Frontier node health checking and state management

Matt Ezell
High Performance Computing Scalable Systems
Oak Ridge National Laboratory
Oak Ridge, TN
ezellma@ornl.gov

***Abstract*—The HPE Cray EX235a compute blade that powers Frontier packs significant computational power in a small form factor. These complex nodes contain one CPU, 4 AMD GPUs (which present as 8 devices), 4 SlingShot NICs, and 2 NVMe devices. During the process of Frontier's bring-up, as HPE and ORNL staff observed issues on nodes they would develop a health check to automatically detect the problem. A simple bash script called checknode collected these tests into one central location ensure that each component in the node is working according to its specifications.**

ORNL developed procedures that ensure checknode is run before allowing nodes to be used by the workload manager. The full checknode script runs on boot before Slurm starts, and a reduced set of tests run during the epilog of every Slurm job. Errors detected by checknode will cause the node to be marked as "drain" in Slurm with the error message stored in the Slurm "reason" field. Upon a healthy run of checknode, it can automatically undrain/resume a node as long as the "reason" was set by checknode itself.

This paper discusses some of the checks present in checknode as well as outlines the node state management workflow.

***Index Terms*—HPC; HPE; Cray; Frontier; Node Health; Reliability**

I. Introduction

Delivering the first verified exascale computer to the world was a large challenge, further complicated by supply chain issues caused by the COVID-19 pandemic shutdowns. Building Frontier required over 60 million individual components, spread across 685 different part numbers. HPE skillfully navigated the logistical feat of acquiring all the required components to build the system, though much of the schedule contingency was consumed by component delays. Parts were arriving from suppliers to HPE the same day they were being integrated into compute blades. Frontier was delivered to Oak Ridge National Laboratory over several weeks in late 2021, and nodes were rapidly handed over to the benchmarking teams for scaling work.

Frontier's HPE Cray EX235a blades are complex, consisting of 1 AMD GPU, 4 AMD GPUs (that present as 8 separate devices), 4 HPE SlingShot 200 Gb/s NICs, and 2 NVMe devices (see Figure 1). It was imperative to ensure that nodes were completely healthy if they were included in the scheduler, to optimize the benchmarkers' time and avoid preventable failures.

HPE and ORNL staff quickly realized that a comprehensive automated health system was required to ensure quality, though sufficient tests were not present on the machine or available at delivery. ORNL evaluated several existing solutions, including the LBNL Node Health Check (NHC) and HPCM's Cluster Health Check (CHC). Eventually it was determined to port an existing in-house script called checknode to the Frontier platform.

> Mean time between failure on a system this size is hours, it's not days, so you need to make sure you understand what those failures are and that there's no pattern to those failures that you need to be concerned with.
>
> *Justin Whitt*
> *OLCF Project Director*

Both hardware and software faults can cause application failures, so it is important to check for both. Some node issues are detectable in-band, but others are only out-of-band. ORNL checknode combined with SEC monitoring provide a comprehensive capability to monitor and alert on node health. ORNL has developed various tools and procedures surrounding this process to minimize the time required to identify unhealthy nodes so they can be repaired and returned to the compute pool.

II. Existing Solutions

ORNL searched for existing software solutions for node health checking. The only viable open-source candidate found was Node Health Check from LBNL and Michael Jennings. ORNL evaluated this software and found it to be functional.

Existing computational platforms at ORNL, such as Summit, already have a custom homegrown node health script in place. Porting this to Frontier turned out to be quite straightforward. Having all the code live in a single file is advantageous so that updates can be easily pushed out to all the compute nodes.

A prototype solution was written in python to understand if a higher-level scripting solution would prove superior to simple bash. Unfortunately, no simpler interface was found that improved upon the existing bash script.

