

COCOA FAQ

Anirudh Modi <anirudh-modi@psu.edu>

v1.1, 1st February 1999

This is the FAQ (Frequently Asked Questions) for COCOA, the inexpensive Beowulf supercomputer of the Aerospace Department at Pennsylvania State University.

1 Introduction

What is COCOA?

COCOA stands for **CO**st effective **CO**mputing **A**rray. It is a Beowulf class supercomputer. Beowulf is a multi computer architecture which can be used for parallel computations. It is a system which usually consists of one server node, and one or more client nodes connected together via Ethernet or some other fast network. It is a system built using commodity hardware components, like any office desktop PC with standard Ethernet adapters, and switches. It does not contain any custom hardware components and is trivially reproducible. <<http://cocoa.ihpca.psu.edu/>>.

What hardware was used to build COCOA?

26 WS-410 workstations from Dell <<http://www.dell.com>>, each consisting of:

1. Dual 400 MHz Intel Pentium II Processors w/512K L2 cache
2. 512 MB SDRAM
3. 4 GB UW-SCSI2 Disk
4. 3COM 3c509B Fast Ethernet adapter (100 Mbits/sec)
5. 32x SCSI CD-ROM drive
6. 1.44 MB floppy drive
7. Cables

In addition, the following were also used:

1. One Baynetworks 450T 24-way 100 Mbits/sec switch
2. Two 12-way Monitor/keyboard/mouse switches
3. Four 500 kVa Uninterruptible Power Supplies from APC.
4. One monitor, keyboard, mouse and 54 GB of extra UW-SCSI2 hard disk space for one PC which was used as the server.

What is the operating system on COCOA?

Linux! In specific, RedHat Linux 5.1 distribution <<http://www.redhat.com>>.

Linux is a free version of the Unix operating system, and it runs on all PC/i386 compatible computers (and now also on PowerPCs, Alphas, Sparcs, Mips, Ataris, and Amigas). The Linux kernel is written by Linus Torvalds <torvalds@transmeta.com> and other volunteers. Most of the programs running under Linux are generic Unix freeware, many of them from the GNU project.

What software is installed on COCOA?

On the server, the following software is installed:

1. Base packages from RedHat Linux 5.1 distribution <<http://www.redhat.com>>
2. Freeware GNU C/C++ compiler as well as Pentium optimized GNU C/C++ compiler (*gcc*, *pgcc*)
3. Fortran 77/90 compiler and debugger by Portland Group
4. Freeware Message Passing Interface (MPI) libraries for parallel programming in C/C++/Fortran 77/Fortran 90.
5. Scientific Visualization Software TECPLOT from Amtec Corporation <<http://www.amtec.com>>

How much did COCOA cost?

Approximately *\$100,000!*

2 Details on how COCOA was built

2.1 Setting up the hardware

Setting up the hardware was fairly straight-forward. Here are the main steps:

1. Unpacked the machines, mounted them on the rack and numbered them.
2. Set up the 24-port network switch and connected one of the 100 Mbit ports to the second ethernet adapter of the server which was meant for the private network. The rest of the 23 ports were connected to the ethernet adapters of the clients. Then an expansion card with 2 additional ports was added on the switch to connect the remaining 2 clients.
3. Stacked the two 16-way monitor/keyboard switches and connected the video-out and the keyboard cables of each of the 25 machines and the server to it. A single monitor and keyboard were then hooked to the switch which controlled the entire cluster.
4. Connected the power cords to the four UPS.

2.2 Setting up the software

Well, this is where the real effort came in! Here are the main steps:

1. The server was the first to be set up. RedHat Linux 5.1 was installed on it using the bundled CD-ROM. Most of the hardware was automatically detected (including the network card), so the main focus was on partitioning the drive and choosing the relevant packages to be installed. A 3 GB growable root partition was created for the system files and the packages to be installed. Two 128 MB swap partitions were also created and the rest of the space (50 GB) was used for various user partitions. It was later realised that a separate /tmp partition of about 1 GB was a good idea.
2. The latest stable Linux kernel (then #2.0.36) was downloaded and compiled with SMP support using the Pentium GNU CC compiler *pgcc* <<http://www.goof.com/pcg/>> (which generates highly optimised code specifically for the Pentium II chipset) with only the relevant options required for

the available hardware. The following optimisation options were used: `pgcc -mpentiumpro -06 -fno-inline-functions`. Turning on SMP support was just a matter of clicking on a button in the *Processor type and features* menu of the kernel configurator (started by running `make xconfig`).

