OSCAR Cluster Administration With C3

Brian Luethke[a], Stephen Scott[a], Thomas Naughton[a]

[a]Oak Ridge National Laboratory

The Cluster Command & Control (C3) tools developed at ORNL provide the ability to treat the cluster as a single entity. That is, a single command may be issued that will affect the entire cluster, or a specified subset of the cluster, in a predictable and controlled manner. This paper is a “hands-on” discussion of system administration of an OSCAR cluster using the C3 tools. First is a brief overview of the C3 tools and their configuration, next is a discussion of the use of C3 tools for cluster administration within an OSCAR cluster, and the discussion ends with a coverage of C3 use outside the standard scope of the OSCAR environment.

Les outils Cluster Command & Control (C3) développés au Laboratoire National d'Oak Ridge (ORNL) fournissent les capacités de traiter un ordinateur de type cluster comme une entité unique. Ainsi, une simple commande pourra affecter l'ensemble du cluster ou un sous-ensemble déterminé de celui-ci, de manière reproductible et controlée. Cet article est une introduction pratique à l'administration d'un cluster OSCAR en utilisant les outils C3. Premièrement un bref apperçu des outils C3 et de leur configuration sera présenté, suivi d'une dicussion sur l'emploi des outils C3 pour administrer un cluster OSCAR. La discussion se terminera par un exposé sur l'emploi des outils C3 en dehors du projet OSCAR.

Introduction

In a rather short time, clusters have gone on from the poor-man’s super-computer to become the defacto standard in super-computing. With the advent of the Open Source Cluster Application Resources (OSCAR) cluster distribution, it has become easy for those desiring super-computer power to exploit computing clusters to satisfy this need. However, with scores of individual machines wired together as one, the need to deal with these machines as a single entity becomes even more important. The Cluster Command & Control (C3) tools developed at ORNL provide this ability – to treat the cluster as a single entity. That is, a single command may be issued that will affect the entire cluster, or a specified subset of the cluster, in a predictable and controlled manner. While somewhat similar to parallel shell environments, C3 differs in that it provides its core functionality as an extensible framework that may be used to derive other parallel tools for both the command line as well as embedding in user applications. This paper is a “hands-on” discussion of system administration of an OSCAR cluster using the C3 tools. First is a brief overview of the C3 tools and their configuration, next is a discussion of the use of C3 tools for cluster administration within an OSCAR cluster, and the discussion ends with a coverage of C3 use outside the standard scope of the OSCAR environment.

Overview

The C3[1][2] tools suite consists of ten general use tools: *cpushimage*, *cshutdown*, *cpush*, *crm*, *cget*, *cexec(s)*, *ckill*, *clist*, *cname*, and *cnum*. The cpushimage and cshutdown are both system administrator tools that may only be used by the root user. The other eight tools may be employed by any cluster user for both system and application level use. A brief description of each core C3 tool follows – detailed information may be found on the web site and in their respective man pages.

- The **cexec** command is the C3 general utility tool as it may execute any command on each cluster node (parallel shell). If serial execution is desired (useful for debugging a cluster or if an interactive command is run) then **cexecs** should be used.
- The **cget** command retrieves files from each cluster node and deposits them in a specified directory on the local machine. To differentiate files from one another, the node’s IP or hostname and cluster name are appended to the filename.
- The **ckill** tool runs the standard Linux kill command on each cluster node for a specified process name. Unlike Linux kill, ckill must use the process name (similar to the Linux killall), as the process ID may be different on each node. Root has the ability to ckill by user / process name pair. Root may also use signals to do a broad based kill.
- **cpushimage** enables the sys-admin to push a cluster node image across cluster nodes with the option to reboot. Cpushimage makes use of SystemImager[3] and SIS[4] to accomplish image transfers.
- **cpush** may be employed by both standard users and root to copy files.
- **crm** is a clusterized version of the standard Linux rm, for file/directory removal.
- **cname** translates node name into a C3 position index value.
- **cnum** takes a range argument and returns the node names of those positions – no range returns all nodes.
- **clist** returns a list of clusters and their type.

C3 File Format

Since OSCAR[5][6] will generate a valid C3 configuration file it is not important to fully understand the C3 configuration file. As long as all updates to the configuration of the cluster are done through OSCAR, little editing of the C3 configuration file should be needed in normal operations.