III. Checknode Function Library

The checknode program is a simple bash script. Several helper functions were developed to ease the addition of new tests to the script. A quick description of some of the functions follows:

- logstdout – log a message to standard output

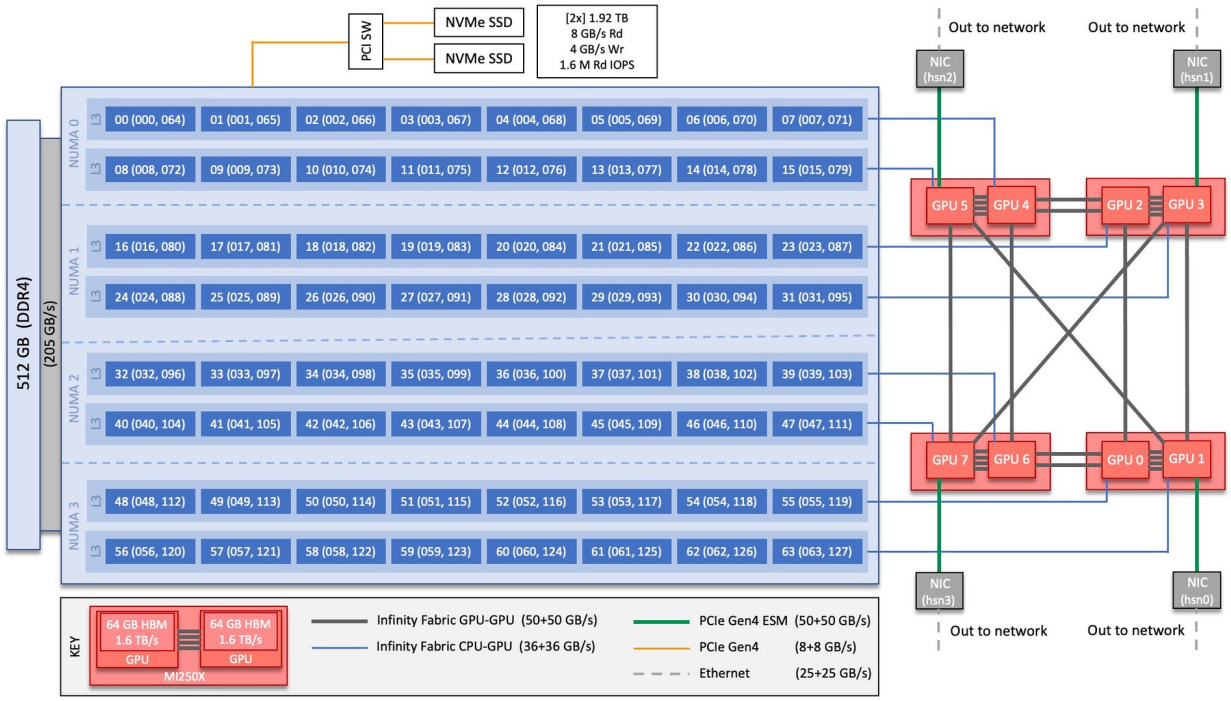

Fig. 1. Frontier Node Architecture

- logstderr – log a message to standard error
- diagerror – mark a fatal error and log to standard error
- compare – ensure that the output of a command matches a certain string
- compare2 – ensure that the output of a command matches one of two strings
- compare_ne – ensure that the output of a command does not equal a certain string
- compare_le – ensure that the output of a command is a number less than given
- compare_ge – ensure that the output of a command is greater than given
- compare_re – ensure that the output of a command matches a regular expression
- compare_nre – ensure that the output of a command does not match a regular expression
- checkproc – ensure a given process exists in the process table
- run – run a command ensure that the return code is 0
- journalgrep – check the system journal for a given match. Cache the result so subsequent checks only need to query messages since the previous check

When these functions detect a failure condition, they call *diagerror* which in turn increments a failure counter and stores the failure message to a variable.

IV. Checknode Tests

The tests present in *checknode* are added and evolve over time as new failure modes are discovered. At a high level, some of the checks include:

- Special Checks
  - Lock to make sure only 1 copy of checknode is running at a time
  - Flag file check to make sure the bootup procedure has finished before attempting any checks
  - Save "early" dmesg to avoid losing information that has rolled over
- General Checks
  - Ensure no stray processes exist
  - Node BIOS version
  - CPU core count
  - Size and availability of DRAM
  - Hugepage availability
- Hign Speed Network
  - All interfaces present
  - Firmware version
  - Link status
  - Link speed
  - MAC mode and correct AMA
  - Link flap count
  - Flow control mode
  - Link layer retry mode
  - Uncorrectable errors
  - Packet Buffer Errors
  - Credit Underflows
  - Retry Handler Running
  - Stuck cxi Services
  - ARP entry count
- GPU

  - All GPUs present
  - VBIOS version
  - RM version
  - Uncorrected HBM errors
  - DGEMM performance
  - xGMI error counts
  - Ability to read metrics
  - Number of retired pages
  - RM Firmware version
  - Queue preemption timeout
  - Total and available GPU Memory
- File systems
  - NVMe Devices Present
  - NVMe Firmware Version
  - NVMe PCIe Gen
  - NVMe PCIe Width
  - NVMe Smart Log Criticals
  - NVMe Spare Space
  - Persistent Volume Group Present
  - All DVS Mounts Present
  - Lustre Mounts Present
  - df Returns within Timeout
  - LNET NIs match AMA

See Figure 2 for example checks.

