CLUSTER HEALTH CHECK

User Manual

Version 1.1.1.0

Revision 1.1

June 11, 2014

TABLE OF CONTENTS

Change History....3

1.0 OVERVIEW....5

1.1 Requirements....5
1.2 Installation....5
1.3 Configuration....5
1.3.1 Tool configuration files....7
1.3.2 Group configuration files....10
1.4 Tool output processing methods....12

2.0 FRAMEWORK....14

2.1 The hcrun command....14
2.2 The compare_results command....26
2.3 Using config_checkcompare_results and hcrun to run health checks....32

3.0 GUIDELINES FOR ADDING NEW TOOLS....39

5.0 PACKAGING....41

5.1 RPM Contents....41

6.0 USE CASES....49

6.1 Initializing your environment)....49
6.2 Verifying consistency within a nodegroup....49
6.3 Verifying consistency within a nodegroup over time....50
6.4 Verifying configuration after maintenance....51
6.5 Using baseline history....51
6.6 Running Node Tests....52
6.7 Running Fabric Tests....52

ABOUT

To determine the health of a cluster, it is necessary to get the current state of all the components which build up the cluster. In most installations these are:

- Compute nodes
- Ethernet network
- InfiniBand network
- Storage

In this document, we describe the usage of the Cluster Health Check (CHC) framework which is used to perform checks on these components. By working with results of these checks, we will be able to state if the cluster is healthy or not.

Change History

Version	Changes

1.1.1.0	Document changes:
	• Add change history
	• Add section on Packaging
	• Add section on Use cases
	• Include link to developerWorks page on CHC
	• Updated section on adding new tools
	• Altered based on CHC changes listed below
	
	CHC changes:
	• Updates to -l option output:
	  ○ Format last column of long listing of tools and groups (-l option)
	  ○ Output config filename for tool or tool group – helpful for determining where to change behavior
	• Processing methods updates:
	  ○ Use compare_results instead of config_check
	  ○ compare_results vs. config_check → better baseline handling; baseline history; multiple input types (xcoll results; or pipe into xcoll); use HC_NODERANGE; more error handling; veryverbose; use term baseline vs. template
	  ○ Add xcoll_diff to processing methods
	• Add support for recommendation and comment keywords to config files
	• Improve remote command support over xdsh
	  ○ Generate environment file and pass using xdsh
	  ○ Handle remote command return codes from xdsh rather than getting the return code from xdsh itself
	• Display recommendation keyword contents for non-passing tool results

Version	Changes
	• Added man pages • Added bpcheck tool for x86 based clusters

1.0 Overview

The Cluster Health Check framework provides an environment for integrating and running a subset of individual checks which can be performed on a single node or a group of nodes. The tests are categorized into different tool types and groups, so that only checks for a specific part of the cluster can be performed. Because the framework and toolset is extensible. users can also create their own groups and tools for checking different components of the cluster.

The main wrapper tool is **`hcrun`**, which provides access to all tools and passes them the environment.

1.1 Requirements

The cluster health check framework with toolkit should be installed on the xCAT server because many of the individual checks and framework use xCAT commands. In addition some of the health checks are MPI parallel applications which run using POE runtime environment.

Required software packages

- xCAT
- POE environment
- Python 2.6

1.2 Installation

The cluster health check framework along with some tools is delivered as an rpm, and is installed as user root with the rpm command:

 # rpm -ihv cluster-healthcheck-1.1.0.0-0.noarch.rpm

The rpm command installs framework which consists few commands and health check tools to */opt/ibmchc*.

To run the health checks directly from command prompt, there are two possibilities.

Either edit *~/.profile* and add the following line:

 # export PATH=$PATH:/opt/ibmchc/bin

Or create a symlink to */opt/ibmchc/bin/hcrun* with:

 # ln -s /opt/ibmchc/bin/hcrun /usr/bin/hcrun

1.3 Configuration

The cluster heath check and verification tools are developed to check the health of individual cluster components or set of cluster components. The health check tools can be grouped into different groups of tools based on cluster components checked by the tools. There are three different types of configuration files (global, tool, group) to configure framework, individual tools and tool groups. These configuration files help to integrate new health check tools seamlessly

with the framework. The general format of the configuration file is keyword value pairs defined under sections.

The Master or global configuration file chc.conf is created in the directory /etc/opt/ibmchc/. The settings can be changed in Master configuration file */etc/opt/ibmchc/chc.conf* as shown in *Example 1-1*.

Example 1-1 /etc/opt/ibmchc/chc.conf

```
[MasterConfig]

# The list of high level directories under
# which configuration of different health
# checking tools and groups are defined.
configpath =
/opt/ibmchc/tools:/opt/ibmchc/ppc64/tools:/opt/ibmchc/x86_64/tools:/opt/ibmchc/conf

# stdout results of health checking tools are saved in the
# file specified below.
toolsoutput = /var/opt/ibmchc/log/toolsoutput

# The summary of health checking tools
# are logged in the file defined below.
summaryfile = /var/opt/ibmchc/log/hcrun_sum.log

# The hcrun logs are stored in the file defined below.
hcrunlog = /var/opt/ibmchc/log/hcrun.log

# The user id which is used to run MPI jobs.
# Tools can access user name using environment variable $HC_USER.
# No default user is used.
hcuser =

# The comma separated UFM server host names or ip
# addresses. No default values.
# Tools can access subnet Managers using environment
# variable $HC_SUBNET_MANAGERS
subnet_managers =

# The comma separated IB Switch nodes or node groups.
# Tools can access IB Switch groups using environment variable $HC_IB_SWITCHES
# No default values.
ib_switches =
```

The defaults values defined are fine for most cases; however the following parameters need to be updated to match each unique cluster's environment:

- hcuser

Some of the tests run with the PE environment. These tests must not be submitted as root, thus a dedicated non-root user must be configured. The value is exported through environment variable *HC_USER*. *The user must have authority to run applications in the PE environment.*

- `subnet_managers`

  If an InfiniBand subnet manager is available, its hostname can be specified in the configuration file . Multiple servers can also be specified as a comma separated list. The value is exported through environment variable *HC_SUBNET_MANAGERS. Tools that require access to subnet manager servers will use this environment variable.*

- `ib_switches`

  Specify the xCAT node groups for the InfiniBand switches as a comma separated list. Note that this requires the switches to be in the xCAT database and have ssh access enabled. Different groups for different model of switches should be created in the xCAT database. The value is exported through environment variable *HC_IB_SWITCHES.Tools that require access to InfiniBand switches will use this environment variable.*

1.3.1 Tool configuration files

Each of the available tools has its own configuration file. The default configuration file should work for most cases. You may wish to customize to your own needs, or if you want to add a tool to the health check framework, you will need to create a configuration file for it. The configuration keywords of the tools are shown in Table 1-1. Each tool configuration file needs to be somewhere in or below the path which is specified by the `configpath` parameter in the global configuration file. All tool configuration files must have the '*.conf*' extension. The following table shows all available parameters or keywords and their value types for such a tool configuration file.

Table 1-1 Health Check Tool specific configuration keywords

Parameter	Required	Value type	Remark
[HealthCheckTool]	Yes	n/a	Section header for the configuration file
name	Yes	String	Name of the Health check Tool
description	Yes	Single line Text	Short description of the tool
Type	Yes	String	Tool type to which this tool belongs. For display purposes only.

Parameter	Required	Value type	Remark
executable	No	Path to executable script or binary	Specifies the program to be run. **Default if not set**: Tool configuration path without '*.conf*' extension
Copy	No	yes | no	Specifies if the tool needs to be copied to the node or node group (uses **xdsh -e**) **Default if not set**: 'yes'
starttool	No	all | local	Specifies if the tool needs to be run on all nodes (provided in a nodelist to hcrun) or just on the node where hcrun is running. **Default if not set**: all
arguments	No	String or Strings separated by space	Default command line arguments for the executable. It is possible to specify non default arguments in the hcrun command line after the tool name argument. If any arguments have multiple words separated with a space then they need to be enclosed within double quotes.
environment	No	Environment variables with their values.	Environment variables exported for the tool (Declaration of multiple variables with ';'). Each environment variable is defined as variable name and value pairs separated by equal (=) sign. If **hcrun -n <NODERANGE>** is specified then the noderange is exported as *HC_NODERANGE*.
precheck	No	Tool name(s)	Optional comma separated list of tool names which will run before this tool
predatacollectiontool	No	Tool name(s)	Optional comma separated list of tool names which will run just before this tool to collect data. Optional command line arguments also can be specified which over-writes the default command line arguments defined for pre-data collection tool.
postdatacollectiontool	No	Tool name(s)	Optional comma separated list of tool names which will run after this tool to collect data. Optional command line arguments also can be specified which over-writes the default command line arguments defined for

Parameter	Required	Value type	Remark
			post-data collection tool.
processmethod	No	`xcoll | compare | xcoll_diff|xcoll, xcolld_diff | xcoll,compare, xcoll_diff`	Optional comma separated list of supported tool output process methods. The process is shown in 1.4 Tool output processing methods, on page 13.
passexitcode	No	Numeric	Optional comma separated list of exit codes of the tool, which indicate the test as “passed”. The exit code “0” is considered default pass exit code.
failexitcode	No	Numeric	Optional comma separated list of exit codes of the tool which indicate the test as “failed”. The non-zero exit codes considered default fail exit provided no pass exit codes are defined. If either pass exit codes, warning exit code, or error exit codes are defined then exit codes which are not members of those exit codes are considered Fail exit codes.
warningexitcode	No	Numeric	Optional comma separated list of exit codes of the tool which indicate the test passed with “warning”. No default exit codes.
errorexitcode	No	Numeric	Optional comma separated list of exit codes which indicate the test has hit “errors” to start. No default exit codes.
recommendation	No	text	Optional text description of recommended actions on a non-passing result. Typical use is to use -l and -t to get a recommendation on a failure, error or warning exit from a tool.
comments	No	text	Option text description for generic comments. Use -l and -t together to see comments.

