

Scalable C3 Power Tools

Brian Luethke and Stephen L. Scott¹
Network and Cluster Computing Group.
Oak Ridge National Laboratory, USA.
{luethkeb / scottssl@ornl.gov}

Abstract

With the growth of the typical cluster reaching 512 and more compute nodes, it is apparent that cluster tools must begin to reach toward the 1000's-of-nodes in scalability. There are a number of problems that must be considered when attempting to create a scalable version of existing cluster tools. For a tool to successfully scale, it must maintain relative performance without sacrificing reliability and ease of use. The version 4.0 release of the C3[1][2] tools has started stretching the Single System Illusion concept into the realm of 1000's of compute nodes by actually improving performance on larger clusters. This paper is a discussion of how this was implemented, how to use this new version of C3, and presents some results comparing the latest release with prior versions of C3.

1. Introduction

In 1999 a project was initiated at Oak Ridge National Laboratory (ORNL) to develop a set of tools that would facilitate the use and administration of clusters in a *Single System illusion* (SSi) style such that a single command could run across numerous machines. This work resulted in the Cluster Command and Control (C3) Power Tools. Version 1.0, completed in 2000, was a proof of concept implementation for in-house use only. While using version 1.0 at ORNL, it was discovered that the functionality was useful, however that serial execution of commands across the cluster nodes was quite slow. Version 2.0 was the first multi-threaded release. Unfortunately, due to a bug in Perl's[3] thread package v2.0 was not released to the public. Version 2.6 was the first public release of the C3 tools. This version used a multi-process method for execution that greatly increased the scalability over version 1.0. At this time, the C3 implementation grew to a size that made it difficult to maintain in Perl. Thus the decision was made to rewrite the entire C3 suite in Python[4]. The result of this was version 3.0, released to the public on August 15, 2001. This version completely updated the command line syntax and contained an expanded multi-cluster capability. However, this version still used the single layer multi-process

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.

fan-out tree used in version 2.6. The most recent release, version 4.0, is the first scalability release. This version retains the full functionality of the prior release and adds the ability to scale the C3 environment to clusters in order of 4096 nodes. This paper is a discussion of the new scalability features of C3 version 4.0 – *the scalability release*.

2. Basic C3 Operation

C3 recognizes three modes of cluster use. The first mode is *direct local*. Direct refers to the command being invoked from the cluster head-node. Local means that the invoking machine (head-node in this case) has knowledge of all nodes that exist in the cluster. Thus, direct local means that the command is invoked from the cluster head-node and the head-node has knowledge of all nodes that exist in the cluster via a local C3 configuration file. The second mode is *direct remote*. Remote in this case means that the machine from which the command is invoked is not the head-node. Thus direct remote means the command is invoked from a machine other than the head-node and because it is direct and the invoking machine has knowledge of the targeted compute nodes. The third mode is *indirect remote*. Indirect refers to the command being invoked from a machine that does not have knowledge of all nodes within the cluster. In this case, the invoking machine is not the cluster head-node and only knows of the cluster's existence and not anything of the individual nodes within the cluster. This is the typical case for off-cluster access where the user's desktop machine only has a reference to the cluster (no individual node information) and the physical cluster configuration information is maintained on the respective cluster's head-node. During execution, C3 will confirm that the local machine's hostname is that listed as the head-node of the cluster in the C3 configuration file. Therefore, either a DNS lookup or a valid hosts file is required. An important note is that the *indirect local* cluster mode is not considered a valid mode, as it will cause an infinite loop at execution.

