

Tru64 UNIX Best Practice

Using cron in a TruCluster Server Cluster

Revision 1.0 September 2001

This Best Practice describes how to use the cron daemon in a TruCluster™ Server environment for the Tru64™ UNIX operating system.

Contents

Using cron in a Cluster

Is This Best Practice Right for You?	1
Before You Begin	1
Understanding the Problem	2
No Simple Solution	3
Evaluating the Deficient Script	3
Applying the Best Practice	4
A More Thorough Solution	4
Where to Place the Shell Script	5
Creating the Add and Remove Script Functions	6
Creating Code to Determine the Member and Resource State	7
Creating Entry Points Called by the CAA Action Script ...	9
The Complete <code>clustercron</code> Shell Script	10
Creating the CAA Resource and Action Script	13
Verifying Success	15
Troubleshooting	16
Comments and Questions	18
Legal Notice	18

Using cron in a Cluster

The cron system clock daemon is a member-specific daemon with member-specific configuration files. Although this functionality provides some flexibility, you cannot schedule jobs to run on any cluster member, instead of on a specific member.

This Best Practice addresses how you can use cron in a cluster to schedule jobs to run on any member.

See the Tru64 UNIX Best Practices Web page for more information about Best Practices documentation.

Is This Best Practice Right for You?

Not all Best Practices apply to all configurations, so you must be sure that it is appropriate for your system and circumstances. To use this Best Practice, you must meet the requirements described in the following table:

Requirement	Description
Operating System	Tru64 UNIX Version 5.0 or higher and TruCluster Server Version 5.0A or higher.
Impact on Availability	This Best Practice does not impact system availability.
Skill Level Requirements	This Best Practice is intended for experienced system administrators or programmers. You must be familiar with shell programming and have root access to the TruCluster Server cluster.

Before You Begin

Before you apply this Best Practice, you must understand some background information and determine whether the lack of a clusterwide cron daemon is an issue in your environment.

Understanding the Problem

The cron system clock daemon is not cluster aware. Because it is not cluster aware, the cron daemon on any member is independent from, and unaware of, a cron daemon on other cluster members.

The cron daemon runs commands according to the instructions in the crontab files, which are located in the `/var/spool/cron/crontabs` directory. In a cluster, the `/var/spool/cron` directory is a context-dependent symbolic link (CDSL) that points to the member-specific directory `/cluster/members/{memb}/spool/cron`. The cron daemon for a cluster member sees only its own entries in isolation and is not aware that another cluster member might have the same crontab entry.

This lack of cluster awareness can complicate certain uses of cron, such as the following:

- A job that you want to run on only a single cluster member at a time. Tying the job to a specific cluster member does not take advantage of cluster availability.

For example, assume that you want a cron job to start a large backup operation, or to invoke the `/usr/sbin/defragment` utility. You do not care on which member the cron job runs, but you do not want it to run on multiple members. You do want it to take advantage of Cluster Application Availability (CAA) and to be able to fail over to another member in the case of a problem.

- A job for a particular cluster resource. In this case, the cron job must run on a cluster member that has access to that resource.

In each of these examples, installing the cron job on either a single cluster member or on all of the cluster members is unlikely to produce the desired effect.

This Best Practice addresses how you might use cron in a cluster to address these types of situations.

Note for Advanced Users

This Best Practice provides shell scripts and explains their function. If you are an advanced user who prefers to examine these scripts directly, see the *Example 6* and *Example 7* examples.

No Simple Solution

You might think that creating a CAA action script such as that shown in *Example 1* might provide a simple solution:

Deficient CAA Action Script

```
case $1 in
  'start')
    crontab /usr/users/auser/hourtest
    exit 0
    ;;

  'stop')
    crontab -r
    exit 0
    ;;

  'check')
    exit 0
    ;;
esac
```

Evaluating the Deficient Script

In this scenario, the `hourtest` file contains your crontab entry, such as the following:

```
0-59 * * * * echo The hour is `date`. >/dev/console
```

Because CAA invokes the `crontab` command as root, `crontab` copies the specified file into the `/var/spool/cron/crontabs` directory as `/var/spool/cron/crontabs/root`. The `crontab -r` command then removes this crontab file when the CAA stop action is called.

This simple approach works in a limited manner, but has several potentially major problems:

- The start action replaces the root crontab file, which you probably do not want to occur.
- The stop action removes the root crontab file, which might create even more of a problem. The `crontab -r` command removes all of the housekeeping cron entries that root needs.
- The stop action will not be called in the event of a system halt or crash. Although CAA will fail over the resource to another member, the root crontab file will still exist for the first member and will be invoked

when that first member comes back up. The cron job will then run on multiple cluster members.