V. Case Study

To study and validate network performance, HPE would regularly run MPI tests on Frontier to ensure no degradation was present. At one point, HPE noticed that MPI all-to-all performance was underperforming compared to previous baselines.

A binary search of the nodes in the job indicated that excluding certain nodes would cause performance to return. Rebooting the problematic nodes would clear the problem, but additional nodes would get into the error state without warning. There was no obvious cause from looking at the logs.

AMD tracked the problem down to a bug in the power management firmware that prevented the CPU from going into burst mode. Without CPU burst, the all-to-all was unable to inject sufficient packets quickly enough to saturate the network. AMD developed a quick, simple test that could identify this issue. The check was added to checknode so that stuck nodes could be quickly removed from the system and prevented from causing issues with user jobs. AMD developed a firmware fix that was installed a couple weeks later.

VI. Integration with Slurm

When checknode determines that a node is unhealthy, it will drain the node with the error message as the reason. In case of multiple errors, only the first error is stored in the reason, but the count of errors is included for reference.

The checknode script integrates with Slurm to gather the current state of a node. If a node is determined to be healthy, there is a flowchart (see Figure 3) to determine if it is safe to return to the batch pool. By default, checknode will only return nodes that have a drain reason that was set by checknode itself. That way, administrator or Slurm operators can hand-set a drain reason that is not automatically cleared by checknode.

VII. Integration with SEC

ORNL has utilized the Simple Event Correlator (SEC) on all of its HPC platforms for over a decade. SEC watches the controller, console, and syslog logs for all the compute nodes, as well as the Slurm controller logs. When certain error conditions are detected, SEC will send an alert an optionally mark the node as drained. Node failures that Slurm detects cause SEC to run svtest on the relevant nodes.

VIII. Helper Script

ORNL developed a script called downnodes that parses the Slurm node information expressed as json and presents the results in an easy-to-digest format. The hardware engineers and the system engineers use this output to understand the current health of the system as well as what actions need to be taken in the short term.

The columns include the node hostname, the xname, the "age" of the message, and the reason. The command can filter for just certain reasons and also print all the matching nodes in a condensed list format. If a command is provided at the end of the command line, downnodes will use the ClusterShell library to run the command in parallel on the matching nodes. A common use case is to run checknode on nodes with certain problems (such as file system timeouts) when the root issue has resolved. In fact, the use case is so common that ORNL has integrated a cron job to automatically return nodes that were drained due to intermittent errors. See Figure 4 for example command output.

IX. Hardware Engineer Procedures

Hardware triage engineers are typically assigned to Frontier on a row-basis, with responsibility to get and maintain their row healthy. That process includes root-causing node failures, marking failed nodes and their partners drained so that they stop running jobs, and entering a hardware ticket so that a technician will take physical action on the node. After the ticket is returned, the triage engineer will run a node screen to ensure that nodes are healthy before returning them to the pool.

X. Future Directions

Tests have been added to checknode over time as problems have been discovered; new checks will inevitably need to be added as additional problems are encountered while Frontier is in production.