3. The new kernel-space NFS server for linux (*knfsd*) <<http://www.csua.berkeley.edu/~gam3/knfsd/>> was installed to replace the earlier user-space NFS server to obtain improved NFS performance. For quick and hassle-free installation, a RedHat RPM package was obtained from <<http://rufus.w3.org/linux/RPM/>>, a popular RPM repository. The default options were used.
4. `ssh` was downloaded from <<http://www.cs.hut.fi/ssh/>>, compiled and installed for secure access from the outside world. `ssh-1.2.26` was preferred over the newer `ssh-2.0.11` as `ssh v2.x` was much slower as well as backward incompatible. `sshd` daemon was started in runlevel 3 under `/etc/rc.d/rc3.d`. Recently, RedHat RPMs for `ssh` have started appearing in <<http://rufus.w3.org/linux/RPM/>> and several other RPM repositories, which make it much easier to install.
5. Both the 3c905B ethernet adapters were then configured; one that connected to the outside world (`eth1`) with the real IP address 128.118.170.11, and the other which connected to the private network (`eth0`) using a dummy IP address 10.0.0.1. Latest drivers for the 3COM 3c905B adapters written by Donald Becker (`3c59x.c v0.99H` <<http://cesdis.gsfc.nasa.gov/linux/drivers/vortex.html>>) were compiled into the kernel to ensure 100 Mbit/sec Full-duplex connectivity. This was checked using the `vortex-diag` utility <<http://cesdis.gsfc.nasa.gov/linux/diag/vortex-diag.c>>. For the configuration, the following files were modified: `/etc/sysconfig/network`, `/etc/sysconfig/network-scripts/ifcfg-eth0` and `/etc/sysconfig/network-scripts/ifcfg-eth1`. Here is how they looked for me after modification:
`/etc/sysconfig/network`:

```
NETWORKING=yes
FORWARD_IPV4=no
HOSTNAME=cocoa.ihpca.psu.edu
DOMAINNAME=ihpca.psu.edu
GATEWAY=128.118.170.1
GATEWAYDEV=eth1
NISDOMAIN=ihpca.psu.edu
```

`/etc/sysconfig/network-scripts/ifcfg-eth0`:

```
DEVICE=eth0
IPADDR=10.0.0.1
NETMASK=255.255.255.0
NETWORK=10.0.0.0
BROADCAST=10.0.0.255
ONBOOT=yes
BOOTPROTO=none
USERCTL=no
```

`/etc/sysconfig/network-scripts/ifcfg-eth1`:

```
DEVICE=eth1
IPADDR=128.118.170.11
NETMASK=255.255.255.0
NETWORK=128.118.170.0
BROADCAST=128.118.170.255
```

```
ONBOOT=yes
BOOTPROTO=none
USERCTL=no
```

6. For easy and automated install, I decided to boot each of the PCs from the network using the BOOT protocol. The BOOTP server was enabled by uncommenting the following line in `/etc/inetd.conf`:

```
bootps dgram udp wait root /usr/sbin/tcpd bootpd
```

A linux boot floppy was prepared with the kernel support for 3c905B network adapter which was used to boot each of the client nodes to note down their unique 96-bit network hardware address (eg. 00C04F6BC052). Using these address, the `/etc/bootptab` was edited to look like:

```
.default:\
    :hd=/boot:bf=install.ks:\
    :vm=auto:\
    :dn=hpc.ihpca.psu.edu:\
    :gw=10.0.0.1:\
    :rp=/boot/client/root:

node1:ht=ethernet:ha=00C04F6BC0B8:ip=10.0.0.2:tc=.default
node2:ht=ethernet:ha=00C04F79AD76:ip=10.0.0.3:tc=.default
node3:ht=ethernet:ha=00C04F79B5DC:ip=10.0.0.4:tc=.default
.
.
.
node25:ht=ethernet:ha=00C04F79B30E:ip=10.0.0.26:tc=.default
```