Applying the Best Practice

Before you use cron in a cluster, be sure to follow the recommendations in *Before You Begin*.

A More Thorough Solution

A more thorough solution to this problem is to create a script that can determine more information about the cluster environment and the status of CAA resources.

The example that is presented in this Best Practice creates a CAA resource (which is arbitrarily named `clustercron`), an associated CAA action script, and a shell script that is located in `/sbin/init.d` and `/sbin/rc3.d`.

The shell script has six entry points: the standard `start`, `stop`, and `restart` entry points that are expected of files in `init.d`, plus `start`, `stop`, and `check` entry points to be called only by the CAA action script.

The high-level view of the solution is as follows:

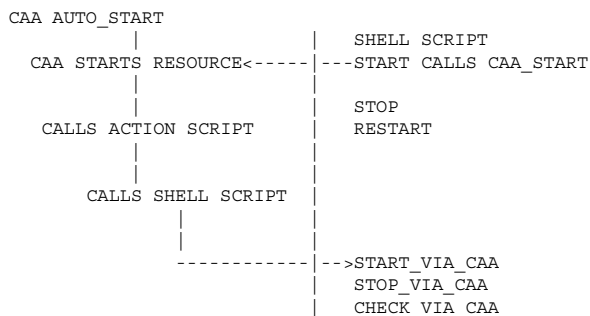
1. When the cluster member boots, CAA attempts to start your resource based on the resource `auto_start` policy; that is, CAA always attempts to start your resource if `auto_start=1`, and attempts to start it only if the resource had been running before the reboot if `auto_start=0`.
2. The `start` entry point in the `/sbin/init.d` script is invoked and also attempts to start the CAA resource. If the resource is already running, `start` determines on which member it is running and whether cleanup is needed on this member.

In particular, if `caa_stat` indicates that the resource is running on another member, remove the crontab entry for the current member if one exists.
3. As part of starting the resource, CAA calls the action script for this resource. The action script calls back into the shell script at a different entry point to add the needed crontab entry for the member on which the resource was started.
4. If CAA fails the resource over to another member, CAA invokes the action script at its `stop` entry point to remove the crontab entry from

the failed member. CAA also invokes the action script at the start entry point to add the crontab entry for the member that is now running the resource.

The following figure, *Figure 1*, shows the interaction between the shell script and the CAA action script:

Shell Script and CAA Action Script Interaction



Where to Place the Shell Script

You might find it most convenient to place your shell script in `/sbin/init.d` and `/sbin/rc3.d` so that it will be invoked automatically at boot time and shutdown time, and when entering a multiuser run level.

By convention, files in the `rc3.d` directory begin with either the letter S (start) or the letter K (kill) and are followed by a two-digit number and a filename. (Typically, files that begin with the letter K are located in the `/sbin/rc2.d` directory.) Files in the `/sbin/rc3.d` and `/sbin/rc2.d` directories are normally links to files in the `/sbin/init.d` directory, as shown in the following example:

```

# ls -l /sbin/init.d/cron
-rwxr-x--- 1 bin bin          2714 Aug 14 23:13 /sbin/init.d/cron
# ls -l /sbin/rc3.d/S57cron
lrwxrwxrwx 1 root bin          14 Sep 12 08:44 /sbin/rc3.d/S57cron -> ../init.d/cron

```

Place your shell script after the `S15clu_cca` entry that starts CAA and after the `S57cron` entry that starts the cron daemon.

This Best Practice assumes that you place the shell script in these directories. See `rc3(8)` for additional information.

Creating the Add and Remove Script Functions

Create script functions that add and remove your crontab entries. The example *Example 2* shows how to add the crontab entry.

Adding the crontab Entry

```
add_cron()
{
  crontab -l > /mytmp/clustercron.$$
  grep -q 'The hour is' /mytmp/clustercron.$$
  if [ $? -ne 0 ] 1
  then
    (
      echo "0-59 * * * * echo The hour is `date\`.>/dev/console" 2
    ) >> /mytmp/clustercron.$$
    crontab /mytmp/clustercron.$$
  else
    echo "Notice: crontab entries for clustercron already exist !"
  fi
  rm -f /mytmp/clustercron.$$
}
```

- 1** One way to add the entry is to use the `crontab -l` command to output the contents of the root crontab file to a temporary file (in a directory that is not world-writable) and then use the `grep` command to look for text that uniquely identifies your desired crontab entry.

