HPC Federated Cluster Administration with C3 v3.0

Brian Luethke and Stephen Scott

Abstract[1]

KEYWORDS: Cluster, Administration, Federated-cluster, multi-cluster

While administrating TORC, HighTORC, and the various other computation clusters at Oak Ridge National Laboratory (ORNL), it quickly became apparent that a solution for the administration of federated clusters, or "clusters of clusters," was needed. The few cluster tools available when this work began could barely manage a single cluster let alone a number of clusters. They also required that the user be directly logged onto a cluster machine. This meant to administer ten clusters required that the administrator login and repeat a task on each of the clusters. This solution does not scale and therefore is unacceptable for our environment. Thus, a solution was desperately needed whereby an administrator could perform duplicate operations across multiple clusters and portions thereof in a scalable and secure fashion from a single location that may not be directly logged onto the cluster being administered. Thus the development of version 3.0 of the Cluster Command and Control (C3) tool suite began.

[1]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.

C3 prior to version three required, as most tools do, that one is physically logged into a cluster in order to perform administration operations. The few existing tools that permit remote administration of clusters were all web based, therefore they suffered security problems and set up hassles associated with installing and maintaining a web server. What we tried to design is an easy to use command line interface that is powerful enough to do most system administrating jobs and secure. These tools also needed to be useful to regular users in building and maintaining their distributed applications. C3 version 2.x already met those requirements so we decided to emulate their functionality while adding the ability to do this with multiple clusters. This paper describes the use of the C3 3.0 tool suite.

1 Brief Description of Commands

Ten general use tools have been developed in the effort thus far: cexec, cget, ckill, cpush, cpushimage, crm, cname, cnum, clist, and cshutdown. 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.

The cexec command is the general utility tool of the C3 suite in that it enables the execution of any command on each cluster node. As such, cexec may be considered the clusterized version of rsh/ssh[1]. A command string passed to cexec is executed "as is" on each node. This provides a great deal of flexibility

in both displaying the command output and arguments passed in to each instruction.

The cget command will retrieve the given files from each cluster node and deposit them in a specified directory location on the local machine. Since all files will originally have the same name, only from different nodes, an underscore and the node's IP or hostname and cluster name are appended to each file name. Whether the IP or hostname is appended depends on which is specified in the cluster specification file. Note that cget operates only on files and ignores subdirectories and symbolic links

The ckill tool runs the standard Linux 'kill' command on each of the cluster nodes for a specified process name. Unlike 'kill', ckill must use the process name as the process ID (PID) will most likely be different on the various cluster nodes. The root user has the ability to further indicate a specific user in addition to process name. This enables root to kill a specific user's process by name and not affect other processes with the same name but owned by other users. Root may also use signals to effectively do a broad based kill command.

The cpushimage enables a system administrator logged in as root to push a cluster node image across a specified set of cluster nodes and optionally reboot those systems. This tool is built upon and leverages the capabilities of SystemImager[2]. While SystemImager provides much of the functionality in this area, it fell short in that it did not enable a cluster-wide push for image transfer. cpushimage essentially pushes a request to each participating cluster node to pull an image from the image server. Each node then invokes the pull of the image from the cluster image server. Of course, this description assumes that SystemImager has already been employed to capture and store a cluster node image on the cluster image server machine.

While cpushimage has the ability to push an entire disk image to a cluster node, as an application support tool, it is too cumbersome when one simply desires to push files or directories across the cluster. Furthermore, cpushimage is only available to system administrators with root level access. From these restrictions grew the desire for a simplified cluster push tool, cpush, providing the ability for any user to push files and entire directories across cluster nodes. cpush uses rsync[3] to push files from server to cluster node.

crm is a clusterized version of the standard 'rm' delete file/directory command. The command will go out across the cluster and attempt to delete the file(s) or directory target in a given location across all specified cluster nodes. By default, no error is returned in the case of not finding the target. The interactive mode of 'rm' is not supplied in crm due to the potential problems associated with numerous nodes asking for delete confirmation.

cshutdown is a cluster wide shutdown operation. cshutdown also has the ability to boot an alternate lilo label for a single boot. This allows you to test a new kernel without making permanent changes or to boot into another operating system temporarily.