Example 1-2 shows configuration file for `fs_usage`tool. The executable is copied to all the specified nodes, runs on all of the nodes, and that only when it exits with an exit code zero (0) it is considered to be a successful check. The `fs_usage` check is part of the tool type node. The `fs_usage` tool supports both tool output process methods (xcoll and compare) as described in “1.4 Tool output processing methods” on page 13.

Example 1-2 /opt/ibmchc/tools/node/fs_usage.conf

```
[HealthCheckTool]
name=fs_usage
description=Checks the usage of the provided filesystems
type=node
executable=/opt/ibmchc/tools/node/fs_usage
arguments=/tmp /var/tmp
copy=yes
starttool=all
processmethod=xcoll,compare
passexitcode=0
errorexitcode=128
```

The configuration file makes it easy to implement or integrate new checks into the cluster health check framework. Even the scripting language can be chosen easily, for example Bash, Python or Perl script, as long as the interpreter is available on the target node. There are mainly two types of scripts or checks. One type is simple query scripts, which query values and output them. The other type, check scripts, which have logic implemented and also provide exit codes for pass, fail, or warning. The following are the simple steps followed for adding new script:

1. Create the health check script
2. Copy your script to the appropriate arch specific folder (*/opt/ibmchc/tools* or */opt/ibmchc/ppc64/tools* or */opt/imbchc/x86_64/tools*)
3. Create the configuration file for the script in the same folder with extension '*.conf*'
4. Make the script executable

1.3.2 Group configuration files

The group configuration file defines multiple tools which should run together in a specific order. There are some pre-defined groups which come with the rpm and users also can define their own groups depending on their needs. Predefined groups are located in */opt/ibmchc/conf/groups/chc.* These configuration files may not require editing. The User-defined groups are located in */opt/ibmchc/conf/groups/user.*

Example 1-3 shows a group configuration file example. All tools will be executed in the given order. The Section header is called *[HealthCheckToolsGroup]*. The name, and description are mandatory keywords and keyword environment is optional. All these keywords take Strings as their values. The keyword *tools* is a mandatory keyword whose value is a comma separated list of tool names in a specific order.

Example 1-3 /opt/ibmchc/conf/groups/chc/node_check.conf

```
[HealthCheckToolsGroup]
#The name of the group
name=node_check

#The short description of the tool group.
description=This group has node check tools
```

```
#The environment variables and their values. These environment variables
# are exported while running the member tools as part of the group.
Environment=

#The member tools of the group
tools=cpu, memory, gpfs_state, leds, temp, nfs_mounts, firmware, fs_usage, os
```

The environment variables defined in the group configuration file are exported for every tool of the group before starting the tool. In the case of a conflict, these environment variables over-write the tools environment variablest. To overwrite tool keywords of the tool configuration files (like arguments or environment), add the following keyword to the group configuration file:

<toolname>.<keywordname>=<new value>

as shown in Example 1-4.

Example 1-4 same as Example 1-3 above but with overwritten fs_usage arguments

---

```
[HealthCheckToolsGroup]
#The name of the group
name=node_check

#The short description of the tool group.
description=This group has node check tools

#The environment variables and their values. These environment variables
# are exported while running the member tools as part of the group.
Environment=

#The member tools of the group
tools=cpu, memory, gpfs_state, leds, temp, nfs_mounts, firmware, fs_usage, os

#The fs_usage arguments is redefined in the group.
fs_usage.arguments=/tmp
```

The tool keyword values redefined in the group for a tool is used while running that tool under that group. Otherwise the default values specified for that keyword in tool configuration file is used. The group environment variables and redefined keyword values of tools can be used to change the behavior of the tool while running as part of that group.

Creating a group configuration file and integrating with the framework is simple one step process. That is;

1. Create the configuration file with extension '*.conf*' in */opt/ibmchc/conf/groups/user* using your favorite text editor.

The new group automatically gets integrated with framework.

1.4 Tool output processing methods

The cluster health check provides different methods to process the output of a tool.
The current version provides the following modes:

- plain (no post processing)
- xcoll
- xcoll_diff
- compare

Some tools can't support `compare` and `xcoll` because the output of those tools may not be in the format suitable to use `xcoll` and `compare` methods. Hence, the tools must define the methods they support as shown in "1.3.1 Tool configuration files" on page 8. The processing method `plain` is the default method. The processing methods can be used for health check groups as well. If any member tool of a group does not support specified methods then that tool is run in default (`plain`) mode.

1. **Plain**
   The `plain` mode is the default mode for all the tools. In `plain` mode `tools` will be executed and the output or results are shown without any post processing. This is useful to check values of individual nodes.

2. **xcoll**
   In `xcoll` mode the tool will be executed and the output or results are piped (forwarded) to the xCAT command `xcoll`. The `xcoll` command summarizes the output of different nodes and nodegroups, whose output matchies, to one "output group". More details about the xCAT command `xcoll` can be found at

   http://xcat.sourceforge.net/man1/xcoll.1.html.

   This method is generally used for the health check tools which query the attributes of the cluster components without checking results. This method is good for checking consistency within a nodegroup.

   If a tool already does xcoll, this processing method should not be configured in the tool's configuration file. If that is the case, a request made on the hcrun command line to use xcoll will be ignored, and the tool will be run.

3. **xcoll_diff**
   The xcoll_diff process method uses the xcat command `xcoll` and the /opt/ibmchc/tools/config/compare_results utility to summarizes the output of different nodes and nodegroups and then compare the various results and display the differences between each variation and a base group. The base group is either the results group that has the largest number of nodes in it, or it is the group that has the node specified in the -s <seed_node> command line argument to hcrun. More details about the xCAT command `xcoll` can be found at

   http://xcat.sourceforge.net/man1/xcoll.1.html.

This method is generally used for health check tools which query the attributes of cluster components without checking results. In addition to checking for consistency within a nodegroup, it is helpful in determining differences when the tool output has many lines. Depending on the tool, you may need to use -v to get the full verbose output to do a proper comparison of results.

Furthermore, nodegroup names will be assigned (nodegroup1, nodegroup2, etc..) for each set of results that come from xcoll_diff. That list is provided before two results groups are compared, and then the name is referenced from that point forward; this is done to reduce clutter in the output.

4. **compare**

The compare process method uses */opt/ibmchc/tools/config/compare_results* utility tool to compare the output of the tools with a given baseline, or expected output. Depending on the tool's configuration, compare_results may call xcoll before processing it. If the tool does not have xcoll as a value for the processmethod keyword in its configuration file, it will not run xcoll. In that case, it is assumed that the output of the tool will be in xcoll format.

The baseline can be created by the cluster health check tool by initially running health check tool against a given seed node. This will create a baseline template with the output of seed node. Typically, the user will verify the template when it is first created. On further invocations of the health check, the results of the different nodes are checked against the baseline. The baseline template can be regenerated or replaced by running health check tool using different or same seed node again. Template regeneration is typically done after maintenance is applied that will change the tool's results.

Exceptions found by C*ompare_results* are listed inError: Reference source not foundError: Reference source not found,on page Error: Reference source not found.

Note: It is extremely important that the seed node has the expected results so that subsequent invocations only reveal mismatches to a known good result.

2.0 Framework

The Cluster Health Check framework consists of mainly the **hcrun** command and the utility command **config_check**. In this chapter the framework commands hcrun and config_check are described.

2.1 The hcrun command

NAME

hcrun - Runs individual health checks or group of health checks and processes their results.

PURPOSE

Runs individual health checks or group of health checks on range of nodes or node groups. The results of the health checks are processed using different process methods.

SYNTAX

hcrun [-h] [-v] [-c] { -l | [[-f] -n noderange] [[–s seed_node] –m [-m process_method]{ [-p] group[,group,...] |-t tool [,tool,... | cmd_args]}}

DESCRIPTION

There are various health checking and verification tools available for checking the cluster health during cluster bring-up, maintenance and problem debug time. These tools are developed to check the health of individual cluster components or group of cluster components. At any time, these individual tools or a group of tools can be executed to check or verify the health of the cluster components. The health check tools are grouped into different groups based on health check functionality or components tested by them. There are separate individual configuration files for each tool customizing the tool to support different cluster environments.

The hcrun command is used to run cluster health checking tools interactively. The xCAT command xdsh, is used to run health checking tools on the specified set (node range) of nodes. If nodes are not specified then the health checking tools are executed on the local node (Management Node) based on configuration of the tool. If tool is configured to run on multiple nodes but did not provide the nodes to run then it would be in error. The hcrun command accepts either a single health check tool with its command line arguments or a list of health check tools

separated by commas. If a list of health check tools are specified then the command line arguments specified in configuration files of individual tools are used while executing respective tools and the tools also get executed in the order they are specified.

The hcrun also accepts a list of health checking tool group names separated by commas. If group names are specified all the tools of the individual groups are executed in an order specified in the group configuration file until either the first health check tool fails or all the tools are executed successfully. Even though the tools of a group are executed in the order the tools are specified in the group configuration file, the pre-check tools of individual tools are always executed before the individual tool is executed. If -c (continue) option is specified then all the health check tools of the groups are executed even if one or more health check tools fail. By default, the execution of health check tools of a group are stopped once a tool fails. If multiple groups are specified then the tools of the groups are executed in the order the groups are specified on the hcrun command line.

The cluster health check tools of the groups are executed on the same nodes specified on hcrun command line. The tools are executed using xdsh command. If any tool of a group should not be started using the xdsh command then the configuration keyword "StartTool" of that tool should be set to "local", in which case that particular tool is started on local node (Management Node) only. For any reason, if a particular tool wants to know the set of nodes (node range), it can access the environment variable **HC_NODERANGE** which is always set by hcrun command to the node range specified to hcrun command.