A future feature to add to checknode is to automatically handle draining partner nodes (the "other" node on the blade) when a hardware action is scheduled. The plan is to have the hardware engineers store the hardware action detail in the node's "extra" field.

```
checkproc munged
compare "$(/usr/sbin/dmidecode -s bios-version)" "1.6.2" "BIOS version incorrect"
compare "$(ps axo stat|grep -c D)" 0 "Processes stuck in IO Wait (D)"
compare_ge $(awk '/MemAvailable/ {print $2}' /proc/meminfo) 460000000 "Avail mem"
[ -e /home/cxi_debug/trstest.py ] && run /home/cxi_debug/trstest.py
journalgrep sq_intr 'amdgpu: sq_intr' 'GPU sq_intr - put in HBM sandbox'
for nvme in nvme0 nvme1 ; do
  [ -e /dev/${nvme}n1 ] || diagerror "NVME namespace ${nvme}n1 does not exist"
  SL=$(/usr/sbin/nvme smart-log /dev/${nvme} -o json)
  compare "$(echo $SL | jq .critical_warning)" 0 "${nvme} critical warning"
done
for gpuid in {0..7}; do
  compare_re "$(cat /sys/class/drm/${gpu}/device/current_link_speed)"
    "16(\.0)? GT/s( PCIe)?$" "GPU ${gpu} link speed"
done
```

Fig. 2. Example checknode checks

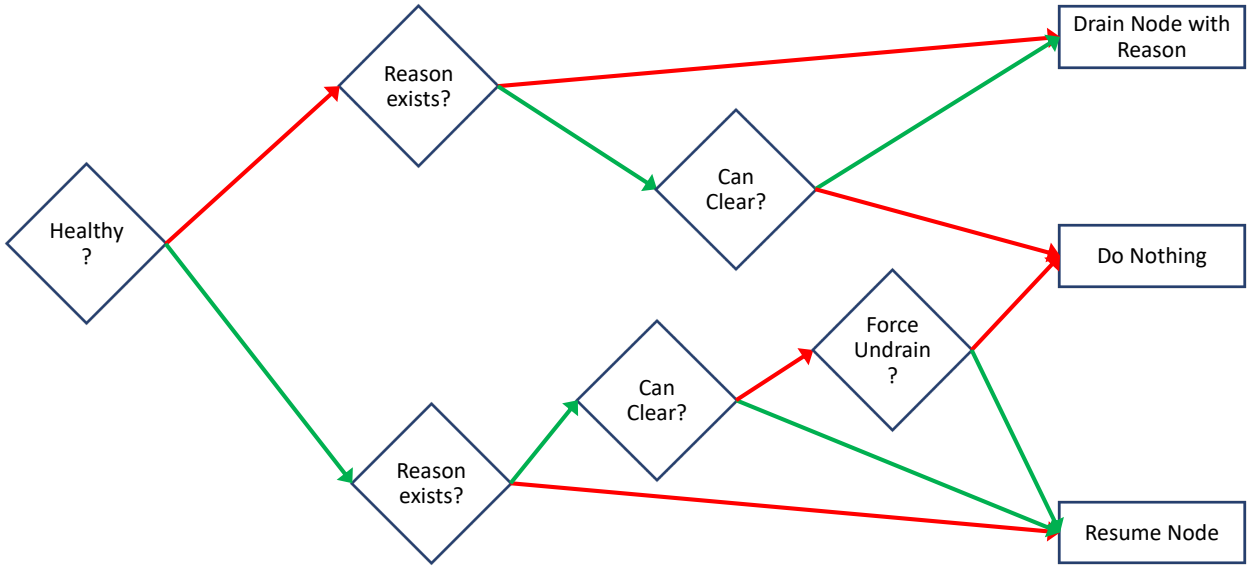

Fig. 3. Slurm Intgration

ACKNOWLEDGMENT

The author would like to thank several individuals for providing information and assistance in preparing this paper. Don Maxwell provided valuable insights and mentorship while developing this work. Jordan Webb is the primary maintainer of the active SEC rules that run on Frontier.

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

```
[root@admin1.frontier tmp]# downnodes -r rebooted
frontier02235 x2105c3s5b0n0 3d  Node unexpectedly rebooted
frontier02298 x2105c7s4b1n0 3d  Node unexpectedly rebooted
frontier02431 x2106c7s7b0n0 1d  Node unexpectedly rebooted
frontier02794 x2109c6s4b1n0 1d  Node unexpectedly rebooted
[root@admin1.frontier tmp]# downnodes -r rebooted -l
frontier[02235,02298,02431,02794]
[root@admin1.frontier tmp]# downnodes -r rebooted hostname
downnodes: frontier02235: exited with exit code 255
frontier02431: frontier02431
frontier02794: frontier02794
frontier02298: frontier02298
frontier02235: ssh: connect to host frontier02235 port 22: No route to host
```

Fig. 4. downnodes Output