Modernizing Cray® Systems Management
Use of Redfish® APIs on Next Generation Cray Hardware

Steven J. Martin, David Rush, Kevin Hughes, Matt Kelly
Cray Inc.
{*stevem,rushd,khughes,mpkelly*}*@cray.com*

***Abstract*—This paper will give a high-level overview and a deep dive into using a Redfish® [1] based paradigm and strategy for low level hardware management on future Cray® platforms. A brief overview of the Distributed Management Task Force (DMTF) [2] Redfish systems management specification is given, along with an outline of some of our motivations for adopting an open specification for future Cray platforms. Details and examples are also provided illustrating the use of these open and accepted industry standard REST API based mechanisms and schemas. The goal of modernizing Cray hardware management, while still providing optimized capabilities in areas such as telemetry that Cray has provided in the past, is considered. This paper is targeted to system administrators, systems designers, site planners, and anyone else wishing to learn more about trends and considerations for next-generation Cray hardware management.**

***Keywords*-Redfish, DMTF, Hardware Management**

I. Introduction

Cray® XC™ systems have used a tightly integrated and proprietary systems management software stack. This proprietary systems management software stack has served Cray and Cray supercomputer customers well over the years, but it has some limitations. The Cray XC software stack is not seen as a platform that is open to customer integration and collaboration. The proprietary stack is also less adaptable to new capabilities over time.

Components of Shasta systems management ecosystem are more modular, provide open interfaces, and include enhanced security features. The Redfish based low level management stack fits well into the overall Shasta design, providing an authenticated REST [3] based API. Redfish replaces Intelligent Platform Management Interface (IPMI) on Commercial Off-The-Shelf (COTS) systems and Cray proprietary interfaces with an emerging common industry standard management approach.

This paper is organized as follows: In Section II, a more in-depth look at the motivation to change from a proprietary management stack to an open-standards based approach is given. In Section III more background is given on the DMTF (organization) and Redfish specification. Section IV gives an overview of how endpoint discovery will be implemented, when all controllers are running standards compliant firmware without hard coded assumptions on higher level software. Section V addresses examples of control actions using Redfish. Section VI covers topics of streaming telemetry and event notifications and includes some Redfish examples. In Section VII we dive into the world of standards based network boot options and how Redfish support is an improvement over IPMI. Finally in Section VIII additional information on Redfish and third party and open source tools are discussed.

This paper is targeted to system administrators, systems designers, site planners, and anyone else wishing to learn more about the next generation of Cray hardware management.

II. Motivation

As stated in Section I the current Cray XC and previous systems are managed by a proprietary and tightly coupled software stack. That software stack was first developed starting in 2002 for the "Red Storm" system at Sandia National Labs [4]–[6] that later became the Cray® XT3™. Requirements and expectations for systems management have evolved over the years since the initial development of the Cray Hardware Supervisory System (HSS) which has been the base for system management at Cray the past 15 years. In addition many of the unique features implemented using proprietary HSS code for early Cray XT3 hardware can now be implemented using non-proprietary and/or open source technology. These opportunities are enabled both by advancements in available open source software, and by the underlying embedded controller hardware which supports out-of-band (OOB) management software and firmware.

The Cray systems management group is chartered to create a common system management environment that supports all platform hardware elements used in building Cray HPC systems. Our motivations for adopting the Redfish open specification for future systems is to provide an open and industry standards-based systems management approach as a base for all hardware management going forward. The choice of Redfish allows us to manage the custom hardware we design using the same industry standard API as used with COTS hardware.

Key goals of the new systems management environment with Redfish at the hardware / firmware level include improving the following:

- Interoperability

- Security
- Resiliency
- Extensibility
- Flexibility
- Scalability
- Diagnosability
- Upgradability

Redfish provides a common and standardized way of managing Cray systems with a robust and well documented set of REST APIs. Using a common and consistent service delivery model will enable Cray to support customer-driven and product-specific integrations.