The cname command returns the node number based on the cluster and node name supplied at the command line. Both this command and the cnum command are useful when the node names of your cluster and their positions in the configuration file are not easily paired.

The cnum command returns the node name based on the node number and cluster supplied at the command line.

The clist command returns a list of clusters defined in the cluster configuration file and the

type of cluster.

2 Installation and Configuration

C3 version 2.x used a list of nodes, one per line, to define a cluster. While it is possible to have several clusters in this list, each node had to be visible to the machine that the C3 command was run from. Many clusters only have the head node exposed with the individual compute nodes on a private network. A list of nodes also does not allow the granularity need to specify which cluster to execute the command from. In version three you have the concept of a cluster configuration block. Each block defines a single cluster – its external entry point, an optional internal entry point (if the nodes are on a private network) and then the list of nodes. This type of cluster is called a direct cluster – the configuration of the cluster is known by the C3 command before runtime. It is possible to have both a local cluster (the machine that the C3 command is run from is the head node) and a remote cluster (the machine that the command is run form is not the head node) use a direct method of definition. One of the advantages of this scheme is that it is possible to build both subsets and supersets of a given cluster. The major drawback to using a direct cluster block on a remote machine is that the user must keep track of which machines are offline and which are online – this can be a real headache. C3 solves this problem with an indirect remote cluster. In this type of cluster block the only thing the C3 command knows is the external interface of a remote cluster. When a C3 command is run it will execute that command on the remote cluster using the default cluster configuration block (the first cluster in the configuration file) on that cluster. In this way a user using his or her desktop need not know how many machines are currently on each cluster they use, only that the head node they have specified exists and has a working copy of C3 version 3 on it. Below is an example configuration file:

```
cluster home { #the cluster
# named home.
# The default cluster as it is the
# first in the configuration file
node0 #the head node,
#      external name only
node[1-15] #the compute nodes
}

cluster TORC { #the cluster
#                named TORC
heimdal:node0 #the head node,
# heimdal is
# the external interface name
# and node0 is
# the internal interface name
node[1-64] #compute nodes
}

cluster htorc { #the cluster
#                named htorc
:htorc-00 #this is a indirect remote
#cluster, htorc-00 is the external
#interface name
}
```

3 Usage

In early versions of C3 we were tied to the implementation of a PERL[4] package to parse the command line. In version 3 we parse the command line ourselves giving us much more flexibility. When designing our API we tried to stay as close to the respective Linux tool as possible so that a user would have a minimum amount of learning to do. We also now have the ability to specify node ranges on the command line. This is very useful for system administrators for doing rolling upgrades. An example of the new API that would execute hostname on the default cluster would be as follows:

```
cexec hostname
```

Extending the paradigm to federated clusters is just as simple, simply specify the clusters you wish to execute on. To execute on the default cluster and on nodes four through six and node eight on the cluster named TORC would be as follows:

```
cexec :  TORC:4-6,8 ls -l
```

In the above example the command is cexec (a general purpose exec, similar to a cluster wide rsh) ":" signifies the default cluster, TORC: signifies the cluster named TORC in the configuration file, 4-6,8 is a node range, and ls –l is the command to be run. There are several ways the C3 tools were designed to be used. The most basic way is from the command line, one command at a time. Next is writing scripts using the C3 tools. And the third way is extending the C3 tools themselves for site-specific functionality.

A good example of using the tools directly on the command line is effecting rolling upgrades. Using cpushimage and SystemImager it is very simple to test out a cluster configuration. A system administrator could build a small test cluster using SystemImager to clone the current cluster. After installing the software and testing the new image you would find it acceptable for roll-out. You would again use SystemImager to retrieve the image from the test cluster. Next, make sure you have a backup image from the production cluster and use cpushimage to test it on a small number of machines. Assuming the image name is new_image the sample command would look like:

```
cpushimage -reboot :0-3 new_image
```

This pushes the new image to only the first four nodes in the cluster and reboots the machine. This allows you easily test the new image. Assuming it works, just type the same command as before removing the :0-3 from the command. That would push the image to every node in the cluster. One of the nice features of this is if you later find a problem with the new image it is easy to roll back to an earlier image that is known to work.