The hcrun command saves the cluster health checking tools results in the file specified by keyword "toolsoutput" in the master configuration file. These results are used for historical analysis. The hcrun command also saves the summary of the cluster health checking tools in a summary file specified by the keyword "summaryfile". The summary file has brief details of the cluster health checking tools like start and end times of the tools along with final status (Fail/Pass) of the tools. In addition to storing the tools results in the file specified by the keyword "toolsoutput", it also displays the health checking tools results to the stdout.

The output of the tools can also be processed using different utility tools like the xcoll command. By default the health checking tools are run in plain mode where the results of the tools are displayed. The hcrun command also supports two other methods "xcoll" and "compare". When xcoll is used the tools output is piped through the xcoll command of xCAT before display, where the xcoll analyzes the tools output and segregate the nodes which have common output. This would help to identify the set of nodes on which the health check is passed and on which health check is failed, or to reveal unexpected inconsistencies.

When "compare" method is used the output of health check tools from each node is compared against a template (baseline output). If output from one or more nodes does not match with the baseline template, alternative templates along with the nodes matching the alternative templates are displayed. This method can be used to find out the nodes which are deviating from the known baseline. The health check is considered to have passed if the tool output from all the given nodes are matched with the template, else the health check is considered to have failed.

OPTIONS

-h The small usage information about the hcrun command is displayed.

-v The hcrun command and cluster health checking tools are executed in verbose mode. The environment variable HC_VERBOSE is set to 1 and exported to health checking tools if hcrun is run with option (-v). By default the HC_VERBOSE is set to 0. To support verbose mode each cluster health checking tool should verify the HC_VERBOSE environment variable. If HC_VERBOSE=1, then the command should run in verbose mode.

-c The execution of health checking tools of a group are to continue running, even if one or more cluster health checking tools of the group fails. This does not apply to any "pre-checks", which are considered to be required to pass.

-l The command lists all the configured cluster health checking tools according to their types. The defined tool groups also listed along with their member tools and other attributes. If –l option along with one or more tool names are specified then all attributes defined for the tools are listed. If –l option along with one or more groups are specified then the tools of the specified groups and other attributes of the groups are listed.

-n *<NODERANGE>*
The cluster health checking tool specified or the cluster health checking tools of the specified group are executed on the nodes specified by NODERANGE. The syntax of the node range is the same as supported by xCAT. The detailed explanation of NODERANGE syntax can be found at xCAT document available at http://xCAT.sourceforge.net/man3/noderange.3.html . An environment variable HC_NODERANGE is set with node range specified and exported to health checking tools.

-f Generally (by default) the execution of the health checking is stopped if one or

more nodes in the node range specified are not reachable. If –f (force) option is specified then the execution of health checking is continued even if one or more nodes are not reachable.

group[,group,...]

One or more group names separated by comma, whose cluster health checking tools are executed. The tools of the groups are executed in the order the groups specified. The tools within a group are also executed in the order the tools defined in that group.

-p If groups are specified on command line, the tools of the individual groups are executed in the order the tools defined in respective groups. If –p (preview) option along with groups is specified then the health checking tools are listed in the order they would get executed but the tools actually would no be executed.

-t tool[,tool,...]

The tools specified are executed. If either only tool name is specified without any command line arguments required for that tool or list of tools are specified then the default command line arguments configured in configuration files of the tools are used while executing the respective tools. If tool name along with command line arguments are specified then the tool is executed using the command line arguments specified by overriding default command line arguments configured in configuration file.

<cmd_args>

The command line arguments passed to the tool when single health check tool name is specified.

-m <process_method>

The methods used to post process the output of the tools. The supported methods are "xcoll", "xcoll_diff", and "compare". When "xcoll" method is used the tools output is piped through the xcoll command of xCAT before display, where the xcoll command analyzes the tools output and segregates the nodes which have common output. If "xcoll_diff", is used, the tools output is piped through xcoll and then compare_results to indicate the differences in results between various nodes. If the "compare" method is used, the output of health check tools form each node is compared against a template (baseline output). If output from one or more nodes does not match with the baseline template, alternative templates along with the nodes matching the alternative templates are displayed. The hcrun uses the utility command config_check (See section **2.2 The compare_results command**) to compare the tools results with baseline.

For more information on processing methods, see Error: Reference source not foundError: Reference source not found1.4 Tool output processing methods" on page 13.

-s <seed_node>

A seed node along with processing method "compare" is used, first the baseline template is created by running tool on the seed node. Then the output of health check tools form each node is compared against template (baseline output) generated. If output from one or more nodes does not match with baseline template, alternative templates along with the nodes matching the alternative templates are displayed. The hcrun uses the utility command config_check (See section **2.2 The compare_results command**) to compare the tools results with baseline.

When the processing method "xcoll_diff" is used, the seed node's results are considered to be the base results for the difference calculation in compare_results – as opposed to using the results from the largest number of nodes.

EXAMPLES

1. To list all the health check tools available run the hcrun command without passing any arguments as shown below.

```
e119f3ems2:~ # hcrun
config_check          : Util tool used to compare the query attributes of nodes against a seed node
switch_module          : Query the switch modules installed
run_ppping            : Runs ppping interconnect test
ibdiagcounters_clear    : Util tool to clean IB diag counters
fs_usage              : Checks the usage of the provided filesystems
run_nsdperf            : Runs nsdperf interconnect test
switch_health          : Checks the switch health report
syslogs_clear          : Util tool to clean the syslogs
firmware              : Checks the BMC and UEFI firmware of the node
run_jlink             : Runs jlink interconnect test
hca_basic              : Checks basic IB fabric parameters
file_rm               : Util tool to remove files copied to the execution node
nfs_mounts             : Checks nfs mounts
memory                : Checks the Total memory on the node
jlinksuspects          : Correlates sub-par jlink BW with BER calculations per Subnet Manager(s)
jlinkpairs            : Gets all jlink pairs whose BW is below threshold
hostfile_copy          : Util tool to copy hostfile to the execution node
lastclear             : Returns timestamp from last clearerrors on the Subnet Manager(s)
clearerrors            : Clears the IB error counters on the Subnet Manager(s)
switch_inv             : Query the switch inventory
switch_clk             : Checks the switch clock consistency
run_daxpy              : Runs daxpy test on node(s)
run_dgemm              : Runs degemm test on the node(s)
leds                  : Checks the leds of the node
ibtools_install        : Installs IB Tools on the Subnet Manager(s)
temp                  : Checks all temperatures
gpfs_state             : Checks the gpfs state
ibtoolslog_clean        : Archives the ibtools logs on the Managment Node and Subnet Manager(s)
switch_code            : Checks to see if the switch code is consistent
ipoib                 : Query IPoIB settings
hostfile_rm            : Util tool to remove hostfile to the execution node
ibqber                : Runs an IB query and BER calculation on the Subnet Manager(s)
switch_ntp             : Checks to see if the switch ntp is enabled
os                    : Checks the OS and kernel version
cpu                   : Checks for the CPU status
berlinks               : Gets the links from a BER calculation on the Subnet Manager(s)
file_copy              : Util tool to copy required files to the remote node
```

```
e119f3ems2:~ #
```

DESCRIPTION

Each tool name, followed by a colon, and followed by a short description of the tools is displayed.

2. To list the tools as per their type and the tool groups in which they are members, run the following command.

```
e119f3ems2:~ # hcrun -l
The Tool Type   : The Tools          : Tool Description
================================================================
node         : firmware           : Checks the BMC and UEFI firmware of the node
               temp               : Checks all temperatures
               run_daxpy          : Runs daxpy test on node(s)
               nfs_mounts         : Checks nfs mounts
               fs_usage           : Checks the usage of the provided filesystems
               leds               : Checks the leds of the node
               os                 : Checks the OS and kernel version
               run_dgemm          : Runs degemm test on the node(s)
               gpfs_state         : Checks the gpfs state
                        memory                  : Checks the Total memory on the node
                        cpu                     : Checks for the CPU status

tools_util       : config_check            : Util tool used to compare the query attributes of nodes against
node
                   file_rm                 : Util tool to remove files copied to the execution node
                   syslogs_clear           : Util tool to clean the syslogs
                   file_copy               : Util tool to copy required files to the remote node
                   hostfile_copy           : Util tool to copy hostfile to the execution node
                   hostfile_rm             : Util tool to remove hostfile to the execution node
                   ibdiagcounters_clear    : Util tool to clean IB diag counters

ib               : switch_ntp              : Checks to see if the switch ntp is enabled
                   ipoib                   : Query IPoIB settings
                   switch_health           : Checks the switch health report
                   switch_module           : Query the switch modules installed
                   hca_basic               : Checks basic IB fabric parameters
                   switch_inv              : Query the switch inventory
                   switch_code             : Checks to see if the switch code is consistent
                   switch_clk              : Checks the switch clock consistency

interconnect     : run_ppping              : Runs ppping interconnect test
                   run_jlink               : Runs jlink interconnect test
                   run_nsdperf             : Runs nsdperf interconnect test

tools_ib         : jlinkpairs              : Gets all jlink pairs whose BW is below threshold
                   ibtools_install         : Installs IB Tools on the Subnet Manager(s)
                   clearerrors             : Clears the IB error counters on the Subnet Manager(s)
                   lastclear               : Returns timestamp from last clearerrors on the Subnet Manager(s)
                   ibtoolslog_clean        : Archives the ibtools logs on the Managment Node and Subnet Manag
)
                   jlinksuspects           : Correlates sub-par jlink BW with BER calculations per Subnet Ma
                   berlinks                : Gets the links from a BER calculation on the Subnet Manager(s)
                   ibqber                  : Runs an IB query and BER calculation on the Subnet Manager(s)

================================================================

The Group Name  : Group Attributes                  : Attributes Values
================================================================
=========
node_test        : Description                      : These are intrusive node health testing tools.
                   Environment                      :
                   Tools                            : ['run_dgemm', 'run_daxpy']
```

```
node_check      : Description                       : This group has node check tools
                  Environment                       :
                  Tools                             : ['cpu', 'memory', 'gpfs_state', 'leds', 'temp', 'nfs_m
'fs_usage', 'os']

ib_check        : Description                       : This group has ib check tools
                  Environment                       :
                  Tools                             : ['ipoib', 'hca_basic', 'switch_code', 'switch_module',
'switch_clk', 'switch_health', 'switch_ntp']

interconnect_test: Description                       : These are intrusive interconnect health  testing too
                  Environment                       :
                  Tools                             : ['run_ppping', 'run_jlink', 'run_nsdperf']

run_ibtools     : Description                       : Runs jlink with a tools_ib pipeline
                  Environment                       :
                  Tools                             : ['clearerrors', 'lastclear', 'run_jlink', 'ibqber', 'b
'jlinksuspects', 'ibtoolslog_clean']

e119f3ems2:~ #
```

DESCRIPTION

First the tools are listed as per their type. The configured types are node, tools_util, ib, interconnect, and tools_ib. Then the configured groups node_test, node_check, ib_check, interconnect_test, and run_ibtools are displayed along with their member tools.