The default location of the C3 configuration file is `/etc/c3.conf`. C3 allows the use of alternate configuration files through a command line option. Through the use of the configuration file it is possible to use C3 to administrate multiple clusters, but that is beyond the scope of OSCAR. If you are interested in a full explanation of the C3 configuration file please see the *c3.conf(5)* man page.

OSCAR will generate a configuration file using one line per compute node (C3 also allows the use of ranges to specify nodes in the configuration file). The following would be a typical configuration file for a four node OSCAR cluster:

```
cluster oscar_cluster {
  node0
  dead remove_this_for_zero-indexing
  node1
  node2
  node3
  node4
}
```

Figure 1

The general format of the c3.conf file is; first a cluster tag followed by the cluster name signifying the beginning of a cluster definition block. In this case the cluster name is oscar_cluster. This is the default name OSCAR supplies for the cluster. The cluster name is followed by an opening curly brace, "{". The next line is the head node specification; OSCAR will generate this from the head nodes hostname (this is required to be the internal interface of the cluster). Because C3 uses the position of the node in the configuration file for node identification on the command line, it is important to maintain position within the configuration file. OSCAR adds the line "`dead remove_this_for_zero-indexing`" resulting in one indexing for the command line. If you would rather have zero indexing remove this line. The dead tag tells C3 to remove the node from execution at run time while maintaining its position in the configuration file. When a node is temporarily offline use the "`dead`" tag to remove it from execution. The list of nodes follows, one per line. Lastly the cluster definition block is closed by a closing curly brace, "}".

It is important to note that C3 is capable of managing multiple clusters from a single configuration file. Each cluster is in it's own cluster definition block with the same format as above. If using C3's multiple cluster capabilities and using the OSCAR add/delete node interface, be aware that OSCAR will replace the c3.conf file with a new version having only the OSCAR cluster in the file. This will require you to manually add the clusters outside of OSCAR back into the file.

Command Line Interface

All commands in C3 follow the same general format: *command [options] [machine definitions] arg arg…*. The *command* is the command being run (cexec, ckill, etc). *Options* are the C3 command options specific to each command. The options may be viewed by either referring to the commands respective man page or using the --help option with each command.

The *machine definitions* block allows specifying a subset of the given cluster from the command line. Please refer to Figure 1 for the following discussion when referring to a configuration file. The first part of the machine_definition section is the declaration of which cluster you wish the command to be executed on, followed by a colon, and then the node ranges you wish to execute on. For example, "*oscar_cluster:*" would refer to the cluster in Figure 1. You may also specify the first cluster in the configuration file (the default cluster) by the use of a colon, "*:*". If no cluster is specified then C3 will execute on the first cluster in the configuration file and every node in the cluster. The default OSCAR configuration will only include one cluster. Ranges may be specified as a list of single and multiple node numbers as follows: "*:1-2,4*" will execute on the nodes in position 1, 2, and 4 of the configuration file. In figure 1 this would correlate to node1, node2, and node4. Notice that the head node does not participate in the execution of a C3 command nor does it occupy a position in the configuration file. Had the dead node line been removed then the first node would be in the 0'th position resulting in the following: "*:0-1,3*". You may also explicitly state the cluster to execute on – for example "*oscar_cluster:2-4*". Lastly each argument required for the respective command must be supplied.

For an example, we will run "*hostname*" on node2 and node4 using "*cexec :2,4 hostname*". To execute on all nodes of the default cluster we would use "*cexec hostname*".

While this is outside the scope of OSCAR, C3 easily extends to execution on multiple clusters by adding multiple cluster definition blocks to the c3.conf file. The following would restart sshd on several clusters:

```
# cexec  torc: xtorc: :  service sshd restart
```

For full documentation on the command line and running multiple clusters see the c3.conf(5) and c3-range(5) man pages.

System Administrator Use of C3

Nearly any operation a system administrator can do on a single machine can be extended into a cluster wide application. Below are several examples of common tasks in everyday maintenance of a cluster. This is not a comprehensive list of the capabilities of C3, but is intended to show the concepts involved.