Using C3 from the command line is also useful for a general user of a cluster. Using the indirect remote cluster a user can develop an application on their desktop and easily distribute the binary to either a single cluster or multiple clusters (via cpush). Using C3 in this way makes a cluster a "black box" – that is the user only has an indiscriminate resource out there called a cluster. They do not need to keep up with the addition of new nodes nor if a few nodes have been taken offline. One of the features of writing the C3 power tools in a platform independent language is that the tools only must run homogeneous within there self. For example, take the above user who wishes to push a binary to several clusters. One of the clusters is an Intel PC cluster and the other is an alpha cluster, both running Linux. Using the GNU gcc cross compiler the user has compiled a binary for an Intel machine and a binary for an alpha cluster (it is possible that their desktop be a power macintosh). They are using text data files so the data can be used by all systems. Assuming that the head nodes home directory is NFS mounted the following commands would distribute the application, its data, and run it:

```
cpush --head Intel: app.Intel app
cpush --head alpha: app.alpha app
cpush --head Intel: alpha: data.txt
cexec --head Intel: alpha: app
```

Notice that once the binaries are renamed when pushed out to the cluster so that a single cexec can start the application. This demonstrates that the level of homogeneity required by each command can be different. In the first two lines

each cluster must be separated but in the last two command their actual architecture is irrelevant as the command being run is identical on each cluster. This is a power paradigm for both users and system-administrator.

The next way the C3 Power Tools can be used is with scripting. Using scripting and image management with C3 is useful for effecting changes for a single user. With C3, once the image is built, it is quite easy to temporarily install a new image with differences ranging from a slightly different communications package, to a different flavor of Linux one the fly. For example we have a user who requires kerberos[5] to be installed in order to run their code, we do not wish to support or maintain this. It only a matter of an hour or so of time (because of the size of the image being transferred, all the interaction required is the initial command run and after it is done checking to make sure the machines rebooted) to switch to a completely different image complete with their data and special configuration fully operational. With scripting this can even be done within a PBS script to change the image before the run and to restore it to the default one afterwards.

The next way the C3 can be used is in scripting. While administrating our clusters one of the largest problems we encountered was generating and managing ssh keys when creating a new user. Unfortunately it is very difficult to write a tool that is a general purpose ssh manager as the policies differ from site to site. While this script is included with C3 it is not part of C3 proper – it is in the examples directory due to the above problems. See figure 1 for the example code. This script works by getting the user-name and group of a new user from the command and then calling the standard Linux adduser binary. next it sets the password for that user with the standard Linux password facility. Then, using C3 all the files that were touched are sent to the cluster nodes, and the any needed directories are created (/home is NFS mounted so the directory only needs to be created on the head node). Lastly the ssh-keys are generated and the authorized_keys2 file is created (to allow users to ssh to one of the compute nodes without the use of a password). Where this script and C3 really show the power available is in combining this method with a command line. Assuming this script is located in /usr/sbin a command as follows:

```
cexec -all /sbin/add_user zbml1 users
```

would add the user zbml1 with the group users to every cluster that the machine this is executed on has access to. Thus it is just as easy to add a user to one cluster, as it is five. The only redundant typing would be when the password is generated but it is trivial to write an expect script that handles this for you.

We also use the C3 tools to take the place of some of the daemons we would probably run. We do not run to run NTP[6] to keep our cluster's date in sync so we wrote our own bash script that gets the current date on the head node and then issues a cexec to set the cluster nodes to the current date. The script is ran once a day in a cron job to keep the cluster in sync.