3. The attributes of a tool can be displayed using hcrun command as shown below.

```
e119f3ems2:~ # hcrun -l -t cpu
Tool Name    : Tool Attributes          : Attributes Values
================================================================
cpu          : Executable               : /opt/ibmchc/x86_64/tools/node/cpu
               Arguments                :
               Environment              :
               Description              : Checks for the CPU status
               Precheck Test            : []
               Predata Collection Test  : []
               Post Data Collection Test: []
               Data Processing Method   : ['xcoll', 'compare']
               Copy                     : yes
               Start Tool               : all
               Groups                   : ['node_check']
         Type              : node
         Pass Exit Codes       : ['0']
         Fail Exit Codes       : []
         Warning Exit Codes    : []
         Error Exit Codes      : ['128']

e119f3ems2:~ #
```

DESCRIPTION

The tool “cpu” along with its various attributes is displayed.

4. The attributes of configured tool group can be listed as shown below using hcrun command.

```
e119f3ems2:~ # hcrun -l node_test
The Group Name  : Group Attributes          : Attributes Values
================================================================
==========
node_test      : Description              : These are intrusive node health testing tools.
              Environment              :
              Tools                    : ['run_dgemm', 'run_daxpy']

e119f3ems2:~ #
```

DESCRIPTION

The tool group "node_test" along with its various attributes is displayed.

5. The health check tool "cpu", which query different attributes of "cpu" and display is run using hcrun command as shown below.

```
e119f3ems2:~ # hcrun -n e119f4m1n0[4-5] -v -t cpu

e119f4m1n04: CPU Model:              Intel(R) Xeon(R) CPU          5160  @ 3.00GHz
e119f4m1n04: Turbo HW:         Off
e119f4m1n04: Turbo Engaged:           No
e119f4m1n04: HyperThreading HW:        Disable
e119f4m1n04: Socket(s):              Core(s) per socket:     2
e119f4m1n04: Active Cores:            4
e119f4m1n04: scaling_governor:  ondemand
e119f4m1n04: scaling_max_freq:  2997000
e119f4m1n04: scaling_min_freq:  1998000
e119f4m1n05: CPU Model:              Intel(R) Xeon(R) CPU          5160  @ 3.00GHz
e119f4m1n05: Turbo HW:         Off
e119f4m1n05: Turbo Engaged:           No
e119f4m1n05: HyperThreading HW:        Disable
e119f4m1n05: Socket(s):              Core(s) per socket:     2
e119f4m1n05: Active Cores:            4
e119f4m1n05: scaling_governor:  ondemand
e119f4m1n05: scaling_max_freq:  2997000
e119f4m1n05: scaling_min_freq:  1998000
The health check tool  cpu                        [ PASS ]
e119f3ems2:~ #
```

DESCRIPTION

As shown from health check tool "cpu" output, various attributes from nodes e119f4m1n04, and e119f4m1n05 are displayed.

6. To process the health check tools output using method "xcoll" run the hcrun command as shown below.

```
e119f3ems2:~ # hcrun -n e119f4m1n0[3-7] -m xcoll  -t cpu
====================================
e119f4m1n06,e119f4m1n03,e119f4m1n05,e119f4m1n04,e119f4m1n07
====================================
CPU Model:            Intel(R) Xeon(R) CPU          5160  @ 3.00GHz
Turbo HW:            Off
Turbo Engaged:         No
```

```
HyperThreading HW:      Disable
Socket(s):              Core(s) per socket:     2
Active Cores:           4
scaling_governor:       ondemand
scaling_max_freq:       2997000
scaling_min_freq:       1998000

The health check tool  cpu                      [ PASS ]
e119f3ems2:~ #
```

DESCRIPTION

The results from health check tool "cpu" is processed by xcoll method and summarized. Here all the nodes, e119f4m1n06, e119f4m1n03, e119f4m1n05, e119f4m1n04, and e119f4m1n07 have same results returned from "cpu" health check tool.

7. To over-write the default arguments defined for a health check tool run the hcrun command by specifying the command line arguments to the tool as shown below.

```
e119f3ems2:~ # hcrun -n e119f4m1n0[3-7] -c  -m xcoll  -t  fs_usage
======================================
e119f4m1n06,e119f4m1n03,e119f4m1n05,e119f4m1n04,e119f4m1n07
======================================
/tmp: 7%
/var/tmp: 7%

The health check tool  fs_usage                 [ PASS ]
e119f3ems2:~ # hcrun -n e119f4m1n0[3-7] -c  -m xcoll  -t  fs_usage /tmp
======================================
e119f4m1n06,e119f4m1n03,e119f4m1n05,e119f4m1n04,e119f4m1n07
======================================
/tmp: 7%

The health check tool  fs_usage                 [ PASS ]
e119f3ems2:~ #
```

DESCRIPTION

As shown above the second execution of health check tool "fs_usage" uses the argument (/tmp) passed on the command line. The first execution uses the default arguments defined in the configuration file.

8. To process the health check tool group results using method "xcoll" run the hcrun command as shown below.

```
[root@c933mnx01 node]# hcrun -n c933f01x31,c933f01x33 -c -m xcoll node_check
==================================
c933f01x33
==================================
CPU Model:              Intel(R) Xeon(R) CPU          X5560  @ 2.80GHz
Turbo HW:               Off
Turbo Engaged:          No
HyperThreading HW:      Enable
Socket(s):              Core(s) per socket:     4
Active Cores:           16
scaling_governor:       performance
```

scaling_max_freq: 2793000
scaling_min_freq: 1596000

==========
c933f01x31
==========
CPU Model: Intel(R) Xeon(R) CPU X5560 @ 2.80GHz
Turbo HW: Off
Turbo Engaged: No
HyperThreading HW: Enable
Socket(s): Core(s) per socket: 4
Active Cores: 16
scaling_governor: ondemand
scaling_max_freq: 2793000
scaling_min_freq: 1596000

The health check tool cpu [PASS]
==========
c933f01x33,c933f01x31
==========
Memory 32097 MB

"2048 MB","DIMM01","BANK01","800 MHz"
"2048 MB","DIMM02","BANK02","800 MHz"
"2048 MB","DIMM03","BANK03","800 MHz"
"2048 MB","DIMM04","BANK04","800 MHz"
"2048 MB","DIMM05","BANK05","800 MHz"
"2048 MB","DIMM06","BANK06","800 MHz"
"2048 MB","DIMM07","BANK07","800 MHz"
"2048 MB","DIMM08","BANK08","800 MHz"
"2048 MB","DIMM09","BANK09","800 MHz"
"2048 MB","DIMM10","BANK10","800 MHz"
"2048 MB","DIMM11","BANK11","800 MHz"
"2048 MB","DIMM12","BANK12","800 MHz"
"2048 MB","DIMM13","BANK13","800 MHz"
"2048 MB","DIMM14","BANK14","800 MHz"
"2048 MB","DIMM15","BANK15","800 MHz"
"2048 MB","DIMM16","BANK16","800 MHz"

The health check tool memory [PASS]
==========
c933f01x33,c933f01x31
==========
fsB : OK
verbsRDMA : OK

The health check tool gpfs_state [PASS]

c933f01x33: LED 0x0000 (Fault) active to indicate system error condition.
c933f01x33: LED 0012 (PS) active to indicate Sensor 0x71 (Power Supply 2) error.
rvitals FAIL!
The health check tool leds [FAILED]
==========
c933f01x33
==========
Ambient Temp: 31 C (88 F)

==========
c933f01x31
==========
Ambient Temp: 30 C (86 F)

The health check tool temp [PASS]
==========
c933f01x33,c933f01x31
==========
ugpfs01-ug:/gpfs/cnfs01/data/u
ugpfs01-ug:/gpfs/cnfs01/data/admin

```
The health check tool  nfs_mounts                [ PASS ]
====================================
c933f01x33,c933f01x31
====================================
BMC Firmware: 1.35 (YUOOE9B 2012/11/29 09:14:45)
          UEFI Version: 1.16 (D6E158A 2012/11/26)
The health check tool  firmware                  [ PASS ]
====================================
c933f01x33,c933f01x31====================================/tmp: 2%
/var/tmp: 2%
The health check tool  fs_usage                  [ PASS ]
====================================
c933f01x33,c933f01x31
====================================
Description:    Red Hat Enterprise Linux Server release 6.3 (Santiago)
Release:        6.3
Codename:       Santiago
Kernel:         2.6.32-279.el6.x86_64

The health check tool  os                        [ PASS ]
[root@c933mnx01 node]#
```

9. To process the health check tool group results using method “xcoll_diff” run the hcrun command as shown below, which shows results from both nodes match

```
[root@c931mnp01 node]# hcrun -n c931f07p01,c931f07p02 -v -c -m xcoll_diff node_check

The health check tool  cpu                       [ PASS ]
```

The above results indicate that both nodes passed to hcrun had the same results. Note that the -v was used to assure that verbose output was given by the cpu tool. Otherwise, there would be nothing to compare.