The `grep` command returns zero if a match is found and nonzero if no match is found. If the crontab entry is not there, append the entry to the temporary file and then use the temporary file as input to `crontab` to rewrite the root crontab file. When the action is complete, remove the temporary files.

- 2** Add your cron job here. Because your CAA action script will invoke the `crontab` as root, the `crontab` command copies the specified file into the `/var/spool/cron/crontabs` directory as `/var/spool/cron/crontabs/root`.

To remove the crontab entry, again use `crontab -l` to output the contents of the root crontab file to a temporary file and then match the text that uniquely identifies your desired crontab entry.

The example *Example 3* shows how to remove the crontab entry.

Removing the crontab Entry

```
remove_cron()
{
  crontab -l > /mytmp/clustercron.$$
  # check to see if we have anything to remove
  grep -q 'The hour is' /mytmp/clustercron.$$
}
```

```

if [ $? -eq 0 ] 1
then
    # remove just the clustercron cron entries
    # note! this code needs to be adapted to your
    # crontab entry.....
    sed -e '/The hour is/d' < /mytmp/clustercron.$$ > /mytmp/clustercron2.$$
    crontab /mytmp/clustercron2.$$
fi
rm -f /mytmp/clustercron.$$ /mytmp/clustercron2.$$
}

```

-
- 1 If there is a match, use `sed` to remove the entry from the temporary file. Then, write the revised file to a second temporary file and use this second temporary file as input to `crontab` to rewrite the root crontab file. When the action is complete, remove the temporary files.

Your script code needs to be intelligent enough to remove only the entries that `add_cron` added.

Note

Administrators need to be aware that the entries are added and removed by scripts. We recommend that administrators not adjust these cron entries by hand.

Creating Code to Determine the Member and Resource State

Add code in the `/sbin/init.d/clustercron` start function to verify that the cron entries are removed if the resource is not started on the current cluster member. This test is needed in case the member crashes, or is halted with an option that does not run the stop scripts, while it has the cron entries for the resource. If such a crash or halt occurs, the CAA action script shutdown entry point is not called and the crontab entry will still exist when the member is restarted.

Example 4 shows a sample `/sbin/init.d/clustercron` skeleton script.

The echo statements are captured in the log files for debugging purposes. You may or may not want to use them.

Determining the Member and Resource State

```
clu_get_info -q
is_cluster=${?}

case "$1" in
'start')
    if [ "$is_cluster" = 0 ] 1
    then
        if caa_start -q clustercron; then 2
            # note: add_cron will be called from caa_start...
            echo "clustercron entry started"
        else 3
            # resource is already started... maybe on another member
            # check to see if our cron entries need to be removed
            # on this member
            m1='caa_stat clustercron | grep STATE'
            m2="STATE=ONLINE on `hostname` | cut -d. -f1" 4
            if [ "$m1" != "$m2" ]
            then
                echo "removing cron"
                remove_cron
            fi
        fi
    else
        # not in a cluster... start clustercron
        exit 0 5
    fi
;;

'stop')
    # No need to do anything. Cleanup is done on start.
    # Allow CAA to fail over the resource if a member of
    # the cluster shuts down.
    exit 0 6
;;

'restart')
    if [ "$is_cluster" = 0 ]
    then
        m1='caa_stat clustercron | grep STATE'
        m2="STATE=ONLINE on `hostname` | cut -d. -f1"
        if [ "$m1" != "$m2" ]
        then
            if caa_stop -q clustercron; then
                echo "clustercron entry stopped"
            fi
            if caa_start -q clustercron; then
                echo "clustercron entry started"
            else
                echo "clustercron entry not started on this member"
            fi
        fi
    else
        # not in a cluster...nothing to do
        exit 0
    fi
fi
```

-
- 1 When the system boots, the `start` entry point is called. The `clu_get_info` command returns zero if successful; that is, this is a cluster member.
 - 2 Attempt to start the CAA resource by a call to `caa_start resourcename`. In many cases, this call is redundant and fails because CAA has already started the resource based on the CAA `auto_start` policy.
The call to `caa_start resourcename` invokes the CAA action script for this resource, which in turn calls the `start_clustercron` entry point in the shell script.
 - 3 If `caa_start` succeeds, the cron job was not already running in the cluster. If the start fails, the cron job is already running somewhere in the cluster and you need to determine whether to remove the crontab entry for this node.
 - 4 Verify the CAA state and compare it to a fabricated ONLINE string that includes the hostname. The `caa_stat` command returns in the form `STATE=ONLINE on hostname`. Knowing this, you can verify the returned state against this string. If they are not the same, remove the crontab entry.
 - 5 If this is not a cluster, the script exits at this point.
 - 6 Do not explicitly stop the resource. Instead, allow CAA to restart the resource in the event of a failure. If you were to explicitly stop the resource via a call to `caa_stop`, CAA would not fail over the resource. You do not need to remove the crontab entry because, when the member reboots, the `start` entry point in the script removes the crontab entry if needed.

