3rd Distributed and Parallel Systems (DAPSYS 2000), September 10-13, 2000, Balatonfüred, Lake Balaton, Hungary. Published by: Kluwer Academic Publishers, ISBN: 0-7923-7892-X

Cluster Command & Control (C3) Tools Suite

Ray Flanery[1], Al Geist[1], Brian Luethke[2], and Stephen L. Scott[1]

[1] *Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37830-6367, USA.*
contact author: scottsl@ornl.gov

[2] *Computer Science Department, East Tennessee State University, Johnson City, Tennessee 37614-0718*

Abstract The computation power of PC clusters running the Linux operating system rivals that of supercomputers of just a few years ago at a fraction of the purchase price. However, the lack of good administration and application tools represents a hidden operation cost that is often overlooked when proposing a cluster. This paper presents a number of command line tools developed at Oak Ridge National Laboratory for use in operating the HighTORC cluster. These same tools provide the "backend" connection to the cluster for our web based GUI tool suite M3C (Monitoring and Managing Multiple Clusters).

Keywords: PC cluster tools, cluster administration

1. INTRODUCTION

While there are numerous tools and techniques available for the administration of clusters, few of these tools ever see the outside of their developer's cluster. Basically they are developed for specific inhouse uses. This results in a great deal of duplicated effort among cluster administrators and software developers. Thus, after building HighTORC[1], a 64-node – 128 processor – Beowulf cluster in the Summer of 1999, it was decided to make an effort to develop and collect a suite of system administration cluster tools that could be released to the cluster community so that others may benefit from our effort as well.

To simplify administration some cluster builders simply NFS mount one master file system to all nodes. While the "one file system" approach does greatly simplify configuration management and application development, it also provides the least scalable model and lowest performance for cluster computing due to machine and network constraints. Our approach is to decentralize every aspect of the cluster configuration in HighTORC to promote scalability. This resulted in every node hosting its own independent operating system with the remaining local disk space used as temporary application and data storage. Of course, this approach generated the need for tools to hide the fact that HighTORC consists of 64 independent machines. Using a number of freely available tools; rsync[2], tomsrtbt[3], ssh2[4], DHCP[5], and Systemimager[6] we developed a suite of cluster command and control (C3) tools 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. These tools are similar in function to those developed by Gropp and Lusk [7] for use on MPPs.

One of the main criteria of the tools is that, they provide the look and feel of commands issued to a single machine. This is accomplished through using lists, or configuration files, to represent the group of machines on which a command will operate. While this requires the building of machine lists, or cluster node lists, it still presents the typical user with a single machine interface. This occurs as generally a cluster is configured and used in its entirety at any given instance by a single user. Thus the cluster node list containing all cluster nodes may be built once and then forgotten. However, a strength of the C3 tool suite is that an individual user may dynamically customize and use their cluster configuration list without affecting any other cluster users.

A second criterion is that the tools be secure. Generally security inside a cluster, between cluster nodes, is somewhat relaxed for a number of practical reasons. Some of these include 1) improve performance, 2) ease programming, and 3) all nodes are generally compromised if one cluster node's security is compromised. Thus, security from outside the cluster into the cluster is of utmost concern. Therefore, user authentication from outside to inside must be done in a secure manner.

The third criterion is tool scalability. A user may tolerate an inefficient tool that takes a few minutes to perform an operation across a small cluster of 8 machines as it is faster than performing the operation manually 8 times. However, that user will most likely find it intolerable to wait over an hour for the same operation to take effect across 128 cluster nodes. Further complicating matters is that many cluster sites are now hosting multiple clusters that are or will eventually be combined into federated computation clusters. Extending even further is the computation Grid[8] where combining federated clusters from multiple sites is the norm.

Toward this effort, this paper describes the command line interface tools for cluster command and control (C3) developed for use on HighTORC. These same tools are used to provide the back end services for the web based M3C tool.