7. The `/etc/hosts` file was edited to look like:

```
127.0.0.1    localhost    localhost.localdomain
# Server [COCO]
128.118.170.11 cocoa.ihpca.psu.edu cocoa.aero.psu.edu cocoa

# IP address <--> NAME mappings for the individual nodes of the cluster
10.0.0.1    node0.hpc.ihpca.psu.edu node0    # Server itself!
10.0.0.2    node1.hpc.ihpca.psu.edu node1
10.0.0.3    node2.hpc.ihpca.psu.edu node2
.
.
.
10.0.0.26    node25.hpc.ihpca.psu.edu node25
```

The `/etc/host.conf` was modified to contain the line:

```
order hosts,bind
```

This was to force the lookup of the IP address in the `/etc/hosts` file before requesting information from the DNS server.

8. The filesystems to be exported were added to `/etc/exports` file which looked like:

```

/boot          node*.hpc.ihpca.psu.edu (ro,link_absolute)
/mnt/cdrom     node*.hpc.ihpca.psu.edu (ro,link_absolute)
/usr/local     node*.hpc.ihpca.psu.edu (rw,no_all_squash,no_root_squash)
/home1        node*.hpc.ihpca.psu.edu (rw,no_all_squash,no_root_squash)
/home2        node*.hpc.ihpca.psu.edu (rw,no_all_squash,no_root_squash)
/home3        node*.hpc.ihpca.psu.edu (rw,no_all_squash,no_root_squash)
/home4        node*.hpc.ihpca.psu.edu (rw,no_all_squash,no_root_squash)

```

9. For rapid, uniform and unattended installation on each of the client nodes, RedHat 5.1 KickStart installation was ideal. Here is how my kickstart file `/boot/install.ks` looked like:

```

lang en
network --bootproto bootp
nfs --server 10.0.0.1 --dir /mnt/cdrom
keyboard us
zerombr yes
clearpart --all
part / --size 1600
part /local --size 2048
part /tmp --size 400 --grow
part swap --size 127
install
mouse ps/2
timezone --utc US/Eastern
rootpw --iscrypted kQvti0Ysw4r1c
lilo --append "mem=512M" --location mbr
%packages
@ Networked Workstation
%post
rpm -i ftp://10.0.0.1/pub/CLUSTER/RPMS/wget-1.5.0-2.i386.rpm
rpm -i ftp://10.0.0.1/pub/CLUSTER/RPMS/xntp3-5.93-2.i386.rpm
/usr/bin/wget ftp://10.0.0.1/pub/CLUSTER/kernel/vmlinuz -O/boot/vmlinuz
/usr/bin/wget ftp://10.0.0.1/pub/CLUSTER/conf/lilo.conf -O/etc/lilo.conf
/sbin/lilo
/usr/bin/wget ftp://10.0.0.1/pub/CLUSTER/conf/hosts.equiv -O/etc/hosts.equiv
sed "s/required\(.securetty\)\/optional\1/g" /etc/pam.d/rlogin > /tmp/rlogin
mv /tmp/rlogin /etc/pam.d/rlogin

```

For more info on RedHat KickStart installation, look at: <http://wwwcache.ja.net/dev/kickstart/KickStart-HOWTO.html>. In one of the post installation commands above, the first line of the `/etc/pam.d/rlogin` file is modified to contain:

```
auth optional /lib/security/pam_securetty.so
```

This is to required enable `rlogin/rsh` access from the server to the client without password which is very useful for the software maintenance of the clients. Also, the `/etc/hosts.equiv` file mentioned above looks like this:

```

node0
node1
node2
node3