10. To process the health check tool group results using method “xcoll_diff” run the hcrun command as shown below, which shows results from the two nodes differ.

```
[root@c931mnp01 node]# hcrun.mga -n c931f07p01,c931f07p02 -v -c -m
xcoll_diff -t fs_usage



################################################################################
#######

Differences between:

  c931f07p01 (nodegroup1)

    AND

  c931f07p02 (nodegroup2)

################################################################################
#######
```

```
- /tmp: 2%

?       ^

+ /tmp: 1%

?       ^

Changed from nodegroup1 to nodegroup2

====================

- /var/tmp: 2%

?           ^

+ /var/tmp: 1%

?           ^

Changed from nodegroup1 to nodegroup2

====================


The health check tool  fs_usage                          [ PASS ]
```

Note: The fs_usage tool passed because the filesystems being checked weren't at warning or error levels. However, xcoll_diff reported a difference in results being reported by the two nodes. xcoll_diff used the largest number of nodes with the same result, but because only two nodes were being checked, both results groups had only one node in them. xcoll_diff breaks the tie by using the first one to report back from xcoll. You can see that the difference between the two node's results is that one has 1% filesystem use and the other has 2%. Note, that the carat (^) is under the 1 and the 2 in the results, which helps you see the differences quickly.

2.2 The compare_results command

NAME

compare_results - Runs health checks and compares the output against a baseline

PURPOSE

Runs the commands from the command file on a single node or range of nodes, and node groups. The output from each node is compared with a baseline template or output collected from a seed node. Differences are highlighted. It is intended as a utility or wrapper for health check commands/scripts, but can be used stand-alone by taking output piped from xcoll.

Note: It is very useful when the results should have the same strings every time. It does not work with ranges of results or real expressions.

SYNTAX

```
compare_results -f|--file <filename> [-s|--seednode <seednode>] [-b|--baseline <baseline>] [-K|--keepbaseline] -c|--command <command> [-m|--mode <command mode>] [-l|--log <log directory>] [-v|--verbose] [--veryverbose]-
```

Usage: Format: compare_results [option]

OPTIONS

```
 -h, --help            show this help message and exit
 -f FILENAME, --file=FILENAME
                       A file that contains results from xcoll. Default is to
                       use STDIN.
 -s SEEDNODE, --seednode=SEEDNODE
                       The node that is considered to be the example against
                       which the others are compared.
 -b BASELINE, --baseline=BASELINE
                       A file that contains the results against which to
                       compare.

 -K, --keepbaseline    Keep the old copy of the baeline. It will re
                       extension '[last modification tim]-[current
                       default=don't keep

 -c COMMAND, --command=COMMAND
                      To run against remote nodes, the command must be able
```

to do so on its own.

-m COMMAND_MODE, --mode=COMMAND_MODE
The mode to use: None| xcoll = pipe through xcoll; xdshcoll = pipe through xdshcoll (doesn't collapse to groups); noxcoll = don't pipe through xcoll

-l DIRECTORY, --log=DIRECTORY
Log directory

-v, --verbose Verbose mode. Default is quiet except on error. Environment variable HC_VERBOSE=1 will also turn it on. Command line on overrides HC_VERBOSE=0.

--veryverbose Very verbose mode. Shows progress of the program. This includes verbose output.

DESCRIPTION

The compare_results utility command is used to run commands from a given command file or script on the specified node(s) or node groups. The output generated from commands of command file is compared using the xCAT command xcoll against the baseline/template, whose name is specified (typically by hcrun). While comparing the results with the baseline, if the output from one or more nodes does not match the baseline, then alternate results are generated which contain those varying results. The alternative results generated are temporarily saved in /tmp The alternative resultsare named by appending a sequence number, starting from 1, to the basename which is compare_results. $PID. The alternate results are removed as soon as the test is completed. The output log describes the differences from the baseline.

The compare_results command can also be used to generate new baselines by running the commands from command file on the seed node specified and to compare the output from all the nodes or node groups specified with the new baseline created. The compare_results command uses the baseline filename provided on the command line in -b|--baseline. If no baseline is given, then one will be temporarily chosen based on either the node indicated in -s|--seednode input, or based on the like results from the largest number of nodes.

The commands specified in the command file are run on nodes or node groups using the xCAT command xdsh. If the commands specified in command file are remote execution commands like rinv, rvitals then the commands of the command file have to be run locally on Management node. By default, the compare_results command displays pass/fail health check by comparing the health check output with baseline, or against the seed_node. It can be run in verbose mode as well to display the details of the heath check result. The veryverbose mode will show the details like verbose, and it will also show the commands progress. The details or logs of the health checks also saved in a log file in directory specified by environment variable HC_LOGDIR. You can use the

`compare_results` command stand-alone (outside of hcrun) by manipulating the command line arguments and environment variables.

There are several different processing modes:

- None = no special processing; the input should have been piped through xcoll before handing it over to compare_results.
- xcoll = pipe input through xcoll before processing
- xdshcoll = pipe input through xdshcoll before processing (this doesn't collapse results groups into xCAT nodegroups, it lists each node individually)
- noxcoll = don't pipe through xcoll; same results as None. This is used by hcrun when -m xcoll or -m xcoll_diff is requested, or if -m compare is requestsed and noxcoll is in the tools processing methods, or xcoll is not in the processing methods of the tool (as defined in the tool's configuration file.)

ENVIRONMENT VARIABLES

The following environment variables are used by compare_results:

- HC_LOGDIR = where to log results
- HC_NODERANGE = node range to use
- HC_VERBOSE = 1 = verbose; 0 = no verbose
- HC_VERYVERBOSE = 1 = veryverbose; 0 = no veryverbose; veryverbose includes verbose output

EXAMPLES

- You can pipe the results from xCAT xcoll into compare_results:
  xdsh … |xcoll|compare_results
- You can point the command to a file that has results from xcoll
  xdsh … | xcoll > FILE
  compare_results -f FILE
- You can have the compare_results run a command to all nodes in a node group and compare against the unique results from the largest number of nodes. The example runs /opt/ibmchc/ppc64/node/cpu against f02

```
compare_results -c 'xdsh f02 -v -e
/opt/ibmchc/ppc64/node/cpu'
```

-
- You can have the compare_results run a command to all nodes in a node group and compare against the unique results from a seed node:

```
compare_results -c 'xdsh f02 -v -e
/opt/ibmchc/ppc64/node/cpu'-s f02n05
```

- You can have the compare_results run a command to all nodes in a node group and compare against the unique results from a seed node and save the seed node's results as a baseline:

```
compare_results -c 'xdsh f02 -v -e
/opt/ibmchc/ppc64/node/cpu' -b
/var/opt/ibmchc/data/f02.cpu.baseline
```

- You can have the compare_results run a command to all nodes in a node group and compare against a saved baseline:
- You can have the compare_results run a command that does xdsh and xcoll against a node group and compare against the unique results from the largest number of nodes:

EXCEPTIONS

Exception	**Comments**
Cannot open <file>. The results input file must exist.	The results input file given in -f can't be opened. It doesn't exist, or there's an exception problem.
Cannot open <file> for write. Does the path exist? Maybe the filesystem is full? Is there a permissions problem with the path?	This is when an output file, like the baseline or log file, cannot be written.
There were no results groups found. Perhaps the nodegroup or seed for the command is not correct. Perhaps the input was not piped through xcoll? If using a command (-c), make sure it returns in xdsh format. If using -m noxcoll with a command, make sure the command returns xcoll format. Exiting.	With hcrun, the most likely issue is that the output of the command provides xcoll output and 'noxcoll' was used as the processing method, or vice-versa.
The baseline must exist, unless you supply a seed node. <baseline> Re-run with a seed node.	You attempted to check against a baseline and none exists.
Two different nodes in the base line: <node1> and <node2> Re-run so that only one seed node is found.	This typically happens when you are comparing between nodes that are configured differently; either by design or because of a config problem
<node> is not in nodegroup '<nodegroup>' of the results group. Try again, making sure that <node> is a member of the results group. Exiting.	This typically happens when you have provided a seednode that is not in the nodegroup in hcrun, or in the command passed via -c to compare_results.
Target node <node> does not match what's in the baseline: <baseline node> It may be that the command is calling a different nodegroup than the baseline did.	

Exception	Comments
Baseline results can't have more than one results group. Make sure that you get only one node's or nodegroup's results from the command. You should fix the discrepancies with the non-seed nodes or the seed node, first. Exiting.	You can't save a baseline when there are mismatches in the nodegroup's configuration. Fix those first.
Cannot open temp file "<file>". - Is /tmp full?' % file)	The temp files are used to store results before they are compared. Typically the problem is that /tmp is full, but it could be a permissions change to /tmp.
Command returned nothing. Please review: <command string>	This is usually a problem with the command that was passed via -c, or as a tool called by hcrun.

2.3 Using `compare_results` and `hcrun` to run health checks

The compare_results utility tool is developed to run under the hcrun command, which exports the HC_TEMPLATESDIR and HC_LOGDIR specified in the master configuration file of hcrun. The baseline name is automatically generated from the health check tool name and health check tools group name, when run under `hcrun` command.

The compare_results tool is used when the “compare” or “xcoll_diff” processing methods are called for by the user in the -m parameter. The “compare” method uses baselines, and the “xcoll_diff” method does a one time compare of results from the nodegroup.