Managing Services

Managing services is one of the most common operations a system administrators may do. For example, restarting the sshd server on each machine may be required if either a change in policy is required or sshd has become unstable. For this example we will be enabling X11 forwarding on the cluster. First, retrieve a copy of one of the nodes sshd_config file:

```
 # cget :1 /etc/ssh/sshd_config  .
```

This will retrieve the sshd_config file from only the node in position one (node1 in our example) to the local directory renaming it *sshd_config_oscar_cluster_node1*. Edit this file to allow X11 forwarding (see the sshd man page for details). Next this file needs to be distributed to the cluster:

```
# cpush ./sshd_config_oscar_cluster_node1 \
> /etc/ssh/sshd_config
```

Cpush is a general file scatter operation, it copies the source file (first file on the command line) from the local machine to the destination on each of the compute nodes using rsync. Once each node's configuration file has been updated you need to restart the service:

```
# cexec service sshd restart
```

Process Management

Another operation that is very useful is the maintenance of processes in general. In this example there is a user, *sgrundy*, who has spawned too many processes and is creating a high load on the compute nodes. The process the user started is called "c_model". Using C3, root would kill this process with:

```
# ckill –-user sgrundy c_model
```

The first thing to note is that ckill does not use a process id, as this will be different on each compute node, but uses the process name. In this respect ckill is much more similar to the Linux killall command. Ckill also kills every process of that name running on the compute nodes under the given user name. Only root may specify a username other than itself (by the use of the –u option). Root also has the ability to kill all processes regardless of the username by specifying "–u ALL" as one of the options. For example, assume that someone has compromised several user accounts and has started several processes that attempt to crack some other user accounts with a process called "cracker". In order to kill every instance of this program root would do the following (using a signal 9 to immediately terminate the process):

```
# ckill –u ALL –s 9 cracker
```

Log Files

It can sometimes be difficult dealing with compute node log files, both because of the amount of information present and the location of their respective log file. C3 can simplify the task of dealing with error logs and system reporting in many cases. For example, */var/log/messages* is one of the standard logging files in Linux. With C3 there are several ways of dealing with these log files depending on the amount of information and the type of information you need. The first example shows someone trying to ascertain when the above mentioned intruder logged into the compute nodes and compare them to the login times on the head node. First you would need to determine his login times on the head node by grepping the local log file for his name (this assumes he did not obtain root access on the cluster and remove such information).

```
# cat /var/log/messages | grep sgrundy
```

Next you want to execute the same command on each compute node. There are several ways of doing this depending on how you want your output formatted. In the first example all processing is done on the compute nodes:

```
# cexec 'cat /var/log/messages |grep sgrundy'
```

It is important to note that the pipe, "|", is inside the single quotes, thus keeping the local shell from interpreting the pipe. In the second example all the processing is done on the head node:

```
# cexec cat /var/log/messages |grep sgrundy
```

This will output /var/log/messages on each compute node, C3 will sort the output, then grep the contents on the local machine. Unfortunately this will not display which node the output is from. While these views are easy for a person to read it is not ideal for another script or program to read. If you need this information in a pipe friendly format then use cexec with the --pipe option as follows:

```
# cexec --pipe cat /var/log/messages \
> |grep sgrundy
```

Since each line is prepended with the cluster name and node name processing the command locally, as was done in the second example, will still be readable.

See Appendix 1 for example output of all the cexec examples.

User Tasks With C3

OSCAR NFS mounts the user's /home directory on the cluster. However, it may not always be in the user's best interest to run their application over NFS. C3 greatly eases running processes with a local file system. Assuming that the user wishes a data file (data.bin) to reside completely on the local storage of each compute node this is easily accomplished by either a cpush or a cexec. For the initial move a cexec would be best used in the following manner:

```
$ cexec cp ~/proj/data.bin /tmp
```

Since cpush uses rsync[7][8] to only move changes of a file across the network, subsequent changes made to the data file would be better handled by cpush:

```
$ cpush ~/proj/data.bin /tmp
```

If the changes are small and data.bin is a large file the reduction in time can be significant.