III. DMTF and Redfish

About DMTF: (From the DMTF web site) "The DMTF creates open manageability standards spanning diverse emerging and traditional IT infrastructures including cloud, virtualization, network, servers and storage. Member companies and alliance partners worldwide collaborate on standards to improve the interoperable management of information technologies. The organization is led by a diverse board of directors from Broadcom Limited; CA Technologies; Dell Inc.; Hewlett Packard Enterprise; Hitachi, Ltd.; HP Inc.; Intel Corporation; Lenovo; NetApp; Software AG; Vertiv; and VMware, Inc."

About Redfish: (From the DMTF web site) "Designed to meet the expectations of end users for simple and secure management of modern scalable platform hardware, DMTFs Redfish is an open industry standard specification and schema that specifies a RESTful interface and utilizes JSON and OData to help customers integrate solutions within their existing tool chains. An aggressive development schedule is quickly advancing Redfish toward its goal of addressing all the components in the data center with a consistent API."

The DMTF recently announced adoption of Redfish by International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) as ISO/IEC 30115:2018. [7] In the announcement the DMTF states "Redfish is a standard API designed to deliver simple and secure management for converged, hybrid IT and the Software Defined Data Center (SDDC). Both human readable and machine capable, Redfish leverages common Internet and web services standards to expose information directly to the modern tool chain."

Cray Inc. has a participation level membership in the DTMF. The DMTF has membership options for corporations, academic alliances, qualified government and end user organizations. For information on DMTF membership go to the DMTF website. https://www.dmtf.org/join/

IV. Endpoint Discovery

Cray XC Series systems utilize an IPv4 addressing scheme containing several fixed network prefixes of `10.[1-5].0.0/16`, & `10.128.0.0/14` with programmatically derived interface IDs. Management controller geolocation information is read from dedicated hardware strapping pins and encoded into its IP address. Blade & Cabinet Controller *system daemons* start on boot and initiate network connections upward to the immediate parent controller, which are well known IP addresses in the component hierarchy, until they eventually connect with the top level control system. The cabinet controller and top level control software are able to map out physical component locations of immediate subordinates by extracting location information encoded in the source address of each TCP connection. Endpoint discovery is an automatic feature of the Cray XC management stack network topology and location-aware hardware components.

In contrast to XC Series system management controllers, Redfish control endpoints have no inherent knowledge of upper level systems. Nor do they encode geolocation bits into their IP address. Network connections are typically initiated from the top down instead of bottom up. They behave much like any other network server where applicable services start on boot and accept requests forever.

Before upper level control systems can instruct Redfish endpoints to do anything useful, endpoints must first be *discovered*. The exact method in which endpoints are discovered depends on several factors. Endpoint discovery is somewhat complicated by the fact that any single method or procedure will not be suitable for all component types. For example, some components may be location-aware while others are not. Firmware components may implement varying revisions or optional features of the Redfish specification. Physical network topology and addressing schemes may factor in as well.

In the case of COTS rack mounted equipment, a predetermined network cabling scheme may be employed. If top of rack switch ports are assumed to connect to the management controller of a given rack slot, then the switch's dynamically learned MAC address table may be of use. Info collected from the MAC address table as show in 1 can be used to construct a DHCP configuration containing fixed address mappings or pre-compute IEEE 64-bit Extended Unique Identifier (EUI-64) [8] derived IPv6 addresses, when deploying a Baseboard Management Controllers (BMC) network configuration using IPv4/DHCP or IPv6/SLAAC respectively. In either case, after a table relating physical location, MAC address, and IP address has been derived, name service records may be created.