When “compare” is used and a “seed_node” is given, a baseline is generated. If a single tool is called for `<templates_dir>/<tool name>.<nodegroup>`
If a tools group is called for the baseline filename is `<templates_dir>/<tool name>[.<tools group>].<nodegroup>`
The default template directory is /opt/ibmchc/conf/templates.
If a baseline already exists, the old baseline is copied over with an extension that indicates the last modified timestamp and the current timestamp: <Yymmdd_HHMMSS>-<YYmmdd_HHMMSS>. This allows you to see the history of baseline files and the time over which they were used.

The following steps have to be followed to run health check using compare_results under hcrun command.

- Create a script that has a consistent output on a given set of nodes. Sometimes this can be a simple wrapper for a command, and may involve using grep -v to remove unique parameters, such as IP addressing.
- Create the hcrun configuration file for the health check tool and define the processmethod of the health check tool as "compare" and possibly "xcoll_diff" as shown in the Example 3-1 below:

Example 3-1 /opt/ibmchc/tools/node/fs_usage.conf

```
[HealthCheckTool]
name=fs_usage
description=Checks the usage of the provided filesystems
type=node
executable=/opt/ibmchc/tools/node/fs_usage
arguments=/tmp /var/tmp
copy=yes
starttool=all
processmethod=xcoll,compare
passexitcode=0
errorexitcode=128
```

- The first time you run compare_reults against a node group via hcrun -m compare, a seed node has to be specified to create a baseline based on the results from the seed node. (-s <seed_node>) On the first run, you should use verbose mode in hcrun (-v). You should check the baseline that is output (the name will be indicated in the STDOUT), and be sure that the results are as expected. If you don't do this, you will at least know if all of the nodes in the node group get consistent results relative to each other, because compare_results will indicate if there's inconsistency and will not save the baseline until all nodes in the nodegroup give the same result. For other possible error messages, see the table of error messages in Error: Reference source not found, on page Error: Reference source not found.
- In subsequent runs, you may use verbose or not, as long as the tool displays the data. If there is a failure, you can find the latest output file listed in STDOUT, which will indicate where there are inconsistencies from the baseline.
- If all nodes mismatch the baseline, and you agree with the new values, you will want to re-run with the seed node, so that a new baseline template is generated. This will be used for comparison on all new runs, and the old baseline will be saved for future reference.

EXAMPLES

1. To run the health check tool “cpu” using compare_results under hcrun command, run the following command:

   Note: The hcrun calls the compare_results utility when method “compare” is specified.

```
e119f3ems2:~ # /opt/ibmchc/bin/hcrun -n f07- m compare -t cpu
The health check tool  cpu                    [ PASS ]
e119f3ems2:~ #
```

2. To run the health check tool “cpu” using coompare_results under hcrun command in the verbose mode, run the following command:
3.

   Note: The hcrun calls the compare_results utility when method “compare” is specified.

```
e119f3ems2:~ # /opt/ibmchc/bin/hcrun -n f07 -m compare -v -t cpu
Using baseline file /opt/ibmchc/conf/templates/cpu.f07.template

Everything matched

Log file in "/var/opt/ibmchc/log/cpu..f07/compare_results.log.20140227_144208"


The health check tool  cpu                    [ PASS ]
e119f3ems2:~ #
```

DESCRIPTION

The compare_results tool will indicate whether nodes match or do not match the baseline template, and the output file that contains the details.

The example indicates that all nodes in the nodegroup (f07) matched the baseline results.

This output line indicates where the detailed output file is to be found:

```
Using baseline
Log file in
"/var/opt/ibmchc/log/cpu.node_check.compute.f07/compare_results.log.20140227_144
208"
```

When run under hcrun, the above details are only output in verbose mode. In either case, an indication of pass/fail is output:

```
The health check tool  cpu                    [ PASS ]
e119f3ems2:~ #
```

4. To run the health check tool “cpu” and get a baseline, do something similar to the following, which uses f07n01 as the seed node:

```
e119f3ems2:~ # /opt/ibmchc/bin/hcrun -n f07 -m compare -s f07n01 -v -t cpu
Baseline created in "/opt/ibmchc/conf/templates/cpu.f07.template"
The health check tool  cpu                          [ PASS ]
e119f3ems2:~ #
```

DESCRIPTION

The -s f07n01 indicates the seed node, or the node that is considered the example for the nodegroup.

The output indicates that a baseline was created and where it was created.

The output indicates that the command was successful. If there were nodes in f07 that didn't match the output of f07n01, the command would fail, and indicate which output lines were mismatching and where they were mismatching.

5. The following indicates a mismatch in the output vs. that in the baseline:

```
e119f3ems2:~ # /opt/ibmchc/bin/hcrun -n f07 -m compare   -v  -t   cpu
Using baseline file /opt/ibmchc/conf/templates/cpu.f07.template


##########################################################
############################

Differences between:

 f07n01 (nodegroup1)

   AND

 f07n02(nodegroup2)

##########################################################
############################



- On-line CPU(s) list:   0-63

?                    ^

+ On-line CPU(s) list:   0-62

?                    ^

Changed from nodegroup1 to nodegroup2

======================
```

```
Log file in
"/var/opt/ibmchc/log/cpu.node_check.compute.f07/compare_results.log.20140227_144
334"
The health check tool  cpu                        [ FAIL ]
e119f3ems2:~ #
```

DESCRIPTION

The baseline file being used is indicated.
There are differences found between f07n01 (the baseline seed node) and f07n02. The nicknames nodegroup1 and nodegroup2 are used. This makes more sense when there is a large list of nodes in one or both of the nodegroups. These are not xCAT nodgroups. This is just a nickname used by compare_results.
In this case, the difference is that f07n02 has 62 instead of 63 online CPUs. hcrun indicates a FAIL.

6. The following is an example of running -m compare with a tools group instead of a single tool. In this case, the tools group “node_check” is used.

```
e119f3ems2:~ # /opt/ibmchc/bin/hcrun -n f07 -m compare  -v node_check
Using baseline file /opt/ibmchc/conf/templates/cpu.node_check.f07.template

The health check tool  cpu                        [ PASS ]
Using baseline file /opt/ibmchc/conf/templates/memory.node_check.f07.template

The health check tool  memory                     [ PASS ]
Using baseline file /opt/ibmchc/conf/templates/ipoib.node_check.f07.template

The health check tool  ipoib                      [ PASS ]
Using baseline file /opt/ibmchc/conf/templates/hca_basic.node_check.f07.template

The health check tool  hca_basic                  [ PASS ]
Using baseline file /opt/ibmchc/conf/templates/os.node_check.f07.template

The health check tool  os                         [ PASS ]
Using baseline file /opt/ibmchc/conf/templates/fs_usage.node_check.f07.template

The health check tool  fs_usage                   [ PASS ]
e119f3ems2:~ #
```

DESCRIPTION

Each tool in the node_check tools group is run individually, and you see that all of them PASS. In each case, the baseline is indicated for each individual tool. Notice that the node_check tools group name is included in the baseline filename.

7.

3.0 Guidelines for adding new tools

There are few simple guide lines that need to be followed while developing the Cluster Health Checking Tools which will be run using `hcrun` command. They are:

1. Each cluster health checking tool should have its own configuration file specifying different attributes of the tool.
2. In general, the tools should exit with zero on successful and non-zero on failure. If not, then the exit codes of the tool need to be configured in the configuration file of the tool with exact exit codes for success, success with warning and failure.
3. If cluster health checking tools exit with a different exit code for different conditions like success, failure, and warning, then those exit codes need to be configured in the configuration file.
4. The cluster health checking tools should check the environment variable HC_VERBOSE to run health checking tool in verbose mode. If HC_VERBOSE=1, then cluster health checking tool should run in verbose mode.
5. In case of errors or warnings, the messages should be printed to "`stderr`" which `hcrun` command grabs and logs in the summary file along with exit codes. The `hcrun` also saves the stderr output along with stdout in the file defined by master configuration keyword "`toolsoutput`".
6. If the tool is developed to start execution only on lthe ocal node then the keyword "`startTool`" should be set to "`local`" in the configuration.
7. If the tool's binary/script is available on all compute nodes at a specified location, then it is not required to copy the binary to compute nodes before starting the execution. The configuration keyword "`copy`" could be set to "`no`" so that `hcrun` would not copy the binary/script to the compute nodes before starting execution.
8. The cluster health checking tools can access HC_NODERANGE environment variable to know the set of nodes user specified to `hcrun` command.
9. The cluster health checking tools can access HC_HOSTFILE environment variable to know the host file generated from the node range. There will be one hostname per line in the host file generated from node range.
10. The cluster health checking tools can also use HC_USER environment variable to access the user id specified in Master configuration file. The HC_USER is generally used for specifying non-root user id. The tools which want to use non-root user id during execution can use the user id defined by HC_USER. This is necessary for tools that use PE runtime.
11. The cluster health checking tools which want to use the hosts where UFM servers (or other subnet manager servers) are running can access the environment variable HC_SUBNET_MANAGERS. The environment variable HC_SUBNET_MANAGERS is set with subnet managers specified in Master Configuration file.
12. The cluster health checking tools which wanted to use the IB switches xCAT node group can access the environment variable HC_IB_SWITCHES. The environment variable HC_IB_SWITCHES is set with ib_switches specified in Master Configuration file.