Distributed processes may also produce local data making the problem similar in scope to the system administrator task of parsing log files. The user can easily use cget to gather all their data from the compute nodes to the head node of the cluster:

```
$ cget /tmp/output.txt .
```

C3 Outside of OSCAR

There are instances where C3 is helpful to do operations that are beyond the scope of OSCAR. OSCAR leverages Red Hat's RPM system to simplify package management. One of the characteristics of the RPM system is that all package names must be unique. There are "work arounds" that modify the name to allow for multiple instances of effectively the same package, e.g. Python (v1.5) and Python2 (v2.x). Regardless, there are times when system administrators would like to add software without using RPMs because of this naming limitation or possibly the software is not available in RPM form. At these times, it is necessary to go outside the OSCAR framework and use standard source archives, i.e. "tarballs". On a cluster this requires a few extra steps but can be greatly simplified by using the C3 tools. Additionally, the Env-Switcher tool that accompanies OSCAR can be leveraged.

The following sections summarize the Env-Switcher tool and provide details on how to use C3 & Env-Switcher to add software to OSCAR clusters from standard source tarballs.

Env-Switcher

Env-Switcher, or simply Switcher [9], is a tool to manage environment settings and is based upon Modules [10]. Switcher enables an administrator to cleanly manipulate system wide tags that are used to setup user environment, e.g. PATH, PVM_ROOT. Switcher allows you to selectively load items into the environment, which helps when managing mutually exclusive packages. The canonical example is the situation where multiple MPI implementations are installed on a system. The switcher modules allow for a system wide default, but also enable users to select an alternate version from the pool of available MPIs.

```
 # Show available MPI's on system
$ switcher mpi --list
lam-6.5.7
mpich-1.2.4

 # Show current default
$ switcher mpi --show
system:default=mpich-1.2.4
system:exists=true

$ which mpirun
/opt/mpich-1.2.4/bin/mpirun
```

Switcher enables the system wide default to be overridden at a per user basis. The user would set an alternate MPI implementation by setting the *mpi* tag as follows:

```
$ switcher mpi = lam-6.5.7
```

Switcher changes take effect on subsequent shell invocations and the environment is maintained across non-interactive shells, e.g. rsh/ssh.

These available *tags*, e.g. lam-6.5.7, mpich-1.2.4, are setup manually by the administrator (or by OSCAR) via a module file. These module files state what environmental changes are necessary for the package. Switcher is smart enough to undo these modifications as needed[1]. The full format is beyond the scope of this article but much can be gleamed from a simple example. A source listing of the

[1] The actual shell modifications are performed by the Modules tools.

mpich2-0.91 module file is given in Appendix 2[2]. The actual environment modifications for the package are lines 20 & 21. This is used the in the next section when discussing the MPICH2-0.91 installation from source tarball.

The "*tarball example*"

The source tarball method of software distribution is fairly standard and often includes an *Autoconf* and *Make* facility. The following example shows the installation of a development version of MPICH [11][12] onto a running OSCAR cluster from source tarball.

Once the software has been downloaded and extracted the software is compiled and built as a standard user.

```
$ tar -zxvf mpich2-0.91.tar.gz -C /tmp
$ cd /tmp/mpich2-0.91
$ ./configure --prefix=/opt/mpich2-0.91
<<Cut output, configures successfully>>

$ make
<<Cut output, builds successfully>>
```

At this point the software has been configured and compiled and is ready to be installed. After becoming root (via `su`) the software is installed into the */opt/mpich2-0.91* directory.

```
 # Become the SuperUser for install
$ su -
# cd /tmp/mpich2-0.91
# make install
```

Since, the compute nodes have the same hardware configuration this compiled/installed result can simply be archived, `tar`'d, and pushed, `cpush`, to all nodes in the cluster[3].

```
# cd /opt
# tar -cvf /opt/mpich2-0.91.tar mpich2-0.91/
<<Cut output, archives successfully>>

# cpush /opt/mpich2-0.91.tar  /opt
```

The archive is extracted on the compute nodes using the change directory flag "-C" to place the output in `/opt`. Single quotes surround the `cexec` command.

```
# cexec 'tar -xvf /opt/mpich2-0.91.tar -C /opt'
```

This puts the software on all nodes and we are now ready to create the Switcher module file as shown in Appendix 2. This file can reside anywhere but it is commonly placed in a "share/" directory under the package's directory. The file is named whatever the *tag* should display, e.g. *mpich2-0.91*. Therefore in this case the directory structure and file would be "/opt/mpich2-0.91/share/mpich2-0.91". This is created on the head node and mirrored on the compute nodes.

```
# mkdir /opt/mpich2-0.91/share
# cexec 'mkdir /opt/mpich2-0.91/share'
# vi /opt/mpich2-0.91/share/mpich2-0.91
<<Add contents as shown in module example>>

# cpush /opt/mpich2-0.91/mpich2-0.91
```