2. REQUIRED SOFTWARE ENVIRONMENT

A number of tools developed and freely published on the web were collected to facilitate the development of our C3 tool suite. The following section briefly describes and provides information where each of these tools may be obtained.

Rsync is a method for mirroring drives similar to rdist[9]. It operates by: 0) get the list of files to be transferred from a source, 1) split the target file into small fixed-size blocks, 2) generate a checksum – both a weak "rolling" 32-bit checksum and a strong 128-bit MD4 checksum, 3) transfer both checksums, 4) remote system uses the checksums to generate a sequence of instructions used to request blocks to be updated, 5) send requests to update blocks. A more detailed explanation of the rsync algorithm may be found at the rsync web page. This tool is used in C3 to effect the movement of files between server and node. Systemimager uses rsync to move disk images from image server to client node. Rsync simplifies file movement between cluster nodes.

Tomsrtbt is self proclaimed as "the most linux on one floppy". Tomsrtbt is used to build a bootable diskette to initially boot a new cluster node prior to using the cl_pushimage command to install and restart the node as a fully functioning cluster node.

OpenSSH is an open source tool that uses the ssh 1.5 protocol. OpenSSH requires the **openSSL** package to work. OpenSSH uses a digital signature generated by OpenSSL for authentication purposes. This eliminates the need to send passwords across the network to remote machines. OpenSSH is an enabling tool that provides a secure means for C3 to connect to remote cluster nodes from outside the cluster's internal network. Without such security the C3 tool suite would either not be permitted to execute or at the very least would be restricted by security policy. C3 uses OpenSSH for a secure means to eliminate the need to login to each cluster node when invoking a command. OpenSSH is used to both execute a remote command (like rsh) and is used by rsync to transfer files.

Dynamic Hosts Configuration Protocol (DHCP), is used to allocate IP addresses to machines on a given network. It can do this dynamically within a given range of IP addresses or statically by associating a NIC's MAC address with a specific IP address. Dynamic allocation of IP addresses makes it easy to swap out machines with little operator intervention. Static allocation of IP address makes it easier to troubleshoot machine/network problems and subsequently debut distributed cluster codes.

Dynamic IP allocation allows DHCP to "lease" an IP address to a client for a specified time period. This time period may be set from a few seconds to forever. Forever in this case ends when the client gives up the IP address – like on a reboot. A short lease will be detrimental to cluster performance, as it will require the cluster nodes to continually access the DHCP server to obtain a new IP address. A forever lease will produce stability at least until a node crashes or reboots. However, periods of instability in clusters tend to affect more than one node thus making debugging using the IP address useless, as they will shuffle on many nodes simultaneously. Our recommendation is to initially dynamically allocate IP addresses within a specified range in order of cluster node identifier (number). Then use the systemimager tool, described later, to change the dynamic allocation to a static allocation scheme. Thus, ease of initial installation with troubleshooting and debugging capabilities is retained. In practice, the static allocation of IP address on clusters eases the replacement of a cluster node.

Systemimager is a cluster system administrator tool, run from root, that enables a cluster node to *pull* the current cluster node image from an outside cluster image server. This proves to be very valuable when initially installing the cluster environment and for subsequent cluster wide operating system and software environment updates.

When combined with tomsrtbt, Systemimager provides a simplified technique to place the initial cluster image on each cluster node. As each machine is booted from the tomsrtbt diskette, it will *pull* the current image from the image server. The image is downloaded, installed, and now the machine may reboot as a cluster node. The only task remaining is to assign a network IP address to the node.