Creating Entry Points Called by the CAA Action Script

Add code in the `/sbin/init.d/clustercron` shell script for the entry points that the CAA action script calls, as shown in *Example 5*.

Entry Points Called by the CAA Action Script

```
'start_clustercron')
    # add cron entries.
    start_clustercron
    add_cron
    ;;
'shutdown_clustercron')
    # remove cron entries
    remove_cron
    shutdown_clustercron
    ;;
'check_clustercron')
    # Check for the presence of the crontab entry.
    crontab -l > /mytmp/clustercron.$$
    grep -q 'The hour is' /mytmp/clustercron.$$
    exitStatus=?
    rm -f /mytmp/clustercron.$$
    exit $exitStatus
    ;;
*)
    echo "Unknown action \"${1}\" specified".
    echo "usage: $0 [start|stop|restart|start_clustercron|
    shutdown_clustercron|check_clustercron"
    exit -1
    ;;
esac
```

The `check_clustercron` entry point is called only when the resource is running. Any action that is invoked by this check executes on the member on which the resource is running; you do not need to determine that member. Because the resource is running, the crontab entry should exist. Therefore, test for the presence of the crontab entry and return the status to CAA.

The Complete `clustercron` Shell Script

Example 6 shows the complete `clustercron` shell script.

`clustercron` Shell Script

```
#!/sbin/ksh

typeset -t m1=''
typeset -t m2=''

add_cron()
{
    crontab -l > /mytmp/clustercron.$$
    grep -q 'The hour is' /mytmp/clustercron.$$
    if [ $? -ne 0 ]
    then
        (
            echo "0-59 * * * * echo The hour is `date\`.>/dev/console"
        ) >> /mytmp/clustercron.$$
        crontab /mytmp/clustercron.$$
    fi
}
```

```

else
    echo "Notice: crontab entries for clustercron already exist !"
fi
rm -f /mytmp/clustercron.$$
}

remove_cron()
{
    crontab -l > /mytmp/clustercron.$$
    # check to see if we have anything to remove
    grep -q 'The hour is' /mytmp/clustercron.$$
    if [ $? -eq 0 ]
    then
        # remove just the clustercron cron entries
        # note! this code needs to be adapted to your
        # crontab entry....
        sed -e '/The hour is/d' < /mytmp/clustercron.$$ > /mytmp/clustercron2.$$
        crontab /mytmp/clustercron2.$$
    fi
    rm -f /mytmp/clustercron.$$ /mytmp/clustercron2.$$
}

start_clustercron()
{
    echo "start_clustercron function"
}

shutdown_clustercron() {
    echo "shutdown_clustercron function"
}

clu_get_info -q
is_cluster=$?

case "$1" in
'start')
    if [ "$is_cluster" = 0 ]
    then
        if caa_start -q clustercron; then
            # note: add_cron will be called from caa_start...
            echo "clustercron entry started"
        else
            # resource is already started... maybe on another member
            # check to see if our cron entries need to be removed
            # on this member
            m1='caa_stat clustercron | grep STATE'
            m2="STATE=ONLINE on `hostname | cut -d. -f1`"
            if [ "$m1" != "$m2" ]
            then
                echo "removing cron"
                remove_cron
            fi
        fi
    else
        # not in a cluster...nothing to do
        exit 0
    fi
fi

