

C3 POWER TOOLS

The Next Generation...

Brian Luethke, Thomas Naughton, and Stephen L. Scott

Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37830-6367, USA

Abstract: The 3rd generation of the Cluster Command and Control (C3) Power Tools is an even more powerful and secure command line cluster interface than its predecessors. Furthermore, C3 continues to extend the *Single System Illusion* (SSi) concept from single clusters to federated clusters (multiple clusters viewed as one computing entity) by providing secure remote access to the C3 Power Tools for both users and administrators in a manner that is as easy to use and as transparent as command line operations issued on a single workstation. This paper will discuss the evolution of C3 and expand on its new capabilities.

Key words: tools, administration, federated clusters, grid

1. INTRODUCTION

There have been quite a few advances by the C3 Power Tools (Cluster Command and Control) since they were first presented at DAPSYS 2000 [1]. Since then, C3 has evolved from a simple command line tool with the ability to operate in an iterative manner across a single cluster of externally exposed nodes to a robust stable tool set that is capable of securely coordinating simultaneous parallel operations across multiple clusters scaling to the Grid realm. Version 3.0 has been used extensively inside Oak Ridge National Laboratory for approximately a year and has been publicly available for download since August 2001. Version 3.1 is in final testing with release anticipated in Summer 2002. This generation of the C3 Power Tools includes many features that have greatly assisted the ORNL Network and Cluster Computing Group in the day-to-day operation of the over 200+

processors in the research clusters of TORC, HighTORC, and eXtremeTORC. The tools are also used to make quick work of the building and administrating of the many ad hoc research clusters at ORNL. C3 is also a core component of the Open Source Cluster Resource Application Resources (OSCAR)[2] providing installation configuration services and administration support. The C3 Power Tools are designed to easily move information into and out of the cluster as if it were a single machine rather than a group of loosely coupled independent machines. Thus, C3 embodies the *Single System Illusion* (SSi) concept of providing the *illusion* of a single system. As a result, the C3 tools will also operate on workstation farms or labs of independent machines that are not configured as a cluster. From a single access point, the tools may be used to invoke any command in parallel across a group of connected machines. All but one module of version 3.x was rewritten in Python – prior versions were completely in Perl. The main reason for the switch was that the codes became unmanageable in Perl due to their size. The design of C3 leverages a number of freely available tools including: Python, Perl, rsync, ssh, rsh, SystemImager, and DHCP. The remainder of this paper will consist of a brief introduction of the C3 commands with the greater portion dedicated to the discussion of the latest capabilities of the version 3.x release.

2. OVERVIEW

The C3 tools suite consists of ten general use tools: `cpushimage`, `cshutdown`, `cpush`, `crm`, `cget`, `cexec`, `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. For those familiar with earlier versions of C3, the `cl-ps` or `cps` command was removed from v3.0. A brief description of each core C3 tools follows – detailed information may be found on the web site and in the man pages [3]. The `cexec` command is the C3 general utility tool as it may execute any command on each cluster node. A command string passed to `cexec` is executed “as is” on each node. All C3 commands are executed in parallel, however, `cexec` is the only tool that also has a serial operation mode. This is to assist debugging complex problems where deterministic behavior is desired. 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 `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. For general users, `cpush` is the tool to copy files or directories or for root use when `cpushimage` is too heavy-handed. `crm` is a clusterized version of the standard `'rm'` delete file/directory command. `cname` assumes that you know a single node name and want to know its position. `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 cluster type.

3. VERSION 3

The 3.x architecture improves on previous versions in the following areas: 1) better management of multiple clusters – in particular, those clusters in multiple administrative domains with computation nodes on a private network – not exposed to direct outside access; 2) enhanced API with support for user specified node ranges; and 3) Python implementation of core code.

C3 enlists the use of three abstract cluster classifications *direct-local*, *direct-remote*, and *indirect-remote* to extend the SSI concept to multiple clusters. Older versions of C3 had only one cluster classification and to reach multiple clusters, all compute nodes had to be exposed to the outside network. This was not practical and also opened a large security hole. This is not the case when using the new cluster abstractions. With these, it is possible to reach multiple clusters where the individual nodes are on a private subnet, sitting behind a dual-ported head node with one port connected to the outside world and the other connected to the internal private network. These abstractions also simplify the discussion of cluster configurations and architectures.