Systemimager requires DHCP to be installed and run on the outside cluster image server. Configuration options include both dynamic and static assigned IP addresses. DHCP dynamically assigns IP addresses by default as machines boot. However, statically assigned addresses are generally advocated for cluster use as it facilitates node debugging and trouble shooting. However, one can not store statically assigned IP addresses on a cluster image when pushing the same image across a number of machines. To do so would produce disastrous results as all machines would then try to use the same IP address. The alternative is to manually change the IP address of each machine after initial startup and then reboot with the correct IP address. This is not very practical as it is time consuming, error prone, and becomes rather annoying after about four machines. Systemimager provides a solution to this problem via it’s makedhcpstatic utility. Simply boot cluster nodes in order of desired IP address as DHCP will sequentially assign the IP number to each machine as it comes on line. Next run makedhcpstatic which will rewrite the /etc/dhcpd.conf file to associate each node’s MAC address with it’s host name. Last, restart DHCP. Now each time a node requests an IP address, the same number will be assigned. This technique works great for installing a large number of machines with contiguous IP addresses. However, if a machine or NIC is replaced, you must manually set the IP address on the node and in the /etc/dhcpd.conf file. However, this is a small price to pay for the occasional machine or NIC failure compared to the effort required to initially build and configure a large cluster.

The only shortcoming of Systemimager is that it requires the cluster node to request an image from the outside cluster image server. While this technique avoids the security problems of an outside machine forcing new software on a cluster node, perhaps taking control of the machine, it also restricts the usefulness of the image update to preplanned updates (*pulls*) driven by coordinated node cron jobs. Our desire was to augment the features of Systemimager such that a system administrator may effectively *push* a new system image in a secure manner across the entire cluster or any portion of the cluster. Thus, the creation of cl_pushimage.

3. C3 TOOLS SUITE

Eight general use tools have been developed in this effort thus far. Cl_pushimage is our single machine *push* answer to the Systemimager *pull* image solution. Like Systemimager, cl_pushimage and cl_shutdown are both root user system administrator tools. The other six tools, cl_push, cl_rm, cl_get, cl_ps, cl_kill, and cl_exec are tools that may be employed by any cluster user both at the system and application level. Cl_push will let you push individual files or directories across the cluster. Cl_rm will permit the deletion of files or directories on the cluster. Cl_get copies cluster based files to a user specified location. Cl_ps returns the aggregate result of the ps command run on each cluster node. Cl_shutdown will shutdown nodes specified in command arguments. Cl_kill is used to terminate a given task across the cluster. Cl_exec is the C3 general utility in that it enables the execution of any command across the cluster.

Cl_pushimage 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. While Systemimager provides much of the functionality in this area it fell short in that it did not enable a single point *push* for image transfer. Cl_pushimage 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 outside cluster image server.

Cl_pushimage uses a PERL script to iterate through a series of IPs provided by the user. The default cluster configuration file /etc/c3.conf specifying all cluster nodes is used if the user does not provide their own IP address or configuration file. At each iteration, IP address, cl_pushimage uses OpenSSH to call the Systemimage tool updateimage on the cluster machine. Thus, effectively pushing an image to the specified nodes. OpenSSH is employed to provide secure root access to each of the cluster machines from the outside cluster image server. Of course this description assumes that Systemimager has already been employed to capture and relocate a cluster node image to the outside cluster image server machine.

SYNOPSIS

```
      cl_pushimage [OPTIONS]…  --image=[list:]imagename
```

OPTIONS

```
      --help          display help message
```

--nolilo	don’t run lilo after update
--reboot	reboot node after update completes

GENERAL

There are two different ways to call cl_pushimage :

-using the default list of clusters

“cl_pushimage –image=imageName

-using a subset of the cluster from a given file

“cl_pushimage –image=list_of_nodes:imageName

While cl_pushimage has the ability to push an entire disk image to a cluster node, it is too cumbersome as an application support tool when one simply desires to push files or directories across the cluster. Furthermore, cl_pushimage is only available to system administrators with root level access. From these restrictions grew the desire for a simplified cluster *push* tool, **cl_push**, providing the ability for any user to *push* files and entire directories across cluster nodes. Cl_push uses rsync to push files from server to cluster node. ***Caution –*** *do not use cl_push to push the root file system across nodes. Systemimager provides a number of special operations to enable cl_pushimage and updateimage to operate properly.*

SYNOPSIS

cl_push [OPTIONS]… --source=Source --destination=[list:]destination

OPTIONS

-d,--delete	removes any file that are on the nodes but not on the server
-s,--source	the directory, file, or pattern to move
-d,--destination	the destination directory or the destination file on the nodes. Using the list option allows you to send to a subset of nodes specified in the file.

