

C3 Power Tools

Brian Luethke and Stephen L. Scott^Δ
Computer Science & Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN 37830-6367 USA

ABSTRACT

Through the administration and use of eXtremeTORC, HighTORC, TORC and the various other HPC 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. While a few cluster tools did exist when this work was started, there were none with the vision to handle multiple clusters as a single cohesive computing environment. These other tools also required that the administrator or user be directly logged into a user account on one of the clustered machines. This resulted in at least one duplicate operation for each cluster being addressed. It is obvious that this process does not scale. Thus a solution was needed whereby an administrator or general HPC user could perform duplicate operations across multiple clusters, and portions thereof, in a scalable and secure fashion from a single location not necessarily one where the user is logged in. Thus was the drive for the development of version 3.0 of the Cluster Command and Control (C3) power tools.

KEYWORDS: cluster, administration, federated-cluster, multi-cluster

1. Introduction

Prior to version 3.0, C3 like other tools, also required that one be physically logged in to the cluster where operations were to be performed. The few existing tools that permit remote administration of clusters were all web based, therefore they suffered the associated security problems and setup grief associated with the installation and maintenance of a web server. This is the 3rd generation of the C3 tools and is an even more powerful and secure command line cluster interface than its predecessors. While version 2.x provided much of this functionality, it was felt that further

strides in the area of multi-cluster accessibility by providing what we call the *single system illusion*.

Ten extensible 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

^Δ Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. DE-AC05-00OR22725.

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.

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

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 `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 simple list of nodes also does not allow the granularity needed to specify from which cluster a command is to be executed. Version three provides for the conceptual abstraction of the cluster configuration in the specification file. Here each block defines a single cluster – it's external entry point, an optional internal entry point if the nodes are on a private network, and the list of cluster compute nodes. This type of cluster is called a direct cluster as the configuration of the cluster is known by the C3 command prior to runtime. Conversely, an indirect cluster is when only the cluster itself is known via its external interface – generally the head node. A local cluster is one where the machine that initiates the C3 command is that cluster's head node. It follows that a remote cluster is one where the machine that initiates the C3 command is not the target cluster's head node. It is possible to have both a local and remote cluster configuration in the C3 configuration. A significant advantage of this scheme is that it is possible to build both subsets and supersets of any reachable cluster.¹

The major drawback to using a direct cluster block on a remote machine is that the user must manually keep track of which machines are offline and which are online – this manual process is tedious at best. C3 solves this problem with an indirect remote cluster. As stated above, in this type of cluster block, C3 only knows is the external interface of the remote cluster. When a C3 command is run, it is resolved via its execution on the remote cluster using the default cluster configuration block – the first cluster specified in that cluster's configuration file. Thus the user need not know from their desktop how many computation nodes are available at any given instant. The only requirement is that the cluster head node's external interface exist and that this head node have a working version three of C3 available. Below is an example configuration file:

¹ A cluster is defined as "reachable" providing the IP address of the head node is known and externally available.

```

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 an indirect
                #remote cluster,
                #htorc-00 is the
                #external interface
                #name
}

```

3. Command Line Usage

Early versions of C3 were tied to the implementation of a PERL[4] package to parse the incoming command line. In order to provide a greater flexibility in version three, with respect to command line content, the decision was made to internally implement the command line parser. The goal was to keep the C3 API as closely aligned with the associated Linux command as possible in order to reduce the learning curve. Version three also added the ability to specify node ranges on the command line. This feature has proven to be very useful by system administrators when performing rolling upgrades to their system's software. An example of the new API to execute the `hostname` command on the default local cluster on nodes four through six and node eight on the cluster named `torc` is as follows:

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

In the above example, the command is `cexec` – a general purpose clusterized `exec` similar to a cluster wide `rsh`. The colon “:” signifies that this command is to execute on the default cluster. The `torc:` signifies to use information from cluster `torc` in the configuration file. The node range (nodes 4, 5, 6) is specified by the `4-6` qualifier and `8` indicates itself. Last, the `ls -l` indicates that the standard

linux `ls -l` command is to be run on the specified cluster and node combination.