C3 commands identify their compute nodes with the help of cluster configuration files. These configuration files name a set of accessible clusters and describe the nodes. The default cluster configuration file, `/etc/c3.conf`, consists of a list of cluster descriptor blocks. These syntactic objects name and describe a single cluster that is accessible to the system's users. The following code is a configuration file for a 66-node cluster. This cluster is a direct-local cluster where all nodes are known at runtime and commands are issued from the head cluster node.

```

cluster local {
    xtorc0:node0          #head node named
                        #xtorc0 external, node0 internal
    node[1-64]           #compute nodes
    exclude 10           #node10 is offline
    exclude [60-64]     #node60 through node64 are offline
    node78               #node78 is not inside a range
    dead node100        #node 100 is offline
}

```

In the above example, the tag `cluster` is followed by label `local` and the open curly brace indicates the beginning of the cluster description block – at the end is a closing curly brace. The label `local` is used to reference this cluster. The first line in a cluster description block represents the head node. The head node may be represented by any of hostname or external interface IP address followed by the colon and then the internal hostname or IP address. Just the hostname or IP address is needed for clusters that have only one interface on the head node.

Next in the configuration block is the node descriptors. They may be specified in three ways: 1) ranges, as in line `node[1-64]` indicating 64 nodes and 2) individual node name as in line `node78`, and 3) by IP address both individually and with ranges. When specifying by IP address, ranges are only valid for the last set of digits – `134.167.12.[2-12]`. As the position a node occupies in the configuration file is significant, nodes that are “offline” must still occupy their position. Nodes not participating and within a range must use the `exclude` qualifier to indicate offline. Only ranges are valid with the `exclude` and must immediately follow the range declaration where applied. Note that `exclude5` would parse as a node name `exclude5` and is not an `exclude` operation – it would indicate an active node named `exclude5`. Out of sequence node names are listed one per line like `node78` and `node100` above. The `dead` qualifier is used to indicate that an out of sequence node is offline as in `dead node100`.

Federated clusters are constructed as a list of cluster descriptor blocks – one per cluster. The first cluster’s descriptor block has a special importance that is analogous to the special significance accorded the first declaration in a *Makefile*. Any instance of a C3 command that fails to name a cluster will by default run on the first cluster in the configuration file.

The remote cluster is another classification used by the C3 cluster abstraction. This means that the command is issued on a machine outside the physical scope of command’s target cluster. This enables commands to be issued, for example, from an administrator or scientist’s workstation not part of the cluster. There are two types of remote clusters – direct and indirect. Direct-remote clusters are when the command is issued from an outside machine that has full knowledge of the cluster’s configuration by having its

own local copy of the cluster configuration file. The advantage of the direct-remote cluster is that subsets or supersets of a remote cluster may be constructed from both the command line and a local script. The disadvantage is that this configuration file must be consistent with that on the cluster head node. The indirect-remote model was developed to eliminate the need to maintain state between the local workstation and remote cluster configuration files. Here only the external interface of a cluster is known prior to runtime. Here the local C3 command tells the cluster to run based on the configuration in the cluster's own local configuration file. This works very well where the user simply wants to affect all available cluster nodes. Placing a colon followed by the cluster's head node external hostname in the head node location of the local cluster configuration block specifies the local configuration file. An indirect-local cluster configuration file for `xtorc0` would look like the following:

```
cluster local {
    :xtorc0
}
```

4. BASIC C3

Node positions can be specified in two ways, one as a range, and the other as a single node. Ranges are specified by the following format: $M-N$, where M is a positive integer (including zero) and N is a number larger than M . Single positions are just the position number. If discontinuous ranges are needed, a comma must separate each range. The range "0-5, 9, 11" would execute on positions 0,1,2,3,4,5,9,11.

C3 uses machine definitions to specify clusters and ranges on the command line - there are four options. First, none specified, this results in execution on all nodes of the default cluster. An example of this is:

```
cexec ls -l
```

Second is a range on the default cluster - form `<:range>`. To execute `ls` on nodes 1,2,3,4,6 of the default cluster:

```
cexec :1-4,6 ls -l
```

Third method is specifying a specific cluster - form `<cluster_name:>`. To execute `ls` on every node in cluster `test`:

```
cexec test: ls -l
```

Fourth is specifying a range on a specific cluster - form `<cluster_name:range>`. Execute `ls` on cluster `test`, nodes 2,3,4,10:

```
cexec test: 2-4,10 ls -l
```

Additionally, these four methods can be mixed on a single command line. To execute `ls` on nodes 0,1,2,3,4 of the default cluster, all of `htorc1` and 1,2,3,4,5 of `htorc2`:

```
cexec :0-4 htorc1: htorc2:1-5 ls -l
```