```

```

;;
'stop')
# No need to do anything. Cleanup is done on start.
# Allow CAA to fail over the resource if a member of
# the cluster shuts down.
exit 0
;;

'restart')
if [ "$is_cluster" = 0 ]
then
m1=`caa_stat clustercron | grep STATE`
m2="STATE=ONLINE on `hostname | cut -d. -f1`"
if [ "$m1" != "$m2" ]
then
if caa_stop -q clustercron; then
echo "clustercron entry stopped"
fi
if caa_start -q clustercron; then
echo "clustercron entry started"
else
echo "clustercron entry not started on this member"
fi
fi
else
# not in a cluster...nothing to do
exit 0
fi
;;

'start_clustercron')
# add our cron entries if needed.
start_clustercron
add_cron
;;

'shutdown_clustercron')
# remove our cron entries
remove_cron
shutdown_clustercron
;;

'check_clustercron')
# See if the resource is running, and report if not
crontab -l > /mytmp/clustercron.$$
grep -q 'The hour is' /mytmp/clustercron.$$
exitStatus=$?
rm -f /mytmp/clustercron.$$
exit $exitStatus
;;

*)
echo "Unknown action \"$1\" specified".
echo "usage: $0 {start|stop|restart|start_clustercron|
shutdown_clustercron|check_clustercron}"
exit -1
;;

esac

```

Creating the CAA Resource and Action Script

CAA controls your cron job and fails it over to another member when necessary. The CAA action script, which is named after the CAA resource name of your choice, invokes the startup and shutdown functions in the shell script. The cron entries are added or removed as the CAA resource is started and stopped. If the cluster member on which the cron job is running fails or is shut down, CAA relocates the crontab entry to another member.

Use the `caa_profile` command to create, validate, delete, and update a CAA resource profile, and use the `caa_register` command to register the resource after you have created it. Use `application` as the resource type and specify the location of the action script.

Set `auto_start` to `auto_start=1` if you want to start the resource automatically, regardless of whether it had been stopped or running before the reboot. Set `auto_start=0` if you want to start the resource only if it had been running before the reboot.

Note

Remember that the shell script in `/sbin/init.d` always attempts to start the resource. If you do want to override an `auto_start` value of `auto_start=0`, or if you want to stop the CAA resource and keep it stopped, remove the shell script from `/sbin/init.d`.

An example `caa_profile -create` command is as follows:

```
# caa_profile -create clustercron -a clustercron.scr -t application -o as=1
```

You can specify either a full pathname for the script file, or its filename, in which case the `caa_profile` command looks for the file in the `/var/cluster/caa/script` directory.

By default, `caa_profile -create` enables the cron job to run on all members with a placement policy of `balanced`. Your cron job can therefore run on any member of the cluster. If this default does not meet your needs, use the `-c` option to specify the members that can host the application resource and use the `-p` option to specify a placement policy.

The default check interval is 60 seconds. The check interval is the maximum amount of time that an application can be unavailable to clients before CAA attempts to restart it. Under some circumstances, this

default can lead to a situation where a resource fails and is not restarted quickly enough. If your cron entry is particularly time critical, you need to determine whether the 60-second check interval meets your needs.

If the `caa_profile -create` command completes successfully, use the `caa_profile -print resource_name` command to verify the profile is as you intended:

```
# caa_profile -print clustercron
NAME=clustercron
TYPE=application
ACTION_SCRIPT=clustercron.scr
ACTIVE_PLACEMENT=0
AUTO_START=1
CHECK_INTERVAL=60
DESCRIPTION=clustercron
FAILOVER_DELAY=0
FAILURE_INTERVAL=0
FAILURE_THRESHOLD=0
HOSTING_MEMBERS=
OPTIONAL_RESOURCES=
PLACEMENT=balanced
REQUIRED_RESOURCES=
RESTART_ATTEMPTS=1
SCRIPT_TIMEOUT=60
```

Finally, use the `caa_register resource_name` command to register the resource with CAA:

```
# caa_register clustercron
```

See `caa_profile(8)` and `caa_register(8)` for additional information.

An example CAA action script is shown in *Example 7*.

The example directs output to log files, using the arbitrary directory `/usr/users/clustercron`. These log files can be useful for debugging purposes. Change the directory as appropriate.

Complete CAA Action Script

```
#!/usr/bin/ksh -p
#
# Start clustercron entry
#

svcName="clustercron"           # Servicename
CAALOGDIR="/usr/users/clustercron" # Directory for logfiles
ACTION=$1                       # Action (either start or stop)
LOG="${CAALOGDIR}/${ACTION}_${svcName}.$$" # Destination for script output

case $1 in
'start')
    ksh /sbin/init.d/clustercron start_clustercron >> ${LOG}
    exit 0
    ;;
'stop')

```

```

ksh /sbin/init.d/clustercron shutdown_clustercron >> ${LOG}
exit 0
;;
'check')
ksh /sbin/init.d/clustercron check_clustercron >> ${LOG}
if [ $? -ne 0 ]
then
exit -1
else
exit 0
fi
;;
*)
echo "usage: $0 {start|stop|check}"
;;
esac

```

Verifying Success

After you apply the Best Practice for using cron in a cluster, you can verify whether it was successful by performing the following steps. You will probably find it easier to call the CAA commands interactively to test your work and get immediate feedback. After you have the script working correctly, make sure that you place it in the `/sbin/init.d` and `/sbin/rc3.d` directories.