4. Example Usage

The C3 power tools were designed and implemented such that they may be used in three manners. First, the most basic is to invoke directly from the command line prompt – one command at a time. Second, is to group the C3 command line operations into scripts. Last is to use the base C3 commands to write derivations and extensions to the basic tool kit to provide for site-specific functionality.

A prime example of using the tools directly from the command line is when performing rolling system upgrades. Through the use of `cpushimage` and `systemimager`, one may easily test a new cluster configuration. For example, a number of machines may be configured into a test cluster by using `systemimager` to clone the current production cluster. If after the installation of software and testing, the new cluster configuration is deemed acceptable, one may retrieve this new cluster image via `systemimager` and then subsequently push that disk image to all other computation nodes via the `cpushimage` command. Note that it is highly recommended that production images be stored for backup. An example follows where the image name is `new_image`:

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

This command pushes the new image to only the first four nodes in the cluster and reboots the machine. This permits one to easily test the new image on a sub-cluster. If this new configuration is found to be acceptable, the following command will push the `new_image` throughout the entire cluster node range.

```
cpushimage -reboot new_image
```

Removing the `:0-3` from the command will cause the above to execute across all cluster nodes in the default configuration file `c3.conf`. If it is later discovered that this `new_image` is later not wanted and instead the `old_image` is desired, the administrator may simply “roll back” by issuing the following command – assuming they previously saved an old image.

```
cpushimage -reboot old_image
```

C3 from the command line is also very useful for a general cluster user. 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 using the `cpush` command. Using C3 in this way makes a cluster a “black box” computation engine – that is the user only knows that they have a resource out specified by a cluster name². They do not need to keep up with the addition of new nodes nor if a few nodes have been taken offline. If the application produces local output `cget` supplies an easy way to retrieve distributed results.

Another good use of image management with C3 is 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 communication package, to a different flavor of Linux on the fly. For example, if a user requires `kerberos`[5] to be installed prior to the execution of their application, it is simple to effect this addition as a part of staging the user’s application. Once scripted, this makes the application a “run and leave it alone” process. After the application has terminated, the process may then continue on to reload the original operating environment and reboot the machines into their original operational state.

Another beneficial way to use C3 is through the scripting of command sequences. While administering clusters at ORNL, one of the more aggravating processes encountered was the generation and managing of `ssh` keys when creating adding a new user to the clusters. Since site policies differ greatly, it is difficult to write a tool that may stand in as the general purpose `ssh` manager. Thus while this script is included with C3, it is not part of C3 proper. Please refer to figure 1 for the code. This code is included in the `examples` directory due to the above possible site conflicts. This script receives the `username` and `group` of the user to add. It then calls the standard Linux `adduser` binary, sets the user password via the standard Linux password facility and then invokes C3 commands to send all the files “touched” to the cluster nodes while creating any new directory structures³. Lastly the `ssh-keys` are generated and the `authorized_keys2` file is created, thusly enabling 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 that below will add the user `zbm11` with the group `users` to every cluster that the machine where this command is executed has access.

```
cexec -all /sbin/add_user zbm11 users
```

Thus it is just as easy to add a user to one cluster, as it is to add that user to twenty-five or more clusters. The only redundant typing is in response to when the password is generated. However, it is trivial to write an `Expect`[9] script to handle this. C3 tools may also be used to replace some cluster wide daemons that would otherwise have to be run. For example, `NTP`[6] is not necessary to keep the node’s time in sync. A bash script created to retrieve the current date from the head node and then issue a `cexec` to set the cluster nodes to the current date replaces this. This script is run once a day in a cron job to keep the cluster in sync. Thus eliminating one of many additional daemon processes running on a typical cluster node.