5. ADVANCED C3

C3 commands may also be used in shell scripts just like any other standard command. In doing so, users and administrators may automate advanced tasks. One such example is that of scripted software installations across all cluster nodes. With the C3 tools, administrators have many options as how to complete this task. One method is to update one node, create a new system image from the updated node, and then push the image across the cluster using `cpushimage`. However, if the software installation is a small task, administrators may not want to go to the trouble of generating a new image and pushing it across the cluster. Instead of pushing a complete image, the administrator may use a script such as that below executed on the server to accomplish the task. The administrator is installing a package from a tarball by pushing the tarball to all nodes using `cpush`, unpacking the tarball on all nodes using `cexec`, and then running the commands necessary to build and install the package using `cexec`. Alternatively, the administrator could have just created a script to install the package on a local machine, pushed the install script and tarball to all nodes, and then run the install script on all nodes using `cexec`. This is the method used for some of the software installation tasks in OSCAR.

```
#!/bin/sh

#copy over package tarball
/opt/c3-3/cpush /root/software/mpich.tar.gz /tmp

#unpack tarball (creates mpich subdirectory )
/opt/c3-3/cexec tar -zxf /tmp/mpich.tar.gz

#build & install
/opt/c3-3/cexec cd /tmp/mpich; ./configure --prefix=/usr/local/mpi
/opt/c3-3/cexec cd /tmp/mpich; make; make install
```

Another difficult task is adding a user to a cluster. The password files and group files may need to be distributed across the cluster along with the creation of the appropriate home directories. This task becomes even more

difficult when dealing with federated clusters. It would require that an administrator repeat each task without variance for each cluster. Below is a scripted example of C3 doing this job.

```
#!/bin/sh

#First add the user to each head node in the cluster:
cexec -head htorc: torc: useradd -g users -m -s /bin/tcsh -u 512 eff

#next add the user to each compute node in the clusters:
cexec htorc: torc: useradd -g users -m -s /bin/tcsh -u 512 eff

#change the users password on one machine (use of cexecs allows
#execution of interactive commands, you could #also have used
#ssh or rsh)
cexecs -head torc: passwd eff

#distribute password files to head nodes
cexec -head torc: cpush -head htorc: torc: /etc/passwd
cexec -head torc: cpush -head htorc: torc: /etc/shadow

#distribute password files to compute nodes
cexec -head torc: cpush htorc: torc: /etc/passwd
cexec -head torc: cpush htorc: torc: /etc/shadow
```

One of the more powerful uses of specifying subclusters from the command line is trying new software out on a small number of machines, (cluster partition), otherwise known as rolling upgrades. The following example installs a new version of MPICH on a three-node partition using range specifiers :0-2 to reduce command scopt.

```
#!/bin/sh

#copy over package tarball
/opt/c3-3/cpush :0-2 /root/software/mpich.tar.gz /tmp

#unpack tarball (creates mpich subdirectory )
/opt/c3-3/cexec :0-2 tar -zxf /tmp/mpich.tar.gz

#build & install
/opt/c3-3/cexec :0-2 cd /tmp/mpich; ./configure -prefix=/usr/local/mpi
/opt/c3-3/cexec :0-2 cd /tmp/mpich; make; make install
```

Scripting the C3 tools can also be advantageous for general users. A common user task is to run a parallel job and gather the resulting output files generated on all nodes. The following script shows a user pushing out an application, executing the application, and gathering the results to their local workstation.

```
#!/bin/sh

uid = `is -u`
user=`id -un`
dir="/tmp/myapp.$uid"
app="/home/$user/apps/myapp/hello"
results="/home/$user/apps/myapp/results"

#create temporary directory
/opt/c3-3/cexec mkdir $dir

#copy over application binary (hello)
/opt/c3-3/cpush $app $dir

#run hello (which creates a hello.out on each node)
/usr/local/mpi/bin/mpirun -np 64 $dir/hello

#collect output files
/opt/c3-3/cget $dir/hello.out $results

#remove temporary directory & contents
/opt/c3-3/crm -r -f $dir
```

6. PARTING REMARKS

C3 is presently a robust and stable tool that is very useful for cluster users and administrators. While there are no plans to change the exposed portion of C3 including the API and core commands, the team is working on internals in an effort to increase the scalability and performance of C3 as well as the underlying tools used to implement C3. The team is also working on posting to the C3 web site – the many useful scripts and tools built using C3.

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.

REFERENCES

- [1] “Cluster Command & Control (C3) Tools Suite,” R. Flanery, G.A. Geist, B. Luethke, and S. Scott), 3rd Distributed and Parallel Systems (DAPSYS 2000), September 10-13, 2000, Balatonfüred, Lake Balaton, Hungary.
- [2] OSCAR – Open Source Cluster Application Resources, <http://oscar.sourceforge.net/>
- [3] C3 – Cluster Command and Control, <http://www.csm.ornl.gov/torc/C3/>