In this discussion, the `active` member is the member on which CAA starts the resource; it may or may not be the member that you are currently using.

1. To determine if the CAA resource is online, and, if so, on which member, use the `caa_stat` command. If the resource is not online, use the `caa_start` command to start the CAA resource.

CAA reports the member on which the resource is started (and therefore the member on which the crontab entry is added), as follows:

```

# caa_start clustercron
Attempting to start 'clustercron' on member 'deli'
Start of 'clustercron' on member 'deli' succeeded.

```

2. To verify that your crontab entry was added on the active member, examine the `/var/spool/cron/crontabs/root` file.
3. To verify that the crontab entry was not added on other members, examine the `/var/spool/cron/crontabs/root` file on those members.
4. To stop the CAA resource, use the `caa_stop` command on any member.

5. To verify that the crontab entry was removed, examine the `/var/spool/cron/crontabs/root` file on the formerly active member.
6. To start the CAA resource on another member, use the `caa_start` command with the `-c` option.
7. To make sure that your crontab entry was added, examine the `/var/spool/cron/crontabs/root` file of the member that is now running the CAA resource.
8. To make sure that the crontab entry was added correctly, perform this sequence of steps again at the next system reboot.

If the Best Practice was not successful, see *Troubleshooting* for information about identifying and solving problems.

Troubleshooting

If you determine that the Best Practice was not successful, as described in *Verifying Success*, use the following table to identify and solve problems. In this discussion, the `active member` is the member on which you run the CAA resource.

Problem	Possible Solutions
The <code>caa_stat</code> command indicates that the CAA resource is offline.	Use the <code>caa_start</code> command interactively to start the CAA resource. The CAA action script then calls back into your shell script and exercises the code paths. If the interactive <code>caa_start</code> command brings the CAA resource online, verify the <code>start</code> entry point in your shell script to make sure that the <code>caa_start -q your-resource-name</code> command starts the correct CAA resource. Also, make sure that you placed your shell script in the <code>/sbin/init.d</code> directory.

Problem	Possible Solutions
The CAA resource is online, but the crontab entry is not added.	<p>The shell script was invoked and it successfully called the <code>caa_start</code> command to start the CAA resource. CAA then called the CAA action script, which calls back into the shell script at the <code>start_clustercron</code> entry point. The problem is likely in the <code>add_cron</code> function code path.</p> <p>Also, you might be looking in the wrong place. The CAA resource and cron job might have started on another member. Use the <code>caa_stat</code> command to determine on which member the resource was started, and verify the crontab entry for that member.</p>
The cron job is running on multiple members.	<p>The <code>remove_cron</code> function assumes that the cron job is running only on one member. Therefore, issue the <code>caa_stop your-resource-name</code> command interactively to stop the CAA resource and remove the crontab entry on the active member. Then, to return to a known state, manually remove the crontab entries on the other members as needed. After you have done this, do the following:</p> <ol style="list-style-type: none"> 1. To start the CAA resource on the active member, issue the <code>caa_start your-resource-name</code> command interactively. 2. To stop the CAA resource on the active member, issue the <code>caa_stop your-resource-name</code> command interactively. Your CAA action script then calls back into the shell script at the <code>shutdown_clustercron</code> entry point. 3. Examine the <code>/var/spool/cron/crontabs/root</code> file for your cron job. The crontab entry should not be there. If it is there, the problem is likely in the <code>remove_cron</code> function code path.

Comments and Questions

We value your comments and questions on the information in this document. Please mail your comments to us at this address:

`best_practices@zk3.dec.com`

Legal Notice

Compaq and the Compaq logo Registered in U.S. Patent and Trademark Office. Tru64 is a trademark of Compaq Information Technologies Group, L.P. in the United States and other countries. UNIX is a trademark of The Open Group in the United States and other countries. All other product names mentioned herein may be trademarks of their respective companies.

Confidential computer software. Valid license from Compaq required for possession, use, or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Compaq shall not be liable for technical or editorial errors or omissions contained herein. The information in this document is provided "as is" without warranty of any kind and is subject to change without notice. The warranties for Compaq products are set forth in the express limited warranty statements accompanying such products. Nothing herein should be construed as constituting an additional warranty.