The third and by far the most powerful way to use the C3 tools is by extending the tools themselves. While the initial versions of C3 was written in Perl, this later version was written in Python[7]. One focus of the Python implementation of C3 was to make it modular thus simplifying their extensibility. Python was choose as it has been increasing in popularity for Linux tools and easily lends itself to package creation. The two significant parts of C3 that were separated from the original code base and moved into packages are the command line parser and the configuration file parser. This allows functionality such as hardware monitoring or BIOS maintenance among others, to more easily be added. Separating the file parser into its own package enables the system administrator to both read the `c3.conf` file as well as use it as a base instance of the cluster architecture. A good example of this use is the setting of 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 create new tools that have the look and feel of Linux and C3 tools thus reducing the learning curve of the command line API.

5. Conclusion

Version 3.0 of the C3 tools suite is a major advance in cluster aware tools. The ability to administrate multiple clusters simultaneously from anywhere one can access each head node is very useful. Thus advancing the single system illusion of one unified operating environment. Version three is a significant advancement over prior versions of C3 in that now one command may be used to launch tasks, execute commands, and run scripts across numerous clusters even with their compute nodes on private networks. Users and administrators now have a way to invoke a single launch point that will effect an operation across multiple cluster computing environments – from their desktop environment.

² As far as C3 is concerned, the user need not even know that it is a cluster. Simply specifying the name that represents the cluster will suffice with out any prior knowledge of the underlying architecture.

³ Here `/home` is NFS mounted so that the directory only needs to be created on the head node.

6. Acknowledgements

The authors would like to acknowledge the assistance of Phil Pfeiffer of East Tennessee State University as well as the entire ORNL cluster computing team.

7. References

1. www.openssh.org
2. www.systemimager.org
3. samba.anu.edu.au/rsync/
4. www.perl.com
5. web.mit.edu/kerberos/www/
6. www.eecis.udel.edu/~ntp/
7. www.python.org
8. www.csm.ornl.gov/torc
9. expect.nist.gov/

```

#!/usr/bin/env python2

import os, sys

try:
    user_name = sys.argv[1]
except IndexError:
    print "must supply a user name"
    sys.exit()

try:
    user_group = sys.argv[2]
except IndexError:
    print "must supply a group name"
    sys.exit()

string_to_execute = "adduser -g " + user_group + " -m " + user_name
os.system( string_to_execute )

string_to_execute = "/usr/bin/passwd " + user_name
os.system( string_to_execute )

string_to_execute = "/opt/c3-3/cpush /etc/passwd"
os.system( string_to_execute )

string_to_execute = "/opt/c3-3/cpush /etc/shadow"
os.system( string_to_execute )

string_to_execute = "/opt/c3-3/cpush /etc/group"
os.system( string_to_execute )

string_to_execute = "/opt/c3-3/cpush /etc/gshadow"
os.system( string_to_execute )

string_to_execute = "/opt/c3-3/cexec mkdir /home/" + user_name
os.system( string_to_execute )

string_to_execute = "mkdir /home/" + user_name + ".ssh"
os.system( string_to_execute )

string_to_execute = "/opt/c3-3/cexec chown -R " + user_name + ":" + user_group + " /home/"
+ user_name
os.system( string_to_execute )

string_to_execute = "/bin/su " + user_name + " -c \'/usr/bin/ssh-keygen -b 512 -t dsa -N
\'\' -f " + os.path.expanduser( "~" + user_name ) + ".ssh/id_dsa\'"
os.system( string_to_execute )

string_to_execute = "cp /home/" + user_name + ".ssh/id_dsa.pub /home/" + user_name +
"/.ssh/authorized_keys2"
os.system( string_to_execute )

string_to_execute = "/opt/c3-3/cexec chown -R " + user_name + ":" + user_group + " /home/"
+ user_name
os.system( string_to_execute )

```

Figure 1