```

```

.
.
.
node25

```

The RedHat Linux 5.1 CD-ROM was then mounted as `/mnt/cdrom` on the server which was NFS exported to the client nodes. A new kernel with SMP support was compiled for the client nodes in very much the same way as for the server and was used to replace the existing kernel in the RedHat book diskette. This kernel however had lesser options compiled in as it was only meant to act as a client. Additionally, option for “kernel level autoconfiguration using BOOTP” was enabled in the *Networking options* menu of the kernel configurator. This was required in order for the node to automatically get its IP address from the server at boot time. Support for The configuration file of the boot diskette was modified so as to boot directly in the KickStart mode. All that was needed to configure each client now was to insert the boot diskette, power-up the workstation and wait until the automatic installation was completed. Simple, eh ?!

10. As soon as all the clients were rebooted after installation, the cluster was up and running! Some useful utilities like `brsh` (<http://www.beowulf.org/software/RPMS/beobase-2.0-1.i386.rpm>) were installed to enable `rsh` a single identical command to each of the client nodes. This was then used to make any fine changes to the installation. NIS could have been installed to manage the user logins on every client node, but instead a simple shell script was written to distribute a common `/etc/passwd`, `/etc/shadow` and `/etc/group` file from the server.
11. Most of the services were disabled in `/etc/inetd.conf` for each of the client nodes as they were unnecessary. The stripped down `/etc/inetd.conf` for the client nodes finally looked like:

```

shell  stream  tcp      nowait  root    /usr/sbin/tcpd  in.rshd
auth   stream  tcp      nowait  nobody  /usr/sbin/in.identd in.identd -l -e -o

```

12. *automount* package was installed on each of the nodes to automatically mount the various user partitions on demand. Although this gave slightly improved NFS performance, it was found to be buggy and unstable. Finally, it was decided that *automount* for Linux was not yet ready for prime-time and was removed in favor of conventional NFS mounts.
13. The Portland Group Fortran 77/90 and HPF compilers (commercial) were then installed on the server.
14. Source code for freeware implementation of MPI library, MPI-CH was downloaded from <http://www.mcs.anl.gov/mpi/> and compiled using `pgcc`. Installing it on the server on the `/usr/local` partition was quite straight-forward with no major hassles. The `mpif77` script was modified to suit our needs and a similar `mpif90` was created. The `/usr/local/mpi/util/machines/machines.LINUX` was then modified to add two entries for each client node (as they were dual-processor SMP nodes). Jobs could now be run on the cluster using interactive `mpirun` commands!
15. A queuing system, DQS v3.0 was downloaded from <http://www.scri.fsu.edu/~pasko/dqs.html>, compiled and installed as `/usr/local/DQS/` making it available to all the client nodes through NFS. Appropriate server and client changes were then made to get it functional (i.e. adding the relevant services in `/etc/services`, starting `qmaster` on the server and `dqs_execd` on the clients) , although a few minor irritants were encountered. These were mainly owing to the bad documentation for DQS. It took a long time for me to figure out exactly how to configure the DQS to recognize a slave node, but once it was done, setting up the same for rest of the nodes was trivial. Wrapper shell scripts were then

written by me for `qsub`, `qstat` and `qdel` which not only beautified the original DQS output (which was *ugh* to begin with!), but also added a few enhancements. For example, `qstat` was modified to show the number of nodes requested by each pending job in the queue. Also, three additional shell scripts `qinfo`, `qload` and `qmem` were written to give some useful load data for the nodes and the cluster resource utilization.

16. COCOA was now fully-functional, up and running and ready for benchmarking and serious parallel jobs! As with the kernel, use of `pgcc` compiler was recommended for all the C/C++ codes. In particular, using `pgcc` with options “`-mpentiumpro -O6 -funroll-all-loops`” for typical FPU intensive number crunching codes resulted in *30 %* increase in execution speed over the conventional `gcc` compiler.

This document is maintained by Anirudh Modi <anirudh-modi@psu.edu>. Mail me if you have any questions and/or suggestions.