The third and most powerful way that the C3 Tools can be used is in extending them. When we wrote C3 one of the focuses was to make it modular. We chose Python[7] as a language both because it is well known and common and it is also very easy to write packages for. The two main parts we separated out of the code into packages are the command line parser and the configuration file parser. This allows you to add functionality such as hardware monitoring, BIOS maintenance, or any functionality you would choose. Splitting the file parser into

its own package also allows a system administrator to both read the c3.conf file but to also use it as a base. A nice example of this would be setting a cron job that once a night reads the c3.conf file and generates an up to date configuration file for PVM. The command line package lets a system administrator to create new tools that have the look and feel of the C3 tools making it easier on their users learning a new command line API.

Included in C3 Version 3.1 is a "contrib" directory where the script mentioned here and other are included. The scripts in this directory are offered for use if your site is configured such that they are applicable (such as the add_user script assumes NFS mounted home directories and use of ssh). These scripts are also intended to be concrete examples of extending and using C3. Also included are full package documentation on the command line parsing object and the c3.conf parser.

4 The Future

Versions 3.x of the C3 Power Tools offer both a system-administrator and a general user great power for managing both a single cluster and multiple clusters. One of the areas that the current versions of C3 are short in is scalability. We are currently working on a version 4 of the tools that take into account homogeneity in clusters and their topography in order to scale the commands into clusters with thousands of nodes.

5 Conclusion

Version 3.0 of the C3 tools suite is a major advance in the tools. The ability to administrate multiple clusters simultaneously from anywhere you can access each head node is very useful. In the same number of command in C3 v2.7 it would take to add a user to a cluster you can now add a user to any number of clusters. Users who write a distributed application that will run on several clusters now have an easy way to distribute their application to the clusters, even form their own desktop.

6 References

1. http:/www.openssh.org/
2. http://www.systemimager.org/
3. http://samba.anu.edu.au/rsync/
4. http://www.perl.com/
5. http://web.mit.edu/kerberos/www/
6. http://www.eecis.udel.edu/sim/
7. http://www.python.org/
8. http://www.csm.ornl.gov/torc/

Figure 1: add_user script

```
#!/usr/bin/env python2
############################################################
#this script adds a user to the local cluster.  It assumes that
#the home directory is nfs mounted and no others are.  Put this in
#a well known location (/root/bin in our case) so it can be called
#with cexec.  This is an example of using the C3 tools in a script
#to automate tasks on a cluster.
############################################################
import os, sys
try:  #get user name from command line
        user_name = sys.argv[1]
except IndexError:
        print "must supply a user name"
        sys.exit()
try:  #get group name from command line
        user_group = sys.argv[2]
except IndexError:
        print "must supply a group name"
        sys.exit()
#adduser to local machine
os.system(  "adduser -g " + user_group + " -m " + user_name )
#set password for local user
os.system(  "/usr/bin/passwd " + user_name )
#distribute the password files and group files to compute nodes
os.system(  "/opt/c3-3/cpush /etc/passwd" )
os.system(  "/opt/c3-3/cpush /etc/shadow" )
os.system(  "/opt/c3-3/cpush /etc/group" )
os.system(  "/opt/c3-3/cpush /etc/gshadow" )
#create ssh directory
os.system(  "mkdir /home/" + user_name + "/.ssh")
#since the script is run by root change ownership of users file to that user
os.system(  "/opt/c3-3/cexec chown -R " + user_name + ":" + user_group + "
/home/" + user_name )
#create users ssh-keys
os.system(  "/bin/su " + user_name + " -c \'/usr/bin/ssh-keygen -b 512 -t dsa
-N \"\" -f " + os.path.expanduser( "~" + user_name ) + "/.ssh/id_dsa\'" )
#set up keys such that they can loginto nodes without a password
os.system(  "cp /home/" + user_name + "/.ssh/id_dsa.pub /home/" + user_name +
"/.ssh/authorized_keys2" )
#make sure everything in their directory is owned by them
os.system(  "/opt/c3-3/cexec chown -R " + user_name + ":" + user_group + "
/home/" + user_name )
```

Proceedings of the
Ottawa Linux Symposium

June 26th–29th, 2002
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
Stephanie Donovan, *Linux Symposium*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Wild Open Source, Inc.*