From Frontier to Framework: Enhancing Hardware Triage for Exascale

Isa Muhammad Wazirzada
HPC & AI Service Delivery
Hewlett Packard Enterprise
Paradiso, Switzerland
isa.wazirzada@hpe.com

Abhishek Mehta
HPC & AI Systems Engineering
Hewlett Packard Enterprise
Seattle, U.S.A
abhishek.mehta@hpe.com

Vinanti Phadke
HPC Diagnostics
Hewlett Packard Enterprise
Bengaluru, India
vinanti.phadke@hpe.com

***Abstract*—In the world of high performance computing, the latest technological advancements alone do not necessarily lead to higher system reliability. Successfully managing compute node failures by way of accurate and timely diagnosis, articulating clear repair plans, and verifying repair success play a vital role in maintaining high levels of system availability. This paper introduces a new solution to triage compute blade failures on the Cray EX hardware platform. We start by discussing the background and motivations for such a solution before proceeding with a description of the solution itself. We also frame the solution's context in quality attributes such as diagnosability, usability, extensibility, interoperability, and observability. Last, the future roadmap of the solution is discussed.**

Keywords—Hardware Triage, Triage Automation, HPC Component Lifecycle, HPE Cray EX Hardware

I. Introduction

Supercomputers are complex systems that bring together bleeding edge technologies. Take the example of the first Exascale system, Frontier, installed at Oakridge National Laboratory. Frontier consists of more than 9400 compute nodes embedded in the HPE Cray EX4000 infrastructure. Each compute node consists of an AMD EPYC™ CPU, four MI250 GPUs interlinked by high-speed XGMI interfaces, as well as HPE Slingshot 200 Gb high-speed NICs. The system is comprised of over 150,000 node-level components interconnected in an extremely dense mechanical framework that is cooled via warm temperature liquid cooling. As impressive as all these technologies are on their own, the real value lies in bringing these technologies together to achieve sustained performance over time. With that in mind, it is critical to recognize that system quality attributes such as diagnosability and serviceability are paramount to achieving high levels of availability throughout the service life of a system.

Therefore, for HPE and our customers, providing a product-level hardware triage framework will help ameliorate the return to service time for failed components, provide a standardized approach to diagnosing hardware failures, reduce the number of no trouble found replacements, and place the experience of top subject matter experts (SME) from R&D into the hands of the teams that directly support systems in the field.

II. Solution

A. Overview

Based on our collective learnings across multiple system deployments, we architected and built a new hardware triage framework, called the Hardware Triage Tool. The framework can diagnose hardware failures, enumerate repair actions, and gather a comprehensive set of logs. Furthermore, to make the framework extensible to new hardware platforms, a YAML based Domain Specific Language (DSL) was developed. For each supported hardware platform, one will find a set of tests enumerated using the new test description language in a test workflow. The inspiration for building a test description language came from Tavern[1], a RESTful API testing tool. By developing a test description language, we were able to create discrete test workflows for multiple hardware platforms. We have aimed from the beginning to decouple the business rules for checking any given hardware platform from the core logic of the framework. This has allowed us to easily add support for new hardware platforms to the Hardware Triage Tool. The advent of this framework has shifted the conversation about how faults are diagnosed to earlier in the development cycle. Hardware and Platform engineers are now building tests during the design and early bring up phases using the standardized approach this framework provides.

B. What's in a name?

To start, it is important to understand where the Hardware Triage Tool fits in the context of the system lifecycle. The Hardware Triage Tool comes into play when a component on a compute node has failed or is suspected of failure. If a compute node has unexpectedly rebooted, powered off during a job, is suspected to be unhealthy after a job, or if a node fails to power on are all examples when the Hardware Triage Tool can be utilized. It is also important to clarify that the Hardware Triage Tool is not a substitute for the discipline of system health checks and is meant to be used once a node is suspected to have a problem. It is still essential to maintain system level health checks along with Prolog and Epilog scripts to run before and after each job to scan for pernicious node level problems. With that said, we will now examine the architectural and design choices of the Hardware Triage Tool, framed in the context of quality attributes, starting with diagnosability.

III. Quality Attributes

A. Diagnosability

The Hardware Triage Tool aims to provide high fidelity hardware failure diagnoses. The test workflows that enable diagnosis were developed with the hardware architects and developers, field, manufacturing, and firmware engineers for each supported platform. We started by asking a question: Did we have all the steps in existing documentation to provide accurate and actionable next steps for the common failures? In response to this, before any code was written, we proceeded to build an overall hardware troubleshooting workflow in the form of a decision tree diagram. The decision tree diagram starts with a key top-level decision which is whether a node is powered on or powered off. From there we split down two

main branches, the node-on branch, and the node-off branch. In Figure 1, we see a portion of the node-off branch. As we traverse this decision tree, each box represents a check that the Hardware Triage Tool must carry out. As a part of design phase for each decision box, we documented the set of condition(s) that must be met for that condition to be registered and what hardware or software actions must be performed next. Furthermore, in Figure 1, the underlined decision box titles represent links to the sections in the design document that go into the specific details on how to identify a particular failure signature and accompanying hardware actions.

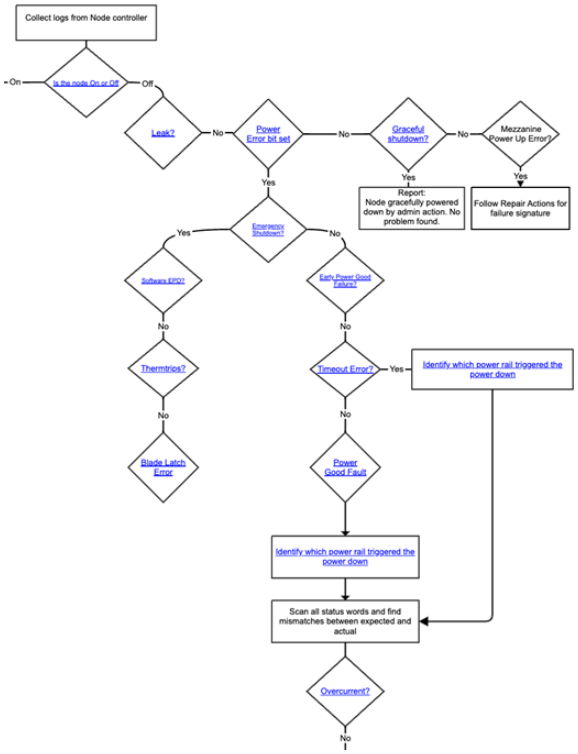

Figure 1: Highlighting a portion of the node-off workflow

Developing the workflow and supporting artifacts was a substantial effort in the development of the Hardware Triage Tool for multiple reasons. First, it encapsulated the complexity of hardware troubleshooting into an easy-to-understand diagram. Second, it provided the development team with actual instances to work with. Third, it provided the structure for us to codify the business rules and platform specific troubleshooting steps for all hardware programs. With this flow in hand and structure in place, we proceeded to architect the Hardware Triage Tool itself.

B. *Usability*

Administrators are provided an actionable insight if a failure condition is observed. The output is concise and points users to the comprehensive log bundle that was gathered. Furthermore, the framework is parameterized and allows users to easily change common runtime parameters via the command line.

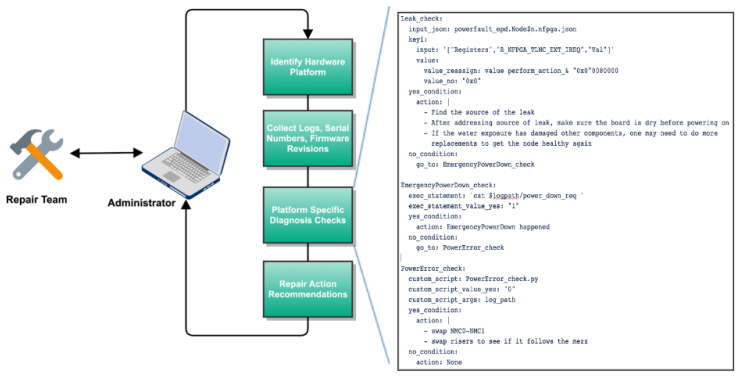

Figure 2: Hardware Triage Tool Flow

Figure 2 demonstrates how the Hardware Triage Tool traverses through a test workflow for a particular hardware platform and performs multiple checks (leak, emergency power down, power error). This example highlights three mechanisms that can be implemented for hardware fault detection:

- The Hardware Triage Tool can analyze a debug JSON file, which is normally generated by the node controller during a fault event. It makes decisions based on pre-defined criteria in the test workflow by checking the reported values. E.g., `Leak_check`.
- It can execute shell commands and provides recommendations based on the outcome. E.g., `EmergencyPowerDown_check`.
- The Hardware Triage Tool can also execute custom scripts, which can be leveraged for analyzing more complex failures. E.g., `PowerError_check`.

Figure 3 shows the Hardware Triage Tool diagnosing a hardware fault on a node in a HPE Cray EX235a blade. In this scenario, the Hardware Triage Tool identified the power rail that failed. It then proceeded to diagnose the reason for failure to be temperature related and finally suggested repair actions to recover the node.

```
Triaging :x1403c1s7b1n0

Node is in Off state
Triaging :x1403c1s7b1n0 Analysis file :  /home/    /opt/clmgr/hardware-triage-tool/logs
Stage analysis : PowerError Detected!
Triaging :x1403c1s7b1n0 Error: Temperature fault/warning fault detected for PDB SIVOC48
Repair Actions:

1. Inspect Saddle Clamp and Screw Torque

2. Inspect thermal interface material

Inspect cooling loop for signs of obstruction
Error: Input Fault or Warning fault detected for CPU0 VDD_3V3_S0
Error: Input Fault or Warning fault detected for CPU0 VDD_3V3_S5
Error: Input Fault or Warning fault detected for CPU0 DIMM VDD_VPP_ABCD_S0
Error: Input Fault or Warning fault detected for CPU0 DIMM VDD_VPP_EFGH_S0
Error: Input Fault or Warning fault detected for CPU0 VDD_1V8_S0
Triage completed!
```

Figure 3: Hardware Triage Tool Flow

C. *Extensibility*

The framework is implemented to easily add support for various hardware platforms. Currently, support exists for various AMD and Nvidia platforms in the HPE Cray EX infrastructure. Intel CPU platform support is on the roadmap. The Hardware Triage Tool was designed to follow a plugin architecture where support for additional hardware platforms

could be added atomically. To date, the Hardware Triage Tool supports the EX235a, EX255a, EX254n, EX4252, EX425, and the EX235n hardware platforms.

Extending support for new hardware programs was enabled by developing a YAML-based test description language. This has allowed the development team to quickly build tests for new hardware platforms. A test workflow can be written using the test description language and is comprised of one or more checks. Within each check one can enumerate which log file needs to be analyzed. Also, if it is machine readable, a specific key can be provided to be checked. When a test condition is met, hardware actions needed to rectify an issue can be specified. To create a workflow, one can also specify which check to move to next. In the example provided in Figure 4 we see that the next check to run is the `PowerError` check if the `leak` check conditions were not met. The test description language enabled us to create high fidelity test workflows by providing a mechanism to traverse the decision tree diagram.

```
---
leak:
  input_json: powerfault_epd.Node$n.nfpga.json
  key1:
    input: '["Registers","R_NFPGA_WNC_DVR_IREQ","Val"]'
    value:
      value_reassign: value perform_action_& "0x0"0080000
      value_no: "0x0"
  yes_condition:
    action: |
      - Find the source of the leak
      - After addressing the source of leak, make sure the board is dry before powering on
      - If the water exposure has damaged other components, one may need to do more replacements to get the node healthy again
  no_condition:
    go_to: PowerError
```

Figure 4: Test Description Language

D. *Customizability*

The framework can be customized to meet the needs for a specific hardware platform or customer environment. Moreover, if a vendor provides a diagnostic tool, it can easily be invoked by the framework. Administrators have the flexibility to create and call out their custom scripts to suit their use cases. Within the test description language there exists an interface to invoke custom scripts or tools. Currently, custom scripts can be written in Bash or Python. This interface is invoked by the addition of the `custom_script` key within a check. In Figure 5, the `custom_script` key invokes a Python script called `check_dracut_shell.py`.

The `custom_script_value_yes` key defines the return value of the script when the script is successful and the `custom_scripts_args` key provides the script a log path as a command line argument. Administrators can develop custom scripts to tackle a specific check they would like to build for their site and invoke them from within the Hardware Triage Tool.

```
Dracut_shell:
  custom_script: check_dracut_shell.py
  custom_script_value_yes: "0"
  custom_script_args: log_path
  yes_condition:
    go_to: Kernel_Panic
  no_condition:
    go_to: EFI_Shell
```

Figure 5: Invoking custom scripts within a check

All hardware supported by the Hardware Triage Tool is enumerated in a configuration file, `hardware.yml`. It contains several attributes about each supported hardware program such as expected number of NICs, correct BIOS revision, NIC firmware version, and PCIe speed. A complete example is provided in Figure 6 for the EX425 compute blade. The values for the attributes shown in Figure 6 can also be customized. For example, if a site has moved to newer NIC firmware, one can update the `hardware.yml` file with the updated version to ensure it flags any nodes that are not at the correct NIC firmware.

```
- name: "EX425"
  attributes:
    hardware:
        workflow_on: "workflows/workflow_EX425_on.yml"
        workflow_off: "workflows/workflow_EX425_off.yml"
        number_of_nics: "0"
        firmware_version: "1.5.41"
        nic_speed: "BS_200G"
        pci_speed: "16.0 GT/s PCIe"
        pci_width: "16"
        link_speed: "16.0 GT/s PCIe"
        link width: "16"
        BIOSVER: "1.6.3"
        BIOSREV: "5.14"
        CPUONLINE: "0-255"
        mem_manufacture: "1"
        DIMM_sizes: "1"
        DIMM_speed: "1"
        DIMM_count: "16"
```

Figure 6: Customizing attributes for the EX425 hardware platform

E. *Interoperability*

The implementation is system manager agnostic so that it can be run on any deployment. By design there are no dependencies on any system management specific utilities or application programming interfaces (API). This is accomplished by leveraging authenticated Redfish API calls or accessing diagnostic data directly from a targeted node controller. The Hardware Triage Tool is portable as it is written in Python and has several utility shell scripts. Being able to leverage the Hardware Triage Tool across multiple types of system managers has several key benefits. First, users of the triage tool will be utilizing a purpose-built solution to capture logs and diagnose faults. Second, it provides administrators with a familiar, supported, and documented tool that alleviates the need for unsupported scripts. Lastly, when enhancements are made to the triage tool, those enhancements will not be limited to any one system management solution.

IV. Looking Ahead

The Hardware Triage Tool is in use at numerous sites. This tool is a product level solution and will continue to incorporate support for new hardware platforms and other enhancements. An early version of the Hardware Triage Tool was released as a part of HPE Performance Cluster Manager (HPCM) 1.10 and the latest Cray System Management (CSM) release, CSM 1.5.l. Looking ahead, the roadmap of the Hardware Triage Tool includes completing the work to support all HPE Cray EX based blade types, adding coverage for newer failure scenarios, HPE Cray XD support, and parallelism so that multiple nodes can be triaged at once.

Acknowledgment

We would like to thank Felix Erales, Suresh Thapa, Mike Hicks, Jeremy Gustafson, Juha Jäykkä, Pete Guyan, Steve Martin, Brian Collum, Alan Mills, for providing platform specific guidance and helping test out the solution. We would

also like to thank Dan Cormack, Amarnath Chilumukuru, Prasanth Kurian for all the support during development of the solution. Priyanka S, Bhuvan Meda Rajesh, and Sai Anirudh Bingumalla for contributing to the development of the framework. Additionally, we would like to recognize Andy Warner and Randy Law for their holistic feedback and for championing the use of the Hardware Triage Tool. Laurence Kaplan provided fantastic feedback and guidance. We would also like to thank Tim Mossing, Claire Menut, and Kevin Henry for helping spread the reach of the Hardware Triage Tool across multiple geographies. Lastly, we are grateful for Javier Izquierdo serving as the executive sponsor and passionate advocate, which allowed the Hardware Triage Tool to come to life.

REFERENCES

[1] "Easier API testing," *Tavern*. https://taverntesting.github.io/

.