2.1 Configuration File

The more important aspects of C3 to understand are its configuration file format. Figure 1 will be used for discussion regarding the standard configuration file. This configuration file contains four cluster definition blocks; each block follows the same syntax rules. The first tag "cluster" is required and specifies that a cluster definition block is beginning. This is followed by the cluster name (any alphanumeric name is acceptable). Each cluster name must be unique, although it is up to the system administrator to guarantee this attribute. Next is the "{" character that delineates the start of the cluster definition. While the cluster name only has meaning within the context of C3, the names given for the nodes must be proper addresses for accessing those interfaces (that is: the names must be resolvable, either through DNS, a proper hosts file, or the IP address of the machine). The first machine in the list is of special significance as this is the head-node of the cluster. When a remote cluster is specified this is the node that

the cluster will be accessed through. The head-node declaration has three forms, in clusters *first* and *feanor* the cluster is on a private network and the head-node has an external interface. In cluster *first* it is easy to see that the first name on the line is the external interface, followed by a colon, then the internal interface name. On *feanor* the hostname *feanor* is the external address and *node0* is the internal interface. In cluster *second* there is only a single name on the head-node line. In this case it is assumed that the external and internal interface is the same. Cluster *torc* is an example of an indirect cluster. This is denoted by the first character on the head-node line being a ":" followed by the hostname of the remote clusters head-node.

Next is the compute node block. Nodes can be specified in two ways. The first method is line by line, one node per line. Cluster *first* uses this example. Each line contains one, and only one, node. To mark a node specified by the singular method as offline, use the "dead" tag. The second method is the use of node ranges, cluster *second* and *feanor* use these. Here, the node declaration begins with the part of the name that is common among all the nodes then followed by a range, enclosed by the "[]" characters. The range must be at the end of the name (if using IP addresses you would do 192.168.1.[1-64]), only one range per line at the end of the name. When dealing with nodes specified via ranges a single node may be marked offline by using the "exclude" tag followed by a space and then the node number. To mark a range of nodes offline, use the "exclude" tag again followed by a space with the range of nodes offline enclosed with the "[]" characters. Multiple exclude tags may be listed, however they only apply to the range statement preceding them. An example of this is in *feanor*: where two non-contiguous node ranges are excluded from the first range in two separate lines. It is very important to note that the words "exclude" and "dead" are not reserved words and are only treated as an offline specifier when syntactically legal to do so. For example, figure 2 uses valid, but not recommended node names.

2.2 Command Line Interface

Beginning with version 3.0 of C3 an effort was made to keep the command line interface (CLI) as close to the standard Linux command as possible. Nearly every option that the Linux command would have is supported under C3; the only time an option was omitted or changed was when it was not applicable to a cluster.

The C3 CLI standard format is as follows:

```
command [options] [machine_definitions] other [other...]
```

"Command" is simply which C3 command is being executed (cexec, cpush, etc...). "Options" are the options being passed to the command, a single dash (-) followed by an alphanumeric character or a double dash (--) followed by

multiple characters. “Machine_definitions” is where the clusters and ranges you wish the command to be executed on are listed. “other” is whatever is required for that specific command. For the examples given below refer to the first configuration file in this paper. Machine definitions follow this format:

```
username@clustername:range
```

username@ is used if your remote account’s login is not the same as your local account. This can be specified for each cluster individually. If this is omitted then the default username is used (you may set an environment variable or the local username will be used). The cluster name is the name you gave the cluster in the configuration file following the cluster tag. Range is given as either multiple nodes in one block (beginning-end) or a single number (number) separated by a comma. Here is an example cexec command.

```
cexec --pipe : zbml1@second: zbml1@feanor:1-5,10
torc: hostname
```

In the above example, the option --pipe is used to tell the cexec command to format the output in a pipe friendly format. Next is a “:” by itself, this represents the default cluster, the first cluster listed in the configuration file. The “:” is simply a notational convenience so the user doesn’t have to type out the full name of the default cluster every time. All of the different formats in the machine_definitions can work this way also: zbml1@:, “:1-3,5”, “zbml1@:1-23” are all valid. Specified next is the cluster named *second* with the alternate username “zbml1”. Following this is the cluster *feanor*, alternate username zbml1, and execute only on nodes 1, 2, 3, 4, 5, 10. Lastly, execute on *torc* with the default username and execute on all nodes on the cluster. The command cexec is executing here is “hostname”.