GENERAL

There are several different ways to call cl_push, below are some of the ways:

-for moving whole directories

“cl_push –source=/home/* --destination=/home/”

the use of the backslash before the “*” prevents the shell from trying to expand the special character before the call is made.

-for a single file

“cl_push –source=/home/filename –destination=/home/”

-for a single file and renaming that file on the nodes

“cl_push –source=/home/filename1 –destination=/home/filename2”

-to move a set of files mathing a pattern

“cl_push –source=/home/*.*c –destination=/home/”

again, notice the backslashes proceding the special characters to prevent the shell from expanding them.

-to move a file to a subset of your nodes

“cl_push –source/home/filename –destination=list_of_nodes:/home/

cl_rm is the cluster version of the rm delete file/directory command. This 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 cl_rm due to the potential problems associated with numerous nodes asking for delete confirmation.

SYNOPSIS

cl_rm [OPTIONS]… --files=[list:]pattern

OPTIONS

--help	display help message
-r	recursive delete
-v	verbose mode, shows error message from rm
-files	the file or pattern to delete

GENERAL

There are several different ways to call cl_rm :

-to delete a directory (must be done recursively)

"cl_rm –r –files=/home/usr/*"

notice the use of a \ before the special character. The shell try's to expand the wildcards before the program is called and this forces the shell not to expand them.

-to delete a single file

"cl_rm –files=/home/filename"

The converse of cl_push is the **cl_get** command. This command will retrieve the given files from each node and deposits them in a specified directory location. Since all files will originally have the same name, only from different nodes, each file name has an underscore and IP or domain name appended to its tail. IP or domain name depends on which is specified in the cluster specification file. Note that cl_get operates only on files and ignores subdirectories and links.

SYNOPSIS

cl_get --target=target --source=[list:]pattern

OPTIONS

None.

GENERAL

There are two basic ways to call cl_get :

-to get a given pattern(in this case a whole directory)

"cl_get --target=/home/usr/ --source=/home/usr/*"

notice the use of a \ before the special character. The shell trys to expand the wildcards before the program is called and this forces the shell not to expand them.

-to get a single file

"cl_get –target=/home/ --source=/home/usr/filename"

notice that target is always a directory as this is the file destination.

The **cl_ps** utility runs the ps command on each node of the cluster with the options specified by the user. For each node the output is stored in /$HOME/ps_output. A cl_get is then issued for the ps_output file returning each of these to the caller with the node ID appended per the cl_get command. The cl_rm is then issued to purge the ps_output files from each of the cluster nodes.

SYNOPSIS

cl_ps –options=ps options [–list=cluster list]

OPTIONS

--options	=	Put the options you want ps to use here, any option ps reconizes is fine here
--list	=	this is an optional list of clusters

GENERAL

An example useage of cl_ps is as follows:

"cl_ps –options=A"

runs ps with the –A options on all nodes. If more than one option is needed, use --options=ABC for the ABC options.

Without a cluster shutdown command it is very time consuming to log onto each node and perform an orderly shutdown process. If the direct approach of simply powering down machines is taken, the penalty will be paid on the subsequent startup as each machine will then spend time checking its respective file system. Although most clusters are not frequently shutdown in their entirety, clusters that multi-boot various operating systems will most definitely benefit from such a command as will all clusters after updating the operating system kernel. Also, on those rare occasions where a cluster must be brought down quickly, such as when on auxiliary power due to a power outage, the **cl_shutdown** is much appreciated. Thus, the cl_shutdown was developed to avoid the problem of manually talking to each of the cluster nodes during a shutdown process. As an added benefit, many motherboards now support an automatic power down after a halt - resulting in an "issue one command and walk away" administration for cluster shutdown.