13. To be of more help to the users, it is suggested that you use the “recommendation” keyword in the tool configuration file as a way to indicate how to resolve problems found by the tool.
14. The “comment” keyword can be used to provide a little more information about the tool, including how you envision it being used.
15. If you are writing a new tool to be used under CHC rather than incorporating an existing tool, consider the following:
    1. Consider if it is best to have a tool that does all of the remote operations itself or if it would be better to leverage hcrun's capability to run commands remotely. For example, if hcrun did not exist, would you run the tool using xdsh with the -e option, or have the tool use xdsh to access the nodegroup?
    2. Consider how a user can leverage the various processing methods for the new tool:
        1. If you write the tool to be run on the management node and use xdsh and xcoll on its own, then you won't find as much use for xcoll and xcoll_diff, and compare. This would be similar to running the tool by itself without hcrun.
        2. If you write the tool so that it can leverage CHC's capability to do xdsh and xcoll, the user can use xcoll to view detailed results (when verbose is given)for each results group, much as they might by simply running. The user could use xcoll_diff to display only the differences between node group results. The user can use the compare method to store baselines and compare for consistency over time. These methods are best used for tools that produce consistent results across a group of nodes, where the results do not have a range of values, or otherwise match a regular expression rather than an exact string.
        3. The HC_PROCESS_METHOD environment variable can be passed to the tool to vary how the tool presents output based on the user's chosen processing method option.

4.0 Reference

1. IBM developerWorks page for IBM Cluster Health Check: https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Welcome%20to%20High%20Performance%20Computing%20%28HPC%29%20Central/page/IBM%20Cluster%20Health%20Check
2. The details about the xdsh command can be found at http://xCAT.sourceforge.net/man1/xdsh.1.html
3. The details about the xcoll command can be found at http://xCAT.sourceforge.net/man1/xcoll.1.html
4. The details about the xCAT `sinv` command can be found at http://xcat.sourceforge.net/man1/sinv.1.html

5.0 Packaging

The rpm path structure is illustrated below:

```
/etc/opt/ibmchc
/opt/ibmchc → bin → tools → ext
            → conf → groups → chc
                   → templates
            → ppc64 → bin
                    → node
            → share → man → man8
            → tools → config
                    → ext
                    → ib → ibtools → lib → perl
                    → node
                    → util
            → x86_64 → bin
                     → node
/var/opt/ibmchc → data
                → logs
```

5.1 RPM Contents

The following are contained within version 1.1.1.0 of IBM CHC:

Path	Description
/etc/opt/ibmchc/chc.conf	Overall config file for IBM CHC.
/opt/ibmchc	Main installation path
/opt/ibmchc/bin/hcrun	Overall wrapper tool
/opt/ibmchc/bin/tools/ext	Contains tools to help package and tool development
/opt/ibmchc/bin/tools/ext/build_tar.sh	Controls the tar for building a new package of just new files.
/opt/ibmchc/bin/tools/ext/findnewfiles	Looks for new files in the package paths.
/opt/ibmchc/bin/tools/ext/packupnew.sh	Packs up files find in 'findnewfiles'. This allows for patches to be easily built and distributed without having to distribute a new rpm.

Path	Description
/opt/ibmchc/conf/groups/chc	Contains configuration files for main tools groups
/opt/ibmchc/conf/groups/chc/ibswitch_check.conf	Tools group for healthcheck of InfiniBand switches.
/opt/ibmchc/conf/groups/chc/interconnect_test.conf	Tools group for running InfiniBand fabric test tools (mostly various bandwidth and connectivity tests)
/opt/ibmchc/conf/groups/chc/node_check.conf	Tools group for node healthchecks
/opt/ibmchc/conf/groups/chc/node_test.conf	Tools group for node tests (cpu and memory tests)
/opt/ibmchc/conf/groups/chc/run_ibtools.conf	Running ibtools tool sgroup
/opt/ibmchc/conf/templates	Default location where baseline files are stored. (Created the first time a baseline is taken)
/opt/ibmchc/ppc64	Contains tools specific to ppc64 architecture
/opt/ibmchc/ppc64/bin/hctoolconf	Config file for ppc64
/opt/ibmchc/ppc64/tools/node	Node tools for ppc64
/opt/ibmchc/ppc64/tools/node/cpu	CPU health check and consistency across a nodegroup
/opt/ibmchc/ppc64/tools/node/cpu.conf	Config file for CPU health check
/opt/ibmchc/ppc64/tools/node/firmware.conf	Firmware health check (simple enough that it only needs a config file)
/opt/ibmchc/ppc64/tools/node/memory	Memory health check and consistency across a nodegroup
/opt/ibmchc/ppc64/tools/node/memory.conf	Config file for memory health check
/opt/ibtmchc/share/man/man8/hcrun.8	Manpage for hcrun.
/opt/ibmchc/tools	Contains platform independent tools
/opt/ibmchc/tools/config	Tools to help with configuration checking

Path	Description
/opt/ibmchc/tools/config/compare_results	Comparing results from other tools run against a nodegroup. Called by hcrun when -m compare is requested
/opt/ibmchc/tools/config/compare_results.conf	Config file for compare_results
/opt/ibmchc/tools/config/config_check	Old config_check that compare_results replaces
/opt/ibmchc/tools/config/config_check.conf	Config file for config_check
/opt/ibmchc/tools/config/fc.py	File comparison utility used by compare_results and config_check
/opt/ibmchc/tools/config/sinv_diff	Used by config_check
/opt/ibmchc/tools/ext	Utilities that help in developing hcrun
/opt/ibmchc/tools/ext/env_echo.conf	Tool to echo the environment
/opt/ibmchc/tools/ib	InfiniBand health check tools
/opt/ibmchc/tools/ib/berlinks	Used to find links being reported above acceptable bit error rate.
/opt/ibmchc/tools/ib/berlinks.conf	Config file for berlinks
/opt/ibmchc/tools/ib/clearerrors	Used to clear errors
/opt/ibmchc/tools/ib/clearerrors.conf	Config file for clearerrors
/opt/ibmchc/tools/ib/hca_basic	Used to check minimum set of HCA configuration parameters
/opt/ibmchc/tools/ib/hca_basic.conf	Config file for hca_basic
/opt/ibmchc/tools/ib/ibqber	Used to calaculate bit error rate
/opt/ibmchc/tools/ib/ibqber.conf	Config file for ibqber
/opt/ibmchc/tools/ib/ibtools	Lower level tools for InfiniBand fabric.

Path	Description
/opt/ibmchc/tools/ib/ibtools/chkpairs_oversw	Getting pair information across a switch fabric
/opt/ibmchc/tools/ib/ibtools/clearerrors.sh	Clears errors
/opt/ibmchc/tools/ib/ibtools/collectData.BER.pl	Does bit error rate calculations
/opt/ibmchc/tools/ib/ibtools/errs_onroute.pl	Looks for errors on a route between two nodes
/opt/ibmchc/tools/ib/ibtools/ibq	Gets error information for links in the fabric
/opt/ibmchc/tools/ib/ibtools/lib/perl/ibtools-time.pm	Perl library for timestamp calculations
/opt/ibmchc/tools/ib/ibtools/lib/perl/ibtools.pm	Perl library for various functions useful for InfiniBand tools
/opt/ibmchc/tools/ib/ibtools/profile.sh	Global variables across ib tools
/opt/ibmchc/tools/ib/ibtools_install	Installs InfiniBand tools on the fabric management server
/opt/ibmchc/tools/ib/ibtools_install.conf	Config file for ibtools_install
/opt/ibmchc/tools/ib/ibtoolslog_clean	Cleans up the InfiniBand tools log
/opt/ibmchc/tools/ib/ibtoolslog_clean.conf	Config file for ibtoolslog_clean
/opt/ibmchc/tools/ib/ipoib	Tool to check IPoIB configuration
/opt/ibmchc/tools/ib/ipoib.conf	Config file for ipoib
/opt/ibmchc/tools/ib/jlinkpairs	Looks for and records pairs that go outside of expectation for the jlink bandwidth test.
/opt/ibmchc/tools/ib/jlinkpairs.conf	Config file for jlinkpairs
/opt/ibmchc/tools/ib/jlinksuspects	Looks for errors in the fabric based on jlink bandwidth test failures.
/opt/ibmchc/tools/ib/jlinksuspects.conf	Config file for jlinksuspects

Path	Description
/opt/ibmchc/tools/ib/lastclear	Determine when the last error clear was done by clearerrrors.sh
/opt/ibmchc/tools/ib/lastclear.conf	Config file for lastclear
/opt/ibmchc/tools/ib/profile.mn.sh	Management node global variables used by InfiniBand tools.
/opt/ibmchc/tools/ib/run_jlink	Run the jlink bandwidth test.
/opt/ibmchc/tools/ib/run_jlink.conf	Config file for run_jlink
/opt/ibmchc/tools/ib/run_nsdperf	Run nsdperf test
/opt/ibmchc/tools/ib/run_nsdperf.conf	Config file for run_nsdperf
/opt/ibmchc/tools/ib/run_ppping	Run ppping test
/opt/ibmchc/tools/ib/run_ppping.conf	Config file for ppping test
/opt/ibmchc/tools/ib/switch_clk	Check InfiniBand switch clocks for consistency
/opt/ibmchc/tools/ib/switch_clk.conf	Config file for switch_clk
/opt/ibmchc/tools/ib/switch_code	Check InfiniBand switch code for consistency.
/opt/ibmchc/tools/ib/switch_code.conf	Config file for switch_code
/opt/ibmchc/tools/ib/switch_health	Check results of InfiniBand switch health command
/opt/ibmchc/tools/ib/switch_health.conf	Config file for switch_health
/opt/ibmchc/tools/ib/switch_inv	Check for consistency in InfiniBand switch card and componentn population.
/opt/ibmchc/tools/ib/switch_inv.conf	Config file for switch_inv
/opt/ibmchc/tools/ib/switch_module	Check for consistency in InfiniBand switch modules.

Path	Description
/opt/ibmchc/tools/ib/switch_module.conf	Config file for switch_module
/opt/ibmchc/tools/ib/switch_ntp	Check NTP configuration of InfiniBand switches.
/opt/ibmchc/tools/ib/switch_ntp.conf	Config file for switch_ntp
/opt/ibmchc/tools/ib/swtools.chkgroup.sh	Tool used by other InfiniBand tools to make sure that a nodegroup contains valid InfiniBand switches.
/opt/ibmchc/tools/node	Node health checks
/opt/ibmchc/tools/node/clock	Check the nodes' clocks for consistency
/opt/ibmchc/tools/node/clock.conf	Config file for clock
/opt/ibmchc/tools/node/firmware.conf	Configuration file to check firmware on the nodes. This must point to the proper platform's firmware check (ppc64 or x86_64)
/opt/ibmchc/tools/node/fs_usage	Check filesystem usage.
/opt/ibmchc/tools/node/fs_usage.conf	Config file for fs_usage
/opt/ibmchc/tools/node/gpfs_state	Check GPFS state.
/opt/ibmchc/tools/node/gpfs_state.conf	Config file for gpfs_state
/opt/ibmchc/tools/node/leds	Check LEDs.
/opt/ibmchc/tools/node/leds.conf	Config file for leds
/opt/ibmchc/tools/node/nfs_mounts	Check NFS mounts.
/opt/ibmchc/tools/node/nfs_mounts.conf	Config file for nfs_mounts
/opt/ibmchc/tools/node/os	Check for consinstency in OS across a nodegroup.

Path	Description
/opt/ibmchc/tools/node/os.conf	Config file for os
/opt/ibmchc/tools/node/pe_setup_chk	Check for proper PE configuration.
/opt/ibmchc/tools/node/pe_setup_chk.conf	Configr file for pe_setup_check
/opt/ibmchc/tools/node/run_daxpy	Run the DAXPY test
/opt/ibmchc/tools/node/run_daxpy.conf	Config file for run_daxpy
/opt/ibmchc/tools/node/run_dgemm	Run the DGEMM test
/opt/ibmchc/tools/node/run_dgemm.conf	Config file for run_dgemm
/opt/ibmchc/tools/node/temp.conf	Config file to check temperature across the nodegroup. It must point to the proper platform's tool.
/opt/ibmchc/tools/util	Utilities used by tools, or handy in other regards.
/opt/ibmchc/tools/util/file_copy	Copy a file
/opt/ibmchc/tools/util/file_copy.conf	Config file for file_copy
/opt/ibmchc/tools/util/file_rm	Remove a file
/opt/ibmchc/tools/util/file_rm.conf	Config file for file_rm
/opt/ibmchc/tools/util/hostfile_copy	Copy a file from a host
/opt/ibmchc/tools/util/hostfile_copy.conf	Config file for hostfile_copy
/opt/ibmchc/tools/util/hostfile_rm	Remove a file from a host
/opt/ibmchc/tools/util/hostfile_rm.conf	Config file for hostfile_rm
/opt/ibmchc/tools/util/ibdiagcounters_clear	Clear InfiniBand diag counters on the nodes (RDMA stack counters)

Path	Description
/opt/ibmchc/tools/util/ibdiagcounters_clear.conf	Config file for ibdiagcounters_clear
/opt/ibmchc/tools/util/syslogs_clear	Clear syslogs on a nodgroup
/opt/ibmchc/tools/util/syslogs_clear.conf	Config file for syslogs_clear
/opt/ibmchc/x86_64	Specific to x86_64 platform
/opt/ibmchc/x86_64/bin/hctoolconf	Config file
/opt/ibmchc/x86_64/tools/node/bp_config	Configuration file used by bpcheck
/opt/ibmchc/x86_64/tools/node/bpcheck.pl	Best practices health check for x86_64 nodes.
/opt/ibmchc/x86_64/tools/node/bpcheck.pl.conf	Config file for bpcheck.pl
/opt/ibmchc/x86_64/tools/node/copy-bpcheck-config.conf	Copies the bp_config file to nodes
/opt/ibmchc/x86_64/tools/node/cpu	CPU health check and consistency across a health check
/opt/ibmchc/x86_64/tools/node/cpu.conf	Config file for cpu
/opt/ibmchc/x86_64/tools/node/firmware.conf	Firmare healthcheck (simple enough that it only needs a config file)
/opt/ibmchc/x86_64/tools/node/memory	Memory health check and consistency across a nodegroup.
/opt/ibmchc/x86_64/tools/node/memory.conf	Config file for memory

6.0 Use cases

This section describes various use cases

More use cases may be available on the IBM developer works page for IBM Cluster Health Check: https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Welcome%20to%20High%20Performance%20Computing%20%28HPC%29%20Central/page/IBM%20Cluster%20Health%20Check

6.1 Initializing your environment)

Edit /*etc/opt/ibmchc/chc.conf*; it is described in detail in *1.3 Configuration*.

The typical fields that are updated are:

- hc_user # This is critical to use test applications like dgemm, daxpy and jlink
- subnet_managers # comma-separated list of node names or a group
- ib_switches # comma separated list of switch names or nodegroups

6.2 Verifying consistency within a nodegroup

The current method for checking consistency of results within a nodegroup is to use the "xcoll" or the "xcoll_diff" processing method; where a nodegroup could be server nodes or switch devices.

The typical flow is:

- Prepare for verification:
  - Pick the tool or tool group that contains the checks that you wish to use to verify a nodegroup.
  - Pick a node that you decide should be the example, or seed node that will be representative of the results for the nodegroup.

Stabilize the example/seed node

- Run the tool or tool group on the node that your seed node:

```
hcrun [example node] -v -t [tool]
     OR
hcrun [example node] -v [tool group]
```

For example, to run the cpu tool against node n1 in the nodegroup 'compute', use:

```
hcrun n1 -v -t cpu
```

```
                                        # the use of -v
may vary based on the tool. The         idea is make
sure you get the detailed results and   not just
pass/fail at this point.
```

- Check the results and verify that they are as you expect. Make any updates or repairs to get to the desired results
- Verify the nodegroup
  - Depending on how you like to approach verification, you may wish to see detailed results at first and then later use the processing method to report only the differences from the example /seed node.

```
                                        # Only report
differences
                                        hcrun compute -v
-s n1 -m xcoll_diff -t cpu


                                        # Report all
results after grouping all nodes that  have the same
results
                                        hcrun compute -v
-s n1 -m xcoll -t cpu
```

- Iterate until all nodes in the nodegroup have the same results

Note: You can decide to run -m xcoll_diff immediately without choosing an example/seed node beforehand. If you don't provide a seed node, the varying results groups are compared against the results group with the largest number of nodes.

```
hcrun compute -v -m xcoll_diff -t cpu
```

6.3 Verifying consistency within a nodegroup over time

This is very similar to verifying consistency within a nodegroup, except it has the added value of comparing against saved, baseline results for that group. This will allow you to verify no unexpected changes to a configuration occurred over time

The following example uses node n1 as the seed node and uses compute as the nodegroup of interest.

- As in, *6.2 Verifying consistency within a nodegroup*, choose a seed node and verify that it is configured as desired.
- Once the seed node is as expected, save a baseline result:

```
hcrun n1 -v -m compare -s n1 -t cpu
```

- Now, you can use this baseline to check the other nodes in the nodegroup, and, in the future to check all the nodes in the nodegroup (including the seed node) against the baseline:

```
hcrun compute -v -m compare -t cpu
```

6.4 Verifying configuration after maintenance

This builds on the concept of verifying consistency within a nodegroup over time. You perform your maintenance and then run against the baseline. You verify that any changes that you see are expected. You fix the unexpected changes. Once all changes are as expected, you take a new baseline. Do this instead of simply taking a new baseline before verifying the changes are as you expected and you will have less data to verify by verifying just the changes relative to the previous configuration.

6.5 Using baseline history

Baseline history can be helpful for detailed debug. Provided that you have maintained the baselines as changes have been applied to the cluster, you can go back and look at the differences between them using the fc.py utility, which is used by the xcoll_diff processing method in hcrun.

Baselines are stored in `<templates_dir>/<tool name>[.<tools group>].<nodegroup>`

The default template_dir is /var/opt/ibmchc/conf/templates/; it can also be defined by the HC_TEMPLATES_DIR environment variable or the value of the templatesdir keyword in /etc/opt/ibmchc/chc.conf.

When a new baseline is requested the old one's filename is updated with the timestamps for when it was last modified and the current timestamp (when the new baseline is replacing it): <baseline name>.<last modified time>-<current time>, where the format of the timestamp is: YYYYmmdd_HHMMSS.

To see the difference between “baseline1_file” and “baseline2_file”, run:

```
/opt/ibmchc/tools/conf/fc.py baseline1_file
baseline2_file baseline1 baseline2

# the baseline1 and baseline2 are required as
shortnames of the baseline files. This is quirky
because fc.py is normally used to compare results
groups and those would normally be the nodegroups for
```

```
each results group file. The arguments are not
optional for fc.py.  You could use diff or sdiff,
instead, but fc.py output may be easier to understand.
```

6.6 Running Node Tests

The two node tests are DGEMM for CPU bandwidth and DAXPY for memory bandwidth.

The following examples use -v and -r to assure verbose output and to get recommendations displayed on failure.

DGEMM:

```
hcrun [nodegroup] -v -r -t run_dgemm
```

DAXPY:

```
hcrun [nodegroup] -v -r -t run_daxpy
```

6.7 Running Fabric Tests

The fabric tests are jlink for node-pair bandwidth, nsdperf for fabric to/from NSD servers, ppping for ping capability between nodes in a nodegroup.

The following examples use -v and -r to assure verbose output and to get recommendations displayed on failure.

PPPING:

```
hcrun [nodegroup] -v -r -t run_ppping
```

JLINK:

```
hcrun [nodegroup] -v -r -t run_jlink
```

NSDPERF:

```
hcrun [nodegroup] -v -r -t run_nsdperf
```