Finally, to make Switcher aware of the new software a new tag name must be added, *mpich2-0.91*. The `switcher` command is executed on the head node and compute nodes so the changes appear in the respective Switcher configuration files.

```
# switcher mpi --add-name mpich2-0.91 \
> /opt/mpich2-0.91/share --silent --force

# cexec 'switcher mpi --add-name \
> mpich2-0.91 /opt/mpich2-0.91/share \
> --silent --force'
```

At this point `switcher mpi --list` will display "mpich2-0.91" as one of the available MPI implementations on the cluster. This approach of C3 and Switcher could also be automated on development clusters to offer "cvs" or "devel" snapshots, e.g. *lam-cvs* or *pvm-devel*. The available commands and syntax for Switcher are available in the SWITCHER(1) manual page.

Remote Administration

Another benefit of C3 is it's remote cluster administration capabilities. Through the use of C3's configuration file it is possible to control a cluster from your desktop with the same interface and power that you would have using C3 locally. The basic requirements are that the person has login access to the head node of the remote cluster and that C3 is installed correctly – OSCAR takes care of last part for you. Next you would follow C3's installation procedure outlined in the install section of C3 documentation on your desktop. Next is the setup of the c3.conf file. For this example, you will use what is referred to as an indirect remote cluster. This is essentially a pointer to the head node of the cluster. No information about the layout of the cluster is stored locally, C3 will determine the layout of the cluster from the head node. For example, assume a user wishes to remotely access the cluster in Figure 1, the indirect remote configuration file

```
cluster torc {
       :garm.csm.ornl.gov
}
```

Figure 2

[2] This was manually added and adapted from the existing LAM and MPICH module files included with OSCAR.

[3] The software can be built on the individual nodes by `cpush`'ing the "mpich2-0.91.tar.gz" tarball to the nodes and performing the commands using `cexec`.

in Figure 2 would be used on the local machine.
Cluster is a key word denoting the beginning of a cluster block, *torc* is the name of the cluster (any name is acceptable). The line ":garm…" signifies that the cluster is an indirect remote cluster. "garm…" is the external interface name of the head node of the clusters (the DNS name – it could also be an IP address). All of the commands shown work in the exact same manner as previously described – from the command line point of view it is impossible to tell if a cluster is remote or local.

Conclusion

This paper provided a number of examples using the C3 tools for cluster administration and operation tasks both within the OSCAR framework and outside the normal scope of OSCAR. While this paper is not intended to serve as a comprehensive list of uses for C3, it does show a number of significant uses for the tool suite and it is intended to provide examples on which the reader may build their own C3 derived works. The C3 development team at ORNL is in the process of writing a more comprehensive documentation set for the use of C3 for the administration and operation of machines ranging from single cluster -to- federated clusters (multi-clusters) and also for single workstations with a single point of administration. Look for this document on the C3 web site in the near future.

References

[1] M. Brim, R. Flanery, A. Geist, B. Luethke, and S. Scott Cluster Command & Control (C3) tools suite. In *To be published in, Parallel and Distributed Computing Practices, DAPSYS Special Edition,* 2002.

[2] "C3", http://www.csm.ornl.gov/torc/C3/

[3] "SystemImager", http://www.systemimager.org/

[4] "SIS", http://sisuite.org/

[5] Thomas Naughton, Stephen L. Scott, Brian Barrett, Jeff Squires, Andrew Lumsdaine, and Yung-Chin Fang. The Penguin in the Pail – OSCAR Cluster Installation Tool. In *Proceedings of SCI'02: Invited Session – Commodity, High Performance Cluster Computing Technologies and Application*, Orlando, FL, USA, 2002.

[6] "OSCAR", http://www.OpenClusterGroup.org/OSCAR

[7] A. Tridgell and P. Mackerras. The rsync algorithm. Technical Report TR-CS-96-05, Australian National University, Department of Computer Science, June 1996.

[8] "rsync", http://samba.anu.edu.au/rsync/

[9] "Env-Switcher", http://env-switcher.sourceforge.net.