SYNOPSIS

cl_shutdown --options=options -t=time [–list=cluster_list --message="message to send"]

OPTIONS

--options	=	Put the options you want shutdown to use here, any options shutdown reconizes is fine here
--list	=	this is an optional list of clusters, see below for the file format.
--t	=	time before shuting down
--message	=	message to display on each node

GENERAL

An example useage of cl_ps is as follows:

"cl_shutdow –options=h –t=1 –message="system shutting down""

Halts each machine in one minute displaying the given message on each node.

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

SYNOPSIS

cl_kill –signal=signal –process=[list:]process name [--user=username]

OPTIONS

```
--signal  = use the same format and signals you would normally use with kill
list:     = this is an optional list of clusters
--process = the name of the process being killed( not the PID or job number )
--user    = the name of the user whose process to kill. this can only be used
            by root. ALL specifies all users. This searches /etc/passwd for a
            UID.
```

GENERAL

An example useage of cl_ps is as follows:

"cl_kill –signals=9 –process=a.out –user=ALL"

does a kill –9 (unconditional kill) on all a.outs(–user=all specifies that all a.outs running on a system be killed, regardless of the user.

The **cl_exec** is the utility tool of the C3 suite in that it enables the execution of any command on each cluster node. As such, cl_exec may be considered the cluster version of rsh. A string passed to cl_exec is executed "as is" on each node. This provides a great deal of flexability in both the format of command output and arguments passed in to each instruction.

SYNOPSIS

cl_exec [OPTIONS] --command="[list:]string"

OPTIONS

```
--help        =  display help message
--command -c  =  The string to be passed the each node in the cluster
-p            =  print the name of the node before executing the string
--list -l     =  file containing a list of nodes to send the string to.
```

GENERAL

There are two basic ways to call cl_exec:

-to execute a command

"cl_exec -command "mkdir temp""

notice the use of the "'s. this allows perl to interpret what is inside the quotes as a single string.

-to print the machine name and then execute the string “cl_exec -p -c "ls -l"”

this allows the ability to read the output from a command such as ls and to know which machine the message came from.

4. CONCLUSION

Although clusters are relatively inexpensive to build while providing good performance in comparison to recent supercomputers, the lack of good administration and application tools represents a hidden cost to cluster owners and users. This paper presented a number of command line tools developed as part of the Oak Ridge National Laboratory’s HighTORC cluster project that are designed to reduce the cost of cluster ownership. These same tools provide the “backend” connection to the cluster for our web based GUI tool suite M3C (Monitoring and Managing Multiple Clusters) [10]. Both the C3 and M3C may be found at http://www.epm.ornl.gov/torc.

Of the three criteria set forth for our tools at the beginning of this paper – single machine look and feel, secure, scalable – the one we acknowledge falling short of is scalability. The current implementation of some of our commands iterate through a list of nodes on the server side. While we acknowledge this is a problem – we are very pleased with performance and cost savings resulting from use on our small 64-node HighTORC cluster. Furthermore, we are presently working on two competing techniques that are expected to greatly improve the scalability of our tools not only on large clusters but also on federated clusters and across the computation Grid.

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.

Brian Luethke is a participant in the Office of Science, DOE Energy Research Undergraduate Laboratory Fellowships, Spring 2000.

REFERENCES

[1] The HighTORC system page, http://www.epm.ornl.gov/torc
[2] Rsync system documentation, http://www.rsync.samba.org
[3] Tom’s root boot utility, http://www.toms.net/rb
[4] OpenSSH Specifications, http://www.openssh.com/
[5] ISC Dynamic Host Configuration Protocol, http:// www.isc.org/products/DHCP
[6] Systemimager documentation, http://www.systemimager.org
[7] Ptools project, http://www-unix.mcs.anl.gov/sut/
[8] The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, Inc., San Francisco, 1999.
[9] Rdist home page, http://www.magnicomp.com/rdist

[10] M3C Tool, http://www.epm.ornl.gov/~jens/m3ctool