Listing 1. Ethernet Switch MAC Address Table

```
SLICE-S1#show mac-address-table dynamic

Codes: *N - VLT Peer Synced MAC
*I - Internal MAC Address used for Inter Process
     Communication
VlanId  Mac Address          Type        Interface   State
 200    a4:bf:01:28:91:4c    Dynamic     Te 1/25     Active
 200    a4:bf:01:38:ed:a1    Dynamic     Te 1/26     Active
```

```
200     a4:bf:01:2c:f8:1d   Dynamic     Te 1/27     Active
200     a4:bf:01:3e:1d:fb   Dynamic     Te 1/28     Active
200     a4:bf:01:51:25:6d   Dynamic     Te 1/29     Active
200     a4:bf:01:28:92:0f   Dynamic     Te 1/30     Active
```

The Redfish specification includes an optional feature in which BMCs include a Simple Service Discovery Protocol (SSDP) [9] responder to assist with endpoint discovery. SSDP is defined as a component of the Universal Plug & Play (UPnP) specification. It is a multicast protocol which operates over IPv4/IPv6 networks. Upper level systems initiate an endpoint discovery search by sending an appropriate payload via UDP to the SSDP multicast address. SSDP enabled endpoints receive the search query and respond with a message payload addressed to the source address of the request. The previously unknown management controller IPv4/IPv6 addresses are indicated in the source address field of the respective response payloads. Search results may include a controller hostname in SSDP result payload *LOCATION* header.

SSDP helps greatly when implemented **and** equipment is location aware. COTS equipment generally can not identify its own physical location. In those cases, SSDP only addresses the question of *what's out there*. It does not indicate where an endpoint is physically located. Cray custom hardware is different. When a Cray custom cabinet is installed, an operator enters a rack index into a front panel display. Cray designed management controllers are able to identify their physical chassis, blade location, and rack indices. Location parameters are used to generate a hostname based on a naming scheme similar to that of Cray XC embedded controllers.

Finally, endpoint discovery involves more than locating BMC IP addresses on the network and mapping them to a physical location. Redfish is implemented using HTTPS which requires an X.509 associated host server certificate and private key. A well behaved HTTPS client must perform several tests to verify the endpoint who answered is legitimate. The most basic part of server verification involves searching for the hostname given in the request URI in the *Common Name* or *Subject Alternative Name - DNS* fields present in the X.509 server certificate. A client must shutdown the connection if there is no match. Because Redfish utilizes HTTP Basic Authentication, the verification step is critically important in preventing credential disclosure to untrusted 3rd parties. Therefore, name service records (/etc/hosts or DNS) must be created as well. Although not strictly required, it is convenient to implement a naming scheme which encodes physical location into the hostname. The first BMC located in rack 7-U12 of a fictitious machine named `slice` may be given DNS A/AAAA records containing the hostname, `x7s12b0.slice.example.com`. It may be troublesome for a human to remember what roles, if any, the hostname *x7s12b0* is to fulfill. In some cases it may also be convenient to define additional aliases along with location-based hostnames.

V. Redfish Control Actions

In this section, the topic of control actions using Redfish is covered by example. First, we take a look at command and response output from two COTS servers supporting Redfish. Then, examples of chassis power control commands running on "early" Cray next generation prototype hardware are shown.

The listings shown include manual formatting and are subject to automatic line wraps to meet layout requirements. One of two command line tools are used in the examples to interact with Redfish endpoints. The first tool shown is the DMTF Redfish *redfishtool*, the second is the *curl* [10] command line tool.

In Listing 2 the output for the "COTS-A" system was generated using the DMTF Redfish *redfishtool* [11] application which lists the REST endpoints for system ID *BQWF72600788*. *redfishtool* walks the endpoint's Redfish URLs to find the requested data and ultimately outputs the data from the URL *https://COTS-A/redfish/v1/Systems/BQWF72600788*.

Listing 2. Redfishtool Systems Status from and COTS-A Node

```
sms:~user> redfishtool.py -S Always -u root -p XXXXXXXX -
    r COTS-A Systems -I BQWF72600788
{
  "Description": "Computer System",
  ...
  "Id": "BQWF72600788",
  ...
  "Actions": {
    "#ComputerSystem.Reset": {
      "ResetType@Redfish.AllowableValues": [
        "On",
        "ForceOff",
        "GracefulShutdown",
        "GracefulRestart",
        "ForceRestart",
        "Nmi",
        "ForceOn",
        "PushPowerBotton"
      ],
      "target":
        "/redfish/v1/Systems/BQWF72600788/Actions/
            ComputerSystem.Reset"
    }
  },
  ...
  "PowerState": "On"
}
```