One important note about using ranges is that they refer to the nodes position in the configuration file – similar to the enumerated type in programming languages. For this reason it is important to mark nodes offline instead of simply removing them from the configuration file. It is also easier to quickly ascertain which nodes are down when they are explicitly entered as offline. Node positions begin counting at 0. Thus, node position 1 is the second node in the list. The head-node declaration does not count as a node position. The zero based numbering scheme may cause number skewing if it is desired to start node counts 1. However, it is easily resolved by using a placeholder at the beginning of the node definition block. For example:

```
cluster example1 { # no placeholder, node1 is in position 0
    external:node0
    node[1-64]
}
```

```
cluster example2 { # placeholder added, node1 is in position 1
    outside:node0
    dead placeholder #dummy entry to change to 1 based indexing
    node[1-64]
}
```

The following `cpush` will push `/etc/passwd` to the first node in both of the above specified clusters:

```
cpush example1:0 example2:1 /etc/passwd
```

Three commands have been added to C3 to help manage the use of multiple clusters and node ranges. First is `clist`: this command lists every cluster in the configuration file and their type (direct local, indirect remote, direct remote). Second is `cname`, it takes a node name and searches the configuration file list returning the position number of that node. Third is `cnum`, it takes a number range and returns the node names those positions represent.

3. Scalable C3

Using C3 in its scalable execution model is very similar to using C3 in the standard non-scalable fashion. Both the command line interface and the configuration file have the same syntax; only the semantics of the options has changed. The use of C3 in its non-scalable execution model is capable of controlling multiple clusters in a single file. At this time when using the scalable execution model C3 is only capable of interfacing with a single cluster. However, one may use the indirect remote model from off the cluster and then use the scalable release on the target cluster(s). This will allow one to interact with multiple large clusters simultaneously.

While from a conceptual point of view it is possible to set up a cluster in this fashion in any 3.x version of C3, it is important that version 4.0 or greater be used to achieve the scalable execution model. This is because in order to implement the scalable execution model many of C3's internal algorithms had to be parallelized. If using versions prior to version 4.0, nearly all of the command will run serially rather than in parallel.

3.1 Configuration File

While the configuration file for the scalable execution model is syntactically identical to the above, the meaning of the positions have changed somewhat. The basic concept is that a single large cluster may easily be viewed as several "logical" smaller clusters. The first decision a system administrator needs to make is how to logically partition the cluster. Ideally the cluster should be partitioned as close to a perfect square as possible, in our example a 64-node

cluster is divided into 8 8-way clusters. C3 easily handles up to 64-way clusters resulting in an estimated top end of 4096 nodes (64 64-way cluster) with only one level of indirection.

Using the same configuration file syntax as in the non-scalable version one can easily create a 64-node scalable cluster configuration as in figure 3. Graphically this would appear as in figure 4.

As you can see, the cluster block in figure 3 is syntactically the same as the cluster configuration file presented in the non-scalable section of figure 1. The difference is that here only one cluster is defined per configuration file and not just one cluster per block as was done in the standard release. In the scalability release, when a command is executed, it is "staged", or sent to the specified staging node, represented by the head-node line, to be executed on the block of compute nodes that the staging node is responsible for, listed in the compute node block. In the above example each chunk of the cluster is defined as a direct remote cluster and it is required that the commands be executed from the head-node of the cluster when using the scalable execution model. One may alternatively define each as an indirect cluster, but then each staging node must have local to that system, the list of compute nodes it is responsible for. Using direct remote staging nodes is slightly slower but is much easier to maintain, while the indirect is faster but more difficult to maintain. Figure 5 is an example of the indirect staging nodes.

One important decision to make is whether the staging node is included within its own list of responsible nodes. C3 will either execute on the compute nodes or the head-node, not both at the same time. Thus, in the above example, it will take two commands to run on every node as C3 will only execute the given command on the list of compute nodes that it holds responsibility for and not itself. This semantics has the convenience of separation of responsibilities but requires more commands for a full cluster execution. However, if a staging node lists itself in their responsibilities list, then a single command will run on the entire cluster. Figure 6 is an example of the staging node adding itself to its list of responsibilities.

This choice partially depends on how the system administrator sets up their cluster. If several nodes are dedicated as staging nodes then you will not want them to participate as a compute node. This will typically give the fastest results, as the only loads on those nodes will be system services and never a user's application. It is important to note that C3 permits alternate configuration files to be specified. The latest release provides the capability to set a per user default configuration file using environment variables.

Physical layout of the cluster may impact the ordering of the nodes. Ideally each cluster block would be within a single switch, including the head node of that

cluster block. This is because intra-switch communication is very fast. For example, if each switch only contains 32 ports then each block should not contain more than 1 staging node and 31 compute nodes – with each of those nodes physically connected to the same switch. It is not as important that the staging nodes all be connected to the same switch as the head node used to initiate the C3 command, though it would also speed execution. It is not recommended that each node service more than 64 nodes unless absolutely necessary.

3.2 Scalable Command Line

The choice a system administrator makes about whether the staging node includes itself in its responsibilities list determines to some extent how the command line is used. Most of these examples will assume that the staging node includes itself in the responsibilities list. In the case where it is not included it would require an extra command so that the execution will also take place on the staging nodes. There will be a few examples of this behavior noted also.

The most important option common throughout the C3 commands is the `--all` option. This option tells C3 to execute the given command on each and every cluster and node in the configuration file. It is recommended that the most commonly used or perhaps all of the C3 commands be aliased to “command --all” for convenience. The following command will push `/etc/passwd` to every node in the scalable cluster:

```
cpush --all /etc/passwd
```

The other option is to explicitly list each sub-cluster on the command line. However, for large clusters this would be quite cumbersome if not impossible to do without human error from the keyboard.

At this point using ranges on a scalable cluster is not as clean as using them on a non-scalable cluster. Because “node48” could be in any of the sub-clusters, or “node35-64” may cross several sub-cluster boundaries. These must be explicitly searched on and explicitly stated on the command line. For example searching in the first scalable configuration file:

```
cname --all node48 node35 node64
```

will return node 48 in part6 position 7, node35 in part5 position 1, node64 in part8 position 8.

To execute on node48 use:

```
cexec part6:7 hostname
```

To execute on 35-64 use:

```
cexec part5:1-7 part6: part7: part8: hostname
```

In cases where the system administrator may want to only execute on the non-staging nodes, for example to read logs. Here, having a configuration file where each staging node only lists itself in its list of responsibilities would be difficult. You would have to explicitly list each sub-cluster with a range excluding position 0, assuming the staging node is the first node in the list. For this reason it is recommended that the administrator keep two versions of the configuration file – one with and one without the staging node listed in its own list of responsibilities. It is important to note that multiple configuration files are allowed and only change the logical view of the cluster to the C3 command.

Here is an example of pushing a /etc/passwd to the entire system using a configuration file where the staging nodes' node is not include within their list of responsibilities:

```
cpush --all --head /etc/passwd  
cpush --all /etc/passwd
```

The first cpush with --head will push the passwd file to the head node and the second cpush command line will push to the nodes in the staging node's responsibility list.

4. Miscellaneous

Two commands currently do not gain any benefit from the scalable execution model. Cpushimage will not function properly – specifically it will fail. Currently systemimager[5], used by cpushimage, does not support staging of images on other machines. However, is possible to create a staging node image that contains the standard nodes' image and then push that image to only to the staging nodes. Next, tell each of the staging nodes to propagate that image to the nodes in their list of responsibilities. This process will result in all nodes receiving their image. While this gains the parallelism of the scalable execution model it takes a significant amount of disk space and is difficult to maintain. The server must maintain three complete images; it's own operating image, a staging nodes image, and a compute node's image. These images must be maintained correctly or the entire cluster may have system errors as a result of a cpushimage. For example: to get the first set of images:

1. build compute nodes.
2. build staging nodes.
3. take compute node image on staging nodes.
4. take staging node image on head-node.

This must be repeated for every change required in the stable image. Next, to propagate an image out to the cluster will require:

```
cpushimage --all --head staging_image
```

This will update all the staging nodes with the image “staging_image” stored on the local machine. Next, to push “node_image” too all the nodes in their respective staging node responsibilities list:

```
cpushimage --all node_image
```

Here, it is extremely important that the staging nodes are not included in their lists of responsibilities. The effect of cpushimage will be random if the staging node includes itself in its list of responsibilities. The result will depend on when the process was started and when the image update is applied to the image storage area. the new update will delete the information being synchronized during the rsync operation.

The other command that will not benefit from the scalable execution model is cget. Cget will still function properly, however it will probably not gain much parallelism, as it is a gather operation with many machines communicating to a single machine.

5. Performance Analysis

The tests presented here were run on eXtreme TORC, ORNL’s 64 node cluster. The operating system was a stock RedHat 7.3[6] system with OSCAR[7][8] version 2.1 installed for clustering services. Each node is a 2.0 GHz pentium4 with 768 MB of RAM. The tests were run over an Intel Ethernet pro 100 Mbit Ethernet adapter with a Foundry FastIron II switch.

The cluster was divided into four sub-clusters with each staging node containing 16-nodes in its list of responsibilities. Each staging node contains a self-reference in its configuration block. Both wall-clock time required for the command as well as server load was recorded. Four files were pushed with cpush, a 1, 10, 100, and a 1000 Megabyte file. The scalable runs are divided into three categories. *Staging* is moving the file from the head-node of the cluster to the staging nodes. *Fan-out* is the time for a staging node to transfer the file to their list of responsibilities. *Scalable total* is total the time taken. Load average is not relevant here since there is no recorded load average for the compute nodes. The last line is provided for a comparison to the non-scalable execution mode of C3.

1 Megabyte

	Total Time Taken	Load Average head
Staging	2.36s	n/a*
Fan-out	14.35s	.80
Scalable Total	16.71s	not relevant
Non-Scalable	31.70s	10.57

* Insufficient time was required to result in a change of load for the head node.

10 megabyte

	Total Time Taken	Load Average
Staging	12.43s	1.27
Fan-out	44.98s	.80
Scalable Total	57.51s	not relevant
Non-Scalable	2m 52.99s	48.03

100 megabyte

	Total Time Taken	Load Average
Staging	1m 54.06s	3.30
Fan-out	5m 57.21s	.80
Scalable Total	7m 51.27s	not relevant
Non-Scalable	27m 00.67s	55.48

1000 megabyte

	Total Time Taken	Load Average
Staging	20m 06.72s	16.48
Fan-out	59m 36.34s	.80
Scalable Total	1h 19m 43.06s	not relevant
Non-Scalable	4h 28m 27.90s	55.51

From these results, it is clear that the change to the scalable execution model results in a significant speedup when compared to the non-scalable execution model. While this result itself is rather significant, another significant effect of this model is the relative “unloading” of the cluster’s head node. When transferring large files using the non-scalable execution model, the head-node is nearly unusable during the run. This is so disruptive that other users cannot even log onto the head node during a file transfer. When using the scalable version of C3, the machines performance was impacted, however the cluster remained a usable resource.

6. Conclusion

This paper discussed the first scalability release of the C3 Power Tools (version 4.0) from Oak Ridge National Laboratory. Presented material included a brief introduction to the operation of standard (version 3.x release) C3 commands,

their command line interface (CLI), as well as the associated configuration file syntax and semantics. This was followed by an introduction to the latest C3 release (version 4.0 also known as the scalability release), which emphasizes a new scalable component that enables the C3 Power Tools to scale its operation to 1000's of cluster nodes. The scalability release's CLI syntax was presented as well as the requisite changes and new semantics of the configuration file. Finally, a number of tests were run across ORNL cluster computing facilities to provide real-world numbers that show the performance improvement and scalability of this new C3 release.

While we are pleased with the improvements that this release brings, plans are in the works to continue in this direction to provide an even more scalable and faster C3 Power Tool suite. It is anticipated that the first iteration of this next release (version 5.0) will arrive later this year.

7. References

- [1] "C3 Power Tools," (with B. Luethke), Commodity, High-Performance Cluster Computing Technologies and Applications, The Sixth World Multiconference on Systemics, Cybernetics, and Informatics (ISAS SCI2002), July 16, 2002, Orlando, FL.
- [2] "C3", <http://www.csm.ornl.gov/torc/C3/>
- [3] "PERL", <http://www.perl.org/>
- [4] "Python", <http://www.python.org/>
- [5] "SystemImager", <http://www.systemimager.org/>
- [6] "Redhat", <http://www.redhat.com/>
- [7] 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
- [8] "OSCAR", <http://www.OpenClusterGroup.org/OSCAR>

```

cluster first { # The default cluster (first in the list) and a
                # direct local cluster
    external:internal # the head-node
    node1             # compute nodes
    node2
    dead node3       # offline node w/o node ranges
}
cluster second { # direct remote - notice that unlike cluster
                 # first the head node is not the local machine
    xtorc
    node1
    node2
    dead node3
}
cluster feanor { # direct remote cluster
    feanor:node0 #example of node ranges
    node[1-64]
    exclude 4    # setting nodes offline in a range
    exclude [32-37] # node can be marked offline in
                  # ranges also
    node[128-132]
}
cluster torc { # indirect remote cluster
    :torc       # external interface of remote
cluster
}

```

FIGURE 1: sample configuration file

```

cluster don't_use { # notice below that what separates dead
                   # and exclude from being hostname is that
                   # in the case of "dead" there are two
                   # words on a single line. In the case of
                   # "exclude" there is a space between the
                   # word exclude and the range. While this
                   # is possible it is not recommended.
    external:internal # head-node
    dead              # first node named "dead"
    dead dead1       # second node "dead1" offline
    exclude[1-64]    # 64 nodes excludel..exclude64
    exclude [5-6]    # exclude5 and exclude6 are
                   # offline
    exclude 8        # exclude8 is offline
}

```

FIGURE 2: configuration file with poor naming technique

```
cluster part1 {
    node1
    node[2-8]
}
cluster part2 {
    node9
    node[10-16]
}
cluster part3 {
    node17
    node[18-24]
}
cluster part4 {
    node25
    node[26-32]
}
cluster part5 {
    node33
    node[34-40]
}
cluster part6 {
    node41
    node[42-48]
}
cluster part7 {
    node49
    node[50-56]
}
cluster part8 {
    node57
    node[58-64]
}
```

FIGURE 3: 8 8-way configuration file

8-way

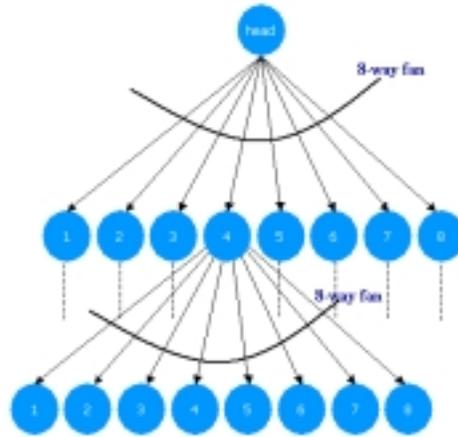


FIGURE 4: 8 8-way for total of 64 nodes

```
cluster part1 {
    :node1
}
cluster part2 {
    :node9
}
.
.
.
cluster part8 {
    :node57
}
```

and node1 would have the following specification while the others would also follow suite.

```
cluster part1 {
    node1
    node[2-8]
}
```

FIGURE 5: indirect staging nodes

```
cluster part1 {  
    node1  
    node[1-8]  
}  
cluster part2 {  
    node9  
    node[9-16]  
}  
.  
.  
.  
cluster part8 {  
    node57  
    node[57-64]  
}
```

FIGURE 6: staging nodes, listing self