[10] John L. Furlani and Peter W. Osel, "Abstract Yourself With Modules, In *Proceedings of the 10th Large Installation Systems Administration Conference (LISA'96)*, pages 193-204, Chicago, IL, September 29 -- October 4, 1996.

[11] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A High-performance, portable implemntation of the MPI message passing interface standard. *Parallel Computing* 22(6):789-828, September 1996.

[12] "mpich", http://www-unix.mcs.anl.gov/mpi/mpich

Research supported by the Mathematics, Information and Computational Sciences Office, Office of Advanced Scientific Computing Research, Office of Science, U. S. Department of Energy, under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

APPENDIX 1

Use of quotes with cexec.

cexec 'cat /var/log/messages |grep sgrundy':

```
[root@node0 test]# cexec 'cat /var/log/messages |grep sgrundy'
************************  local  ************************
processing node  node1
processing node  node2
--------- node1---------
Mar 11 18:23:01 node1 sshd[1073]: Accepted publickey for sgrundy from 10.0.0.50 port 32805 ssh2
Mar 11 18:23:01 node1 sshd(pam_unix)[1073]: session opened for user sgrundy by (uid=0)
Mar 11 18:23:01 node1 sshd(pam_unix)[1073]: session closed for user sgrundy

--------- node2---------
Mar 11 18:23:01 node2 sshd[1208]: Accepted publickey for sgrundy from 10.0.0.50 port 32806 ssh2
Mar 11 18:23:01 node2 sshd(pam_unix)[1208]: session opened for user sgrundy by (uid=0)
Mar 11 18:23:01 node2 sshd(pam_unix)[1208]: session closed for user sgrundy
```

cexec cat/var/log/messages |grep sgrundy

```
[root@node0 test]# cexec cat /var/log/messages |grep sgrundy
Mar 11 18:23:01 node1 sshd[1073]: Accepted publickey for sgrundy from 10.0.0.50 port 32805 ssh2
Mar 11 18:23:01 node1 sshd(pam_unix)[1073]: session opened for user sgrundy by (uid=0)
Mar 11 18:23:01 node1 sshd(pam_unix)[1073]: session closed for user sgrundy
Mar 11 18:23:01 node2 sshd[1208]: Accepted publickey for sgrundy from 10.0.0.50 port 32806 ssh2
Mar 11 18:23:01 node2 sshd(pam_unix)[1208]: session opened for user sgrundy by (uid=0)
Mar 11 18:23:01 node2 sshd(pam_unix)[1208]: session closed for user sgrundy
```

cexec --pipe cat /var/log/messages |grep sgrundy

```
[root@node0 test]# cexec --pipe cat /var/log/messages |grep sgrundy
local node1: Mar 11 18:23:01 node1 sshd[1073]: Accepted publickey for sgrundy from 10.0.0.50 port 32805
ssh2
local node1: Mar 11 18:23:01 node1 sshd(pam_unix)[1073]: session opened for user sgrundy by (uid=0)
local node1: Mar 11 18:23:01 node1 sshd(pam_unix)[1073]: session closed for user sgrundy
local node2: Mar 11 18:23:01 node2 sshd[1208]: Accepted publickey for sgrundy from 10.0.0.50 port 32806
ssh2
local node2: Mar 11 18:23:01 node2 sshd(pam_unix)[1208]: session opened for user sgrundy by (uid=0)
local node2: Mar 11 18:23:01 node2 sshd(pam_unix)[1208]: session closed for user sgrundy
```

APPENDIX 2

Listing of module file: ***mpich2-0.91***

```
#%Module -*- tcl -*-
#
# MPICH modulefile for OSCAR clusters
#

proc ModulesHelp { } {
  puts stderr "\tThis module adds MPICH to the PATH and MANPATH."
  puts stderr "\tHence, the mpicc, mpiCC, mpif77, and mpirun commands"
  puts stderr "\tthat you run will be from MPICH."
}

module-whatis   "Sets up the MPICH environment for an OSCAR cluster."

# Don't let any other MPI module be loaded while this one is loaded

conflict mpi

# It's real simple.  Append to the PATH and to the MANPATH.

append-path PATH /opt/mpich2-0.91/bin
append-path MANPATH /opt/mpich2-0.91/man
```