Listing 3. Redfishtool Systems Status from and COTS-B Node

```
sms:~user> redfishtool.py -S Always -u root -p XXXXXXXX -r
    COTS-B Systems -I System.Embedded.1
{
  ...
  "PowerState": "Off",
  "Description": "Computer System which represents a
      machine (physical or virtual) and the local
      resources such as memory, cpu and other devices
      that can be accessed from that machine.",
  ...
  "Id": "System.Embedded.1",
  ...
  "Actions": {
    "#ComputerSystem.Reset": {
      "target":
```

```
        "/redfish/v1/Systems/System.Embedded.1/Actions/
            ComputerSystem.Reset",
      "ResetType@Redfish.AllowableValues": [
        "On",
        "ForceOff",
        "GracefulRestart",
        "PushPowerButton",
        "Nmi"
      ]
    }
  },
  ...
}
```

Listing 3 uses *redfishtool* to "GET" the same level of data from our second example "COTS-B" system. This listing shows the same JavaScript Object Notation (JSON) [12], [13] data elements as listed for the "COTS-A" system. Comparing the two demonstrates a few points:

- The ordering of JSON formatted data elements should not be expected to be the same between vendors.
- Not all vendors, or systems from the same vendor support all the same "ComputerSystem.Reset" actions.
- Example systems:
  - Both support:
    - *"On"*
    - *"ForceOff"*
    - *"GracefulRestart"*
    - *"Nmi"*
    - *"PushPowerButton"*
  - The "COTS-A" system adds:
    - *"GracefulShutdown"*
    - *"ForceRestart"*
    - *"ForceOn"*

The differences in reset actions for COTS-A and COTS-B illustrates the flexibility Redfish offers. Only operations that make sense are supported, yet this same flexibility causes challenges for developers of higher level tool.

In Listing 4 powering the "COTS-A" system on and off using the Redfish API is demonstrated. Note that after the "GracefulShutdown" was requested it took several calls (and some time) for the systems "PowerState" to change from "On" to "Off".

Listing 4. Using Redfish to power On and off the "COTS-A" system

```
sms:~user> curl -u root:XXXXXXXX \
        https://COTS-A/redfish/v1/Systems/BQWF72600788 \
        | python -m json.tool | grep PowerState
    "PowerState": "Off",
sms:~user>
sms:~user> curl -u root:XXXXXXXX \
        -H 'Content-Type: application/json' \
        -d '{"ResetType":"On"}' \
        https://COTS-A/redfish/v1/Systems/BQWF72600788 \
        | python -m json.tool
{
  "@odata.context": "/redfish/v1/$metadata#Message.Message
      ",
  "@odata.type": "#Message.v1_0_4.Message",
  "Message": "Successfully Completed Request",
  "MessageId": "Base.1.0.Success",
  "Resolution": "None",
  "Severity": "OK"
}
sms:~user>
sms:~user> curl -u root:XXXXXXXX \
        https://COTS-A/redfish/v1/Systems/BQWF72600788 \
        | python -m json.tool | grep PowerState
    "PowerState": "On",
sms:~user>
sms:~user> curl -u root:XXXXXXXX\
        -H 'Content-Type: application/json'\
        -d '{"ResetType":"GracefulShutdown"}'\
        https://COTS-A/redfish/v1/Systems/BQWF72600788 |
            python -m json.tool
{
    "@odata.context": "/redfish/v1/$metadata#Message.
        Message",
    "@odata.type": "#Message.v1_0_4.Message",
    "Message": "Successfully Completed Request",
    "MessageId": "Base.1.0.Success",
    "Resolution": "None",
    "Severity": "OK"
}
sms:~user>
sms:~user> curl -u root:XXXXXXXX \
        https://COTS-A/redfish/v1/Systems/BQWF72600788 \
        | python -m json.tool | grep PowerState
    "PowerState": "On",
sms:~user>
sms:~user> curl -u root:XXXXXXXX \
        https://COTS-A/redfish/v1/Systems/BQWF72600788 \
        | python -m json.tool | grep PowerState
    "PowerState": "On",
sms:~user>
sms:~user> curl -u root:XXXXXXXX \
        https://COTS-A/redfish/v1/Systems/BQWF72600788 \
        | python -m json.tool | grep PowerState
    "PowerState": "On",
sms:~user>
sms:~user> curl -u root:XXXXXXXX \
        https://COTS-A/redfish/v1/Systems/BQWF72600788 \
        | python -m json.tool | grep PowerState
    "PowerState": "Off",
sms:~user>
```

