <x>” switch with make (<x> specifies number of processes to run), and then calling “distcc <compiler>” to do the compilation
- DistCC then controls which node the compilation should go to, according to how many other compilation jobs the node is running
- All preprocessing of the source is done locally, and the resulting source is then compiled on the destination node. All linking is done locally.

How we’re using DistCC

- Originally started with 2 Gentoo and 2 Debian installations. Just needed to ensure that the same major compiler versions (GCC 3.3) were being used, and worked flawlessly. Mostly used for compiling MSc projects, and by Gentoo boxes for compiling packages from source. Also great for compiling the Linux kernel!
- Now consists of 5 Gentoo installations, which has made this even more invaluable for package compilations ☺

Issues related to DistCC

- Other compilers aside from GCC?
  - Should in theory work with any compiler that supports the same compiler switches as GCC (esp. -E for preprocessing, and switches such as -I, -c, -L and -l that control path settings and linking behaviour).
- If multiple machines do a concurrent distributed compile, then these machines are not aware of each other’s compilations. Can cause machines to become overloaded processing compile jobs on behalf of two or more others! In general, doesn’t happen too often.
- Could be very useful on a cluster with heavy user load where the users perform all actions on one server: this server can then distribute out the compilations to slave nodes to achieve a balance.

Questions and suggestions?