Node Health Checker Scaling Improvements and Automatic Dump and Reboot Capability

Kent Thomson
OS/IO
Cray, Inc
St Paul, USA
Email: thomson@cray.com

***Abstract*—The Node Health Checker (NHC) component runs after job failures to take compute nodes out of service that are likely to cause future jobs to fail. Before NHC can take nodes out of the availability pool, however, it must run some tests on them to assess their health. While these tests are running, the nodes being tested cannot have new jobs run on them. This period of time is known as 'Normal Mode'. By decreasing the average time of normal mode, job throughput can be increased. Performance investigation into the average run time of NHC normal mode showed that instead of scaling logarithmically with the number of nodes being tested, it instead scaled linearly, which becomes much slower at larger node counts. By localizing and fixing the bug causing the improper scaling the normal mode run time of node health was decreased by, in the best case, 100x. The analytical techniques involved in identifying scaling will be shown, including curve fitting and performance extrapolation using software tools. Additionally, the method of isolating the location of the bug by testing the different pieces of NHC separately will be discussed. Once the source of the poor scaling is revealed as calls to an external program for each node being tested, the fix of caching the required information on NHC startup in an intelligent manner is explained.**

Additionally, the new automatic dump and reboot feature of NHC is discussed. An architectural overview is given, along with common usage scenarios.

***Keywords*-NHC, scaling, dump, reboot, dumpd**

I. NHC Introduction

The Node Health Checker (NHC) is a software component on a Cray system that takes compute nodes out of the job availability pool if they fail to pass certain basic tests of their functionality. NHC is launched from a service node and connects to a list of compute nodes give to it by ALPS. A node health binary on the compute node then runs several tests designed to ensure that the node will be able to adequately run any future jobs. Examples of these tests include a test for all file systems listed in the /etc/fstab file, a test to ensure a reasonable amount of available memory on the compute node, and a test that verifies that the previous job has exited cleanly from the node.

A general knowledge of the three main pieces of NHC is integral to understanding the issues surrounding its scaling. The process is initiated by a binary that lives on the service nodes, called xtcheckhealth. xtcheckhealth oversees the testing process for a set of nodes given to it by ALPS and also manages state transitions should the nodes be unhealthy. The xtcheckhealth binary initiates the testing process by creating a binary fanout tree that connects all of the nodes in the set. The fanout tree connects to a daemon on each compute node called xtnhd. This daemon then runs the xtnhc binary that performs the actual tests. The results of the tests are sent up the fanout tree and back to the original xtcheckhealth binary. The xtcheckhealth binary then uses the results of the tests to decide which nodes should be taken out of the job availability pool. Figure 1 is a diagram of the different components of NHC.

NHC has two modes that it operates in: 'normal mode' (NM) and 'suspect mode' (SM). Normal mode runs just after an ALPS job has completed. During NM ALPS is essentially 'stuck' on the set of nodes it passed to NHC. Until NHC decides the health of each of the nodes in the job, and verifies any state changes that it performed to communicate node unhealthiness, NHC will stop ALPS from releasing the compute nodes in its list. In this mode, each test in NHC is run on the node once while the node is in the 'up' state. If a test that has an NHC action of 'admindown' in the configuration file fails on the compute node then SM is initiated on that node. In SM, the node is set to the 'suspect' state, effectively removing it from the job availability pool, and all of the NHC tests are run on the node. In SM, however, any tests that fail are restarted and rerun for the duration of SM, which is 35 minutes by default. If at any point during SM all of the tests pass the node is set back to 'up' and jobs can be run on it. If the SM timer expires and some tests are still failing, then the node is set to 'admindown' and NHC has finished with that node. At that point the admin must intervene to attempt to repair the node or wait until the next system reboot.

NHC has three 'actions' that can be specified for a test, listed here in order of increasing severity: 'log', 'admindown', and 'die'. The 'log' action is used when the failure of a test is significant, but not crucial enough to drive the node into the admindown state. In the case of a test with the 'log' action an error is printed to the console and no further action

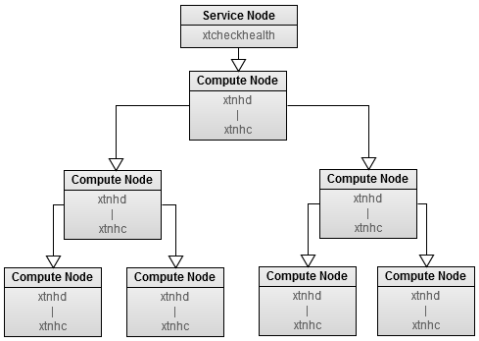

Figure 1. NHC Architectural Diagram

is taken. Tests with the action of 'log' are not run in SM. If a test with the 'admindown' or 'die' action fails the node is set to suspect state for the duration of SM, and then set to the admindown state if the test has not passed during SM. The 'die' action is even more severe than the 'admindown' action in that the compute node is shut down if the end of SM is reached without a test with the action of 'die' passing. The new automatic dump and reboot functionality adds three additional states that will be discussed later in this paper: 'dump', 'reboot', and 'dumpreboot'.

II. Node Health Scaling Performance

A. Motivation

By default, NHC only goes out to the compute nodes and runs its tests when the previous ALPS job encounters an error. ALPS jobs that run normally do not cause NHC to run its tests. A question was posed about the performance impact of having NHC run its tests after every job rather than just after failed jobs. This can have the benefit of more quickly finding nodes that may cause future jobs to fail, and it can allow administrators to run their own scripts after every job.

Please note that scaling improvements focused only on NM run time. The reasons for this are twofold. Firstly, reducing the NM run time can have an impact on total job throughput, as ALPS is unable to schedule jobs on the nodes that NHC is running NM on. This can be extremely pronounced when the NHC configuration file has been altered to cause NHC to run after every ALPS job. Secondly, the condition that causes linear scaling to occur disappears during SM. Once the isolation of the linearity is discussed this will become more clear. Regardless of SM mode considerations, the NM speed up is still significant and important, especially as system node counts increase.

No data had previously been collected on NHC runtime performance or scaling. A small investigation was done on an in house Cray system. NHC was run across node counts up to 550 in increments of 50. The investigation was set up such that no tests would fail and drive the node into suspect or admindown state. Additionally, a random set of nodes was

Table I
NHC RUN TIMES AT VARIOUS NODE COUNTS

Node Count	NHC Run Time (ms)
50	594
100	608
150	642
200	830
250	977
300	1168
350	1329
400	1538
450	1733
500	1881
550	2081

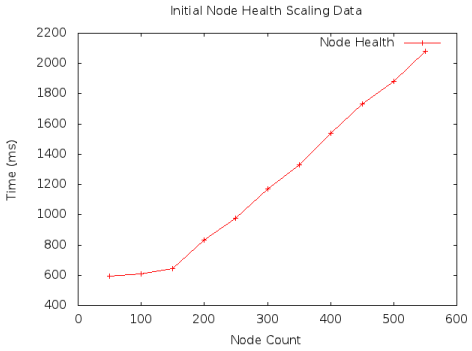

Figure 2. Initial NHC run times

chosen for each run so as to average out any jitter or noise in the system that may cause inaccurate run times. Figure 2 plots the run times of NHC over various node counts as found in table I. Please note that future data sets will not be reproduced in this paper as those sets are too large to be conveniently included.

Somewhat surprisingly, the trend line that emerges is clearly linear, or in the big O notation $O(n)$. This is surprising as NHC uses a binary fanout tree to connect to the compute nodes for the exact reason that a binary fanout tree generally provides logarithmic scaling, or $O(\log n)$. At the node counts present in Cray systems linear scaling can quickly cause undesirably long run times. It was decided that a more thorough investigation of this aberration from the expected performance was necessary.

B. Curve Fitting and Gnuplot Overview

It was decided to make use of a curve fitting tool to analyze subsequent data. Curve fitting is the act of deriving an equation that approximates experimentally measured data. This equation is necessary to verify that the desired scaling is being observed, and is extremely useful in extrapolating run times to node counts larger than what can be found in house at Cray, or indeed even in the field.

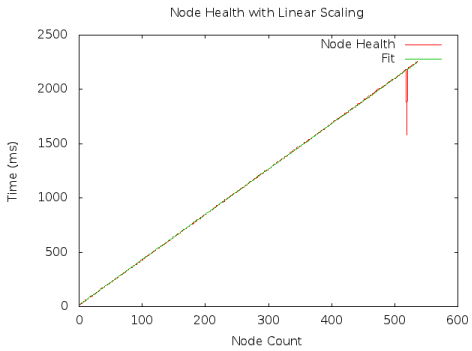

Figure 3. NHC run times with more data points

The open source tool Gnuplot (http://www.gnuplot.info/) was chosen as the curve fitting tool. It has robust and flexible curve fitting paired with graphing capabilities, and the ability to easily handle data in the format that is most convenient to produce as part of the NHC scaling investigation. All graphs and equations found in this paper were produced by Gnuplot.

C. More Thorough Investigation

A script was created to automatically gather the run times of NHC at various node counts on in house Cray systems. Multiple runs were performed for each node count. Each of the runs, even at the same node count, used a random selection of compute nodes to minimize the effect of transient disturbances in the system that might skew the data. The data points in the rest of the figures, unless explicitly stated otherwise, are averages of all of the runs taken at that node count.

Figure 3 illustrates the linearity of NHCs scaling performance. It also includes a line corresponding to the fit equation found from the data using Gnuplot.

The linearity of NHCs scaling is striking. The fit line is almost indistinguishable from the actual collected data, with the exception of a downward spike seen around the 520 node mark. Such spikes were occasionally observed and can be explained by transient conditions that cause components such as NHC to run faster or slower than normal for a brief period.

Please note that in all equations in this paper, unless explicitly stated otherwise, t is measured in milliseconds and n is measured in number of nodes. The following is the equation of the fit curve for figure 3.

$$t = 3.19n + 259 \quad (1)$$

For node counts much larger than about 800 the 259 constant factor becomes largely insignificant and the equation simplifies to:

$$t \approx 3.19n \quad (2)$$

Table II
PROJECTED NHC RUN TIMES AT LARGE NODE COUNTS

Node Count	NHC Run Time (s)
1000	3.4
10000	32.2
20000	64.1

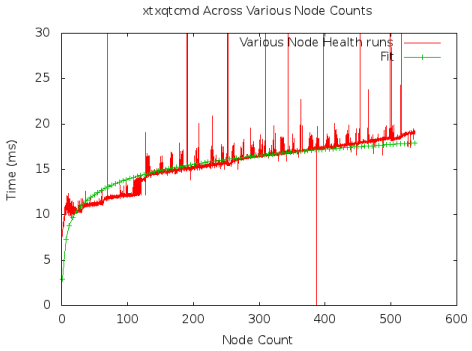

Figure 4. Run times of xtnhc when called by xtxqtcmd

Table II illustrates why such scaling becomes undesirable at large node counts. Normal mode on a node count of 20,000 takes over one minute to run.

D. Isolating the Linearity

The linearity in NHC could only be in one of the three pieces: the xtcheckhealth binary on the service nodes, the xtnhc binary on the compute nodes, or xtnhd and the fanout tree. The hypothesis was that the linearity would be found in the fanout tree piece, as small errors in a fanout tree can easily lead to suboptimal scaling.

A plan was made to eliminate both xtcheckhealth and xtnhc as the cause of the linearity, which would leave the fanout tree as the only culprit. The strategy for eliminating xtcheckhealth was to use a binary called xtxqtcmd to set up the NHC fanout tree and run xtnhc just as xtcheckhealth would, but without any additional overhead such as database connections or configuration file parsing. The xtxqtcmd binary was created to leverage the NHC fanout tree to efficiently run commands on compute nodes, and creates a fanout tree in a way identical to xtcheckhealth. The strategy for eliminating xtnhc was to replace xtnhc on the compute nodes with a binary that exits immediately with success. It was decided that the xtnhc elimination strategy would be tried second as it was extremely unlikely that the linearity would be found in xtnhc.

The run times of xtxqtcmd calling xtnhc at various node counts were gathered. Figure 4 is a graph of those runtimes.

This graph confirmed that the linearity was caused by xtcheckhealth. While the data is quite noisy, the run times are up to two orders of magnitude faster than when xtcheckhealth is used. Additionally, the data approximates a log-

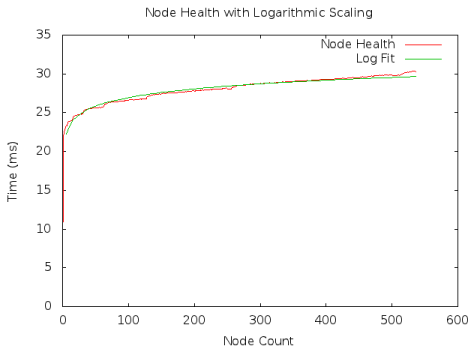

Figure 5. NHC with scaling fix up to 500 node count

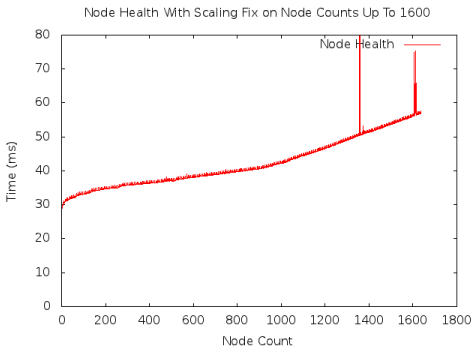

Figure 6. NHC with scaling fix up to 1600 node count

arithmic curve. A best fit logarithmic curve is included in the graph for reference, but the data is too noisy for the fit curve to be reliable for extrapolation to larger node counts. Regardless, the implication that the linearity resides in xtcheckhealth is clear.

Through some instrumentation of xtcheckhealth the cause of the linearity was ultimately found in a call to an external program. As some background, NHC is provided with the nids of the compute nodes that it is to check on. To make error messages about a nid more helpful NHC also prints out the cname that corresponds to that nid. On the service node, this conversion of nid to cname is done by calling an external program, rca-helper. This program, given a nid, returns the cname string associated with that nid. Each call to this program takes about three milliseconds, and this program was being called once for each nid in the list given to xtcheckhealth.

E. Scaling Fix

The fix to the linearity was straightforward. The rca-helper binary has an option where it will print out information that can be used to determine the cname of every nid in the system. xtcheckhealth was fixed to call rca-helper with this option at start up and cache the cname information for all of the nids in the list given to xtcheckhealth.

A version of xtcheckhealth with the scaling fix was run on node counts up to 550. Figure 5 is a graph of this data including the logarithmic fit. The logarithmic curve fits the data quite nicely.

The fit curve in Figure 5 is described by the following equation.

$$t = 1.61\ln(n) + 19.6 \tag{3}$$

Around the time that this fix was being implemented there was coincidentally a large in house system being used to identify scaling issues. A run of NHC with the scaling fix was done across node counts up to 1600. Figure 6 is a plot of that data.

In an interesting turn of events, a new linearity starts to appear in the NHC run time on node counts greater than 950.

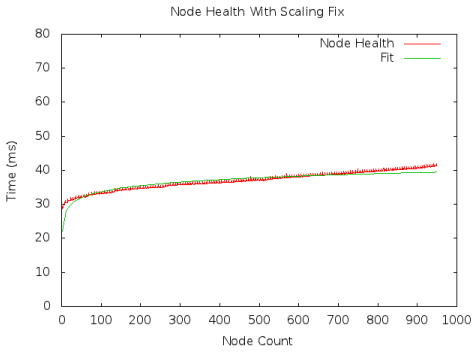

Figure 7. NHC with scaling fix up to 950 node count

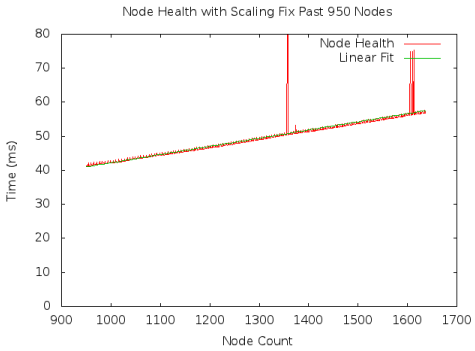

Figure 8. NHC with scaling fix on node counts from 950 to 1600

The two ranges (node counts less than 950 and node counts greater than 950) were split apart and analyzed separately. Figure 7 is a graph of times for node counts up to 950. The logarithmic fit curve is included as well.

The equation describing this logarithmic region is:

$$t = 2.54\ln(n) + 22.0 \tag{4}$$

Figure 8 is a graph of node counts greater than 950. The linear fit curve is included.

The equation describing the line is:

$$t = 0.024n + 18.6 \tag{5}$$

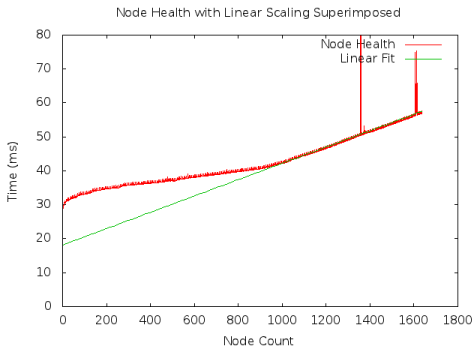

Figure 9. NHC on node counts up to 1600 with superimposed linear fit

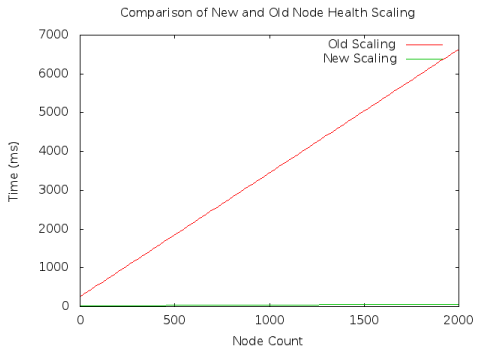

Figure 10. Comparison of old and new NHC scaling

Table III
COMPARISON OF NHC RUN TIMES WITH AND WITHOUT SCALING FIX

Node Count	Without Fix (ms)	With Fix (ms)
1000	3400	42
10000	32200	258
20000	64100	498

Figure 9 super-imposes the linear fit curve on top of the run time graph. This super-position illustrates how the logarithmic curve of the region from zero to 950 nodes hides the long-term linearity of the graph.

F. Comparison of Old and New Scaling

This new linearity is accounted for by the architecture of NHC. When xtnhc completes on each compute node a message is sent from that node up the fanout tree and ultimately to xtcheckhealth. xtcheckhealth must process the responses one at a time to determine the appropriate action for each node. This processing is a linear operation and so limits the run time to $O(n)$ at larger node counts.

Even with the new linear scaling NHC has much better performance than before the fix. Table III has projected run times of both versions of NHC across various node counts. For larger node counts the difference is on the order of a factor of 100.

Plotting the two fit equations on the same graph as in Figure 10 demonstrates the large difference in the run times. Fixing the issue that caused NHC to scale linearly with number of nodes instead of logarithmically decreased the normal mode run time significantly.

III. AUTOMATIC DUMP AND REBOOT

The latter half of this paper deals with the automatic dump and reboot feature. A design and architectural overview is presented along with several use cases for normal operation.

A. Design Goals

The main design goals for Automated Dump and Reboot (henceforth ADR) were twofold: to reduce the amount of manual dumping and rebooting that admins must perform on a system and to integrate that functionality into existing mechanisms, namely NHC. NHC without ADR does some error detection and reporting, but very little data gathering and no recovery. It writes messages to the console indicating the tests that failed and changes the state of compute nodes to take them out of the job availability pool. Any further debugging or action, however, is left up to the system admin. Oftentimes the next logical step in debugging an issue is taking a dump of an affected node. Once that has been completed a reboot of the unhealthy nodes can often resolve any issues they were having and allow future jobs to be successfully run on the nodes. ADR is an attempt to automate as much as possible in the dump and reboots steps. The hope is that admins will then be freed up to spend their time on issues that are not as clear cut as the ones encountered as part of ADR. Since NHC was already responsible for reporting problems with the health of a compute node, it made the most sense to integrate ADR into NHC.

Two additional sub-goals were also outlined for ADR: avoid detrimental actions if the admin does want to manually dump or reboot a node, and have a configuration file that is extensible enough to easily allow any command on the SMW to be run as part of the ADR cycle. There are often times when an admin does not want an automated system touching a node. As such the daemon associated with ADR watches for evidence of admin interference and cancels any actions for that node. As well, it is easy to imagine wanting to add additional steps in the dump and reboot cycle. A simple example would be sending an email alerting an admin or other entity to the existence of a new dump.

B. What is a dump?

A dump is a copy of node memory containing structures and data that can be used to reconstruct the state of the kernel at the time of the dump. It is a way to preserve the error state of a compute node, while also allowing copies of that state to be made and sent out for inspection. When read

with the appropriate program a dump can be an invaluable window into what processes are running on a node, what system calls they are performing, any files they have open, and several other pieces of information useful in debugging. Oftentimes a dump is the only way to properly debug any number of problems that may plague a node, from hung applications to memory shortages to core hangs.

Dumps are taken manually by using the ldump program on the SMW. Before taking a dump, the node is often halted by means of a non-maskable interrupt (NMI) that causes a kernel panic on the node. This effectively freezes the node in place to allow a dump to be self-consistent. Dumps that are taken while a node is still running are often partially unreliable, as node memory can be written while the dump is being performed.

C. Reboot Information

Reboots in dumpd are performed by calling the xtbootsys command with the –reboot flag. By default only one outstanding call to xtbootsys is made at a time, though up to 50 nodes (by default) can be passed to one invocation of xtbootsys. The 50 node limit was empirically determined on in-house Cray systems. Reboots of in chunks of 50 nodes or less was observed to give the best mix of a relatively large amount of simultaneous node reboots with less system-wide impact.

D. Dumpd SMW Daemon

The commands to send NMIs to nodes, take dumps, and initiate reboots are only available on the SMW. This presented a problem, as NHC lives exclusively on the service and compute nodes. To that end a new daemon, dumpd, was created on the SMW to actually perform dump and reboot actions. This new daemon is composed of four different components: the dumpd binary written in C, the python script executor, the mznhc database, and the dumpd.conf configuration file. Each of the components has a different role to play in performing dumps and reboots. They will each be explained in detail. Figure 11 shows the relationship between the different pieces of dumpd.

Before each of the pieces of dumpd are explained it is important to understand the concept of ‘actions’ as used in dumpd. Actions are comma separated lists of commands that dumpd can perform. For example, if a node is just to be rebooted the action would be “reboot”. If instead a node was to be halted, dumped, and then rebooted in order (as is often requested by NHC) the action would be “halt,dump,reboot”.

The dumpd binary has three jobs: listen for requests from NHC or other sources, create a database entry that describes the request, and start the executor script to actually perform the request. Requests are sent to dumpd in the form of HSS events. These events were chosen as the transport method for requests because of their ubiquity on a system and because of the large amount infrastructure in place to deal

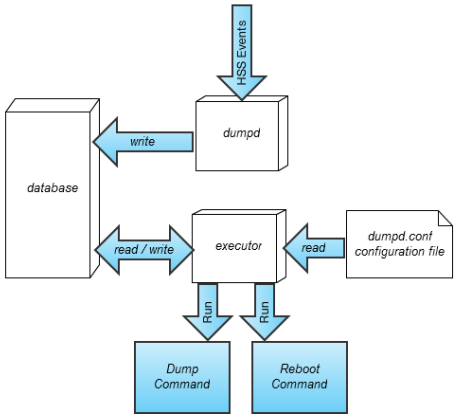

Figure 11. Dumpd Components and Interactions on the SMW

with these events. HSS events can be sent from every part of a Cray system, including the SMW, blade and cabinet controllers, and both service and compute nodes. There is also a large amount of existing code that can be used to process these events. HSS events contain space for a sender-defined payload. Dumpd uses this payload to transmit the request information. When the dumpd binary receives a request, it parses out the information from the event’s payload and places that information into a database on the SMW. The executor python script is then called.

The database is used to store information about requests for recovery and also to act as a communication path between the dumpd binary and the executor python script. There is a python script on the SMW, called dumpd-dbadmin, that allows admins a convenient interface to see any database entries currently in the database.

The dumpd configuration file is found in /etc/opt/cray-xt-dumpd/dumpd.conf on the SMW. It contains various configuration parameters and the actual command definitions of what is meant by ‘halt’, ‘dump’, and ‘reboot’. As an example, the following is the definition, in dumpd.conf, of the halt action:

```
[halt]
command: xtnmi --partition $partition $cname
max_cnames: unlimited
timeout: 60
```

The $cname variable is replaced with a comma separated list of cnames to perform the action on, and the $partition argument is replaced by the partition name that the cnames came from, for example ‘p0’ or ‘p1’. The max_cnames variable is used to determine the number of cnames in the comma separated list substituted into $cname. The timeout variable specifies, in seconds, how long the command is allowed to run for before being killed and considered to have exited in error.

The python script executor is the entity responsible for

actually performing the dumps and reboots. Upon startup by dumpd it reads dumpd.conf for the definitions of its various actions and other assorted configuration parameters. When a request contains multiple actions (for example "halt,dump,reboot") the executor performs each comma separated action in order. So in the previous example, executor would perform "halt", then "dump", then "reboot". When performing an action, the executor is smart enough to aggregate nodes with the same action together. For example, if reboot requests were sent for nodes with cnames c0-0c0s1n0 and c0-0c0s1n1 executor would call xtbootsys once, with both cnames present in the call. In that case $cname would get substituted with c0-0c0s1n0,c0-0c0s1n1. The amount of aggregation of cnames and the number of outstanding calls to a program can be controlled by the dumpd.conf configuration file. The executor runs until there are no more database entries to process. While it is running, dumpd may be continually placing new database entries in the database, and so the executor may end up processing many more requests than were present when it was started.

E. NHC Integration

Dumpd is tightly integrated with NHC. Three new NHC test actions were created for dumpd integration: 'dump', 'reboot', and 'dumpreboot'. Each of these will be discussed in detail. To use any of the three new actions the 'dumpdon' variable in the NHC config file must be set to 'on'. This variable is a quick way to disable all ADR requests from NHC. Several use cases of automatic dump and reboot can be found in the "Use Cases" subsection.

The 'reboot' action is the most straightforward, and so will be discussed first. If a test with the action of 'reboot' fails, at the end of suspect mode NHC will set the node to the 'unavail' state and send an HSS event to dumpd with an action string of 'reboot'. The 'unavail' state is used instead of the 'admindown' state for both a technical and non-technical reason. The technical reason is that nodes that are in the 'admindown' state stay in the 'admindown' state after the conclusion of a reboot. This would essentially negate the automatic aspect of dumpd, as a human would need to intervene and set the node to the 'up' state. The non-technical reason is that nodes in the queue to be rebooted are still in a state of flux, and so should not be treated as though NHC has finished with them.

The 'dump' action is slightly more complicated in its execution. At the end of SM, any nodes that have failed a test with the action 'dump' are set to the 'admindown' state. An additional parameter - maxdumps - is found in the configuration file. If a set of nodes fails a test with the 'dump' action, a dump is not requested for every single failing node. This could quickly overload the storage capabilities of the SMW while offering very little additional debugging information. Instead, only a random subset of the failing nodes have dumps performed on them. The number in this subset is determined by the maxdumps parameter. xtcheckhealth sends out HSS events for this random subset of nodes with the dumpd action string "halt,dump". The rest of the nodes are treated as though their action was 'admindown'.

The 'dumpreboot' action is, as the name suggests, a combination of the 'dump' and 'reboot' actions. Indeed, in the instance when a node has 2 failing tests, one of them with the 'dump' action and one of them with the 'reboot' action, the node is treated as though it failed with the 'dumpreboot' action. When a node fails with this action it is set to the 'unavail' state at the end of SM. The node is then added to set of nodes that are candidates to be dumped. If the node is chosen for a dump, xtcheckhealth sends an HSS event with the dumpd action string "halt,dump,reboot". If the node is not chosen for a dump, then xtcheckhealth sends an HSS event with the dumpd action string "reboot".

The three previous NHC test actions, 'log', 'admindown', and 'die', each have an associated severity. For example, if 2 tests failed on a compute node, one with the 'log' action and one with the 'admindown' action, the 'admindown' action took precedence. Similarly, the 'die' action took precedence over all other actions. The three new test actions have a similar precedence, but there are some complicating factors. The states are listed along with explanations of their precedences.

- Die - If a test with the 'die' action fails, the node shutdown is performed in all cases. This does not interact well with the 'dump' action, as a node that has been shut down has destroyed most of its error state information.
- Log - Superseded by all other actions
- Admindown - Superseded by all other other actions with the exception of Log
- Dump or Reboot individually - Supersede the 'admindown' action and the 'log' action. If one test has failed with the 'reboot' action and another has failed with the 'dump' action the node is treated as if a test with the 'dumpreboot' action had failed.
- DumpReboot - Supersedes all other actions with the exception of the 'die' action.

F. Using Dumpd Without NHC

Dumpd can also be used independently, without NHC. There is a utility found on both the service nodes and the SMW called 'dumpd-request' that can be used to request actions of dumpd. A typical usage of dumpd-request would be:

```
dumpd-request -a halt,dump,reboot -c c0-0c1s2n3
```

The -a option tells dumpd-request what series of actions to request. In the above example a halt, dump, and reboot are being requested for the cname specified with -c. The -c option will also take a comma-separated list of cnames. If

an admin has defined additional actions in the dumpd.conf configuration file then those actions can be used as well.

The following is an example of a custom action. The following lines are added to dumpd.conf:

```
[log]
command: echo $cname reboot >> /tmp/my.log
max_cnames: 1
```

If dumpd-request was then called as in the following:

```
dumpd-request -a log,reboot -c c0-0c1s2n3
```

then the command specified for 'log' in the dumpd.conf file would be performed first, followed by a reboot.

G. Use Cases

The following are some typical use cases that may be encountered.

1) Nodes failing an NHC test are dumped: This use case goes through the entire sequence of a set of nodes failing tests that all have the NHC 'dump' action. An example of a real life situation would be an application run across a set of nodes that hangs upon exit.

In this example the pertinent NHC config file options are:

```
dumpdon: on
maxdumps: 3
Application: Dump 240 300 0
```

After an ALPS job exits uncleanly, NHC is called with a nidlist consisting of all nids in the range 100-200. NHC runs NM on these nodes, and find that they all fail the application test. They are all set to 'suspect' state. NHC then runs SM on these nodes, and after 35 minutes they all continue to fail the application test. All of the nodes are then set to the 'admindown' state.

Next, xtcheckhealth must pick some nodes to be rebooted. The set of possible nodes to be dumped encompasses the entire list, every nid in the range 100-200. Since the maxdumps parameter is set to 3, xtcheckhealth picks at random nids 135, 148, and 188 to be dumped, with corresponding cnames c0-0c1s0n0, c0-0c2s0n0, and c0-0c3s0n0. xtcheckhealth then sends out dumpd requests for each of those cnames with the action string "halt,dump".

On the SMW, the dumpd daemon receives the first HSS event. It adds the information contained within that event, including the cname of the node and the action string, into the database. Dumpd then calls executor to process the request. After calling executor dumpd receives the next two events and adds them to the database.

The executor python script starts and sees three requests in the database. It takes all nodes that have the 'halt' action and performs the command defined in dumpd.conf, by default `xtnmi --partition $partition $cname`. Once that command has exited with zero, executor starts on the next action defined, 'dump'.

The dump action is defined as follows in dumpd.conf ('\' characters have been added to the end of lines that are split up due to the formatting of this paper).

```
[dump]
command: ldump -r xt-hsn@boot-$partition \
-o $dump_dir/$cname.$time $cname
max_cnames: 1
simultaneous: 1
timeout: 1200
```

In the above, $dump_dir is a directory specified earlier in dumpd.conf and $time is the time stamp when the action is performed. Since max_cnames is set to 1 and simultaneous is set to 1, this means that only one node can be dumped at a time. The executor starts on the oldest request in the database. It performs the 'dump' action for each cname in turn, and then removes the requests from the database. The executor then exits.

2) Nodes failing an NHC test are rebooted: This use case goes through the entire sequence of a set of nodes failing tests that all have the NHC 'reboot' action. An example of a real life situation would be setting the action of the memory test to 'reboot' to have nodes with low memory rebooted.

In this example the pertinent NHC config file options are:

```
dumpdon: on
maxdumps: 3
Memory: Reboot 20 30 30 1000
```

After an ALPS job exits uncleanly, NHC is called with a nidlist consisting of all nids in the range 100-199. NHC runs NM on these nodes, and find that they all fail the memory test. They are all set to 'suspect' state. NHC then runs SM on these nodes, and after 35 minutes they all continue to fail the memory test. All of the nodes are then set to the 'unavail' state, since they are all to be rebooted.

xtcheckhealth sends out dumpd requests for each of the cnames corresponding to nids 100-199 with the action string "reboot".

On the SMW, the dumpd daemon receives the first HSS event. It adds the information contained within that event, including the cname of the node and the action string, into the database. Dumpd then calls executor to process the request. After calling executor dumpd receives the next 99 events and adds them to the database.

The executor python script starts and sees some number of requests in the database, depending on how quickly dumpd can place requests in the database.

The reboot action is defined as follows in dumpd.conf ('\' characters have been added to the end of lines that are split up due to the formatting of this paper).

```
[reboot]
command: $dumpdbin/runpty xtbootsys \
--partition $partition --reboot \
--compute-loadfile CNL0 --reason \
"Automatic reboot done by dumpd at \
time $time" $cname
```

```
max_cnames: 50
simultaneous: 1
accumulation_time: 10
timeout: 1200
```

A note about the 'runpty' program above. Since xtbootsys is a TCL script, it will exit if backgrounded or called without a terminal. Since dumpd is a daemon, there is no terminal for xtbootsys to use. The runpty binary sets up a pseudo-terminal that xtbootsys then uses to run.

Since the 'reboot' action has an accumulation time of 10 seconds, executor will wait for 10 seconds after the last request with the reboot action has been received before proceeding with the reboot. This is to stop a situation where a reboot is started on 1 or 2 early requests even though a large nubmer of requests may be coming very shortly. The executor will not wait for the accumulation time if the number of requests for a reboot is the same or larger than max_cnames.

The executor starts the reboot on the first 50 nodes to be entered into the database. xtbootsys is called and successfully reboots the nodes. Once xtbootsys returns executor removes the successful requests from the database and reboots the next chunk of 50 nodes. Once they have finished the executor removes the requests from the database and exits.

3) Nodes failing an NHC test are both dumped and rebooted: This use case goes through some of the sequence of a set of nodes failing a test that has the NHC action 'dumpreboot'. An example of this might be tracking down a bug that causes apinit, the ALPS daemon on the compute node, to become unresponsive. With 'dumpreboot' a dump will be taken, but the nodes will also be returned to service.

In this example the pertinent NHC config file options are:

```
dumpdon: on
maxdumps: 1
Alps: DumpReboot 30 60 30
```

After an ALPS job exits uncleanly, NHC is called with a nidlist consisting of all nids in the range 100-199. NHC runs NM on these nodes, and find that they all fail the alps test. They are all set to 'suspect' state. NHC then runs SM on these nodes, and after 35 minutes they all continue to fail the alps test. All of the nodes are then set to the 'unavail' state, since they are all to be rebooted eventually.

xtcheckhealth randomly chooses one nid to be dumped, in accordance with the limit set in maxdumps. A dumpd request is sent for the cname associated with that nid with the action string "halt,dump,reboot". The rest of the nids have requests sent with their corresponding cnames and the action string "reboot".

The definitions of 'halt', 'dump', and 'reboot' have been reproduced in earlier use cases and so will not be reproduced here. On the SMW, dumpd gets the first request and places it in the database. Dumpd then starts the executor script and continues to place requests in the database when they are received.

The executor sees one request with the first action 'halt' (this would be the node to be dumped) and starts that command. It also sees several requests with the action 'reboot'. It waits until 50 of those have been entered into the database and starts the reboot command. Just after the reboot command starts the halt command finishes. The executor then starts on the next action for that request, which is dump. The dump command finishes 2 minutes later with the prior reboot still outstanding. After 5 more minutes the reboot finishes. The executor now sees 50 nodes queued for a reboot and starts a reboot command. When that finishes sucecssfully the requests are removed from the database.

4) Using dumpd.conf to shut down nodes before reboots: This use case involves a little more advanced use of dumpd, but can be useful in certain situations. This use case specifically deals with using the dumpd configuration file to cleanly shut down nodes prior to a reboot.

The following is added to dumpd.conf to define the new 'shutdown' action:

```
[shutdown]
command: xtcli shutdown $cname
max_cnames: 50
simultaneous: 1
accumulation_time: 1
timeout: 60
```

What is required now is a way to always have the 'shutdown' action run before the 'reboot' action. This can be accomplished by adding '`pre: shutdown`' to the end of the 'reboot' definition in dumpd.conf. The 'reboot' definition would then look like so:

```
[reboot]
command: $dumpdbin/runpty xtbootsys \
--partition $partition --reboot \
--compute-loadfile CNL0 --reason \
"Automatic reboot done by dumpd at \
time $time" $cname
max_cnames: 50
simultaneous: 1
accumulation_time: 10
timeout: 1200
pre: shutdown
```

For every new request in the database, executor will add the 'shutdown' action before the 'reboot' action. This means that for every request received after the configuration file is changed, the action string "halt,dump,reboot" will be changed to "halt,dump,shutdown,reboot" and the action string "reboot" will be changed to "shutdown,reboot".

There is an analagous 'post' option as well that runs specified actions after the defined action. Be warned, however, that dumpd enforces the rule that you cannot run anything after a reboot.

H. Admin / dumpd collisions

Occasionally, there may be times when an admin wishes to reboot a node that dumpd has in its queue or has started an action on. Dumpd makes a best effort to avoid this collision by monitoring for a specific HSS event signaling the reboot of a node. If dumpd detects one of these events it removes that node entirely from its request database. Any currently executing actions on that node will run to completion in case other nodes in that action could be successful.

Two ldumps may non-destructively dump one node simultaneously, so no effort is made to avoid dump collisions. There are some variables in dumpd.conf that can stop dumps from being taken if a specified amount of space has already been used for dumps. Please note that this is a relatively soft limit, as the size of a dump cannot be accurately known in advance. If a hard limit on the number of dumps taken is required, a separate disk partition purely for dumps may be a good alternative.

I. Practices to Avoid

It is important to remember that a reboot is not always a panacea, and can indeed be counter-productive in certain situations. A good example is setting NHC to request a reboot on the failure of the file system test. If the file system is a remote file system such as lustre or DVS, it is extremely unlikely that a reboot of a compute node will cause the file system to function normally when the compute node comes up. Most likely every time the node comes up the file system will still cause test failures, which can cause node to be rebooted multiple times in a row. This is not desirable in most situations.

Dumps can take up quite a large amount of space on an SMW, and there are pathological situations on the compute node where the kernel uses too much memory that can cause dumps even larger than normal. Because of this the max_dumps parameter in the NHC configuration file should be set quite low, perhaps in the low single digits, or even just to 1. The likelihood of additional information being present in multiple dumps is usually quite low, especially since NHC chooses nodes to dump at random.

IV. Conclusion

The isolation and fix of a scaling issue in NHC allowed the normal mode run time to be decreased by up to two orders of magnitude.

Automatic dumping and rebooting functionality present in NHC and the new SMW daemon, dumpd, help increase automation of the common admin tasks of dumping and rebooting.