In the Redfish world much of the physical hardware is modeled as different types of chassis. In the next generation of highly optimized Cray systems each cabinet will support multiple "Enclosure" chassis that provide physical power and cooling support to front and rear module slots. Bulk power rectifiers at the enclosure chassis level are enabled via Redfish API control. When higher level software issues a request to turn on rectifiers for an enclosure via the Redfish API, platform firmware first verifies that cooling is enabled, and if not, blocks the operation until cooling is available. Assuming cooling and other environmental checks are nominal, bulk power rectifiers are enabled. In Listing 5, output of a GET operation targeting a "Chassis Enclosure" with bulk power turned off is presented. Listing 6 shows a basic *curl* command line to enable Enclosure power, and the change in output from a GET operation on a "Chassis Enclosure" afterwards.

Listing 5. Redfish GET Enclosure Status

```
sms:~user> curl -u root:XXXXXXXX \
    https://cmm7/redfish/v1/Chassis/Enclosure
{
  "@odata.context": "/redfish/v1/$metadata#Chassis.Chassis
      (Links,Status,Id,ChassisType,Manufacturer,AssetTag,
      PowerState,Name,Actions)",
  "@odata.etag": "W/\"1524072943\"",
  "@odata.id": "/redfish/v1/Chassis/Enclosure",
  "@odata.type": "#Chassis.v1_4_0.Chassis",
  "Actions": {
    "#Chassis.Reset": {
```

```
        "ResetType@Redfish.AllowableValues":
          [ "On", "Off", "ForceOff" ],
        "target": "/redfish/v1/Chassis/Enclosure/Actions/
            Chassis.Reset"
      }
  },
  "ChassisType": "Enclosure", "Id": "Enclosure",
  "Links": {
    "Contains": [
      { "@odata.id": "/redfish/v1/Chassis/B0" },
      { "@odata.id": "/redfish/v1/Chassis/M2" },
      { "@odata.id": "/redfish/v1/Chassis/B1" },
      { "@odata.id": "/redfish/v1/Chassis/M1" },
      { "@odata.id": "/redfish/v1/Chassis/M0" },
      { "@odata.id": "/redfish/v1/Chassis/B2" },
      { "@odata.id": "/redfish/v1/Chassis/M3" },
      { "@odata.id": "/redfish/v1/Chassis/B3" },
...
    ],
    "ManagedBy":
      [{ "@odata.id": "/redfish/v1/Managers/BMC" }]
  },
  "Manufacturer": "Cray Inc", "Name": "Enclosure",
  "PowerState": "Off",
  "Status": { "State": "Enabled" }
}
```

Listing 6. Curl: Enclosure Power On

```
sms:~user> curl -u root:XXXXXXXX \
     -H 'Content-Type: application/json' \
     -d '{"ResetType":"On"}' \
      https://cmm7/redfish/v1/Chassis/Enclosure/Actions/
          Chassis.Reset

sms:~user> curl -u root:XXXXXXXX \
     https://cmm7/redfish/v1/Chassis/Enclosure | grep
          PowerState
  "PowerState": "On",
```

Listing 7 shows a simplified shell script (using *curl*) that enables the ability to "Get" chassis status as well as perform the following supported "Chassis" power control related "Reset Actions":

- On
- Off
- ForceOff

By default the script targets the *Enclosure* chassis.

Listing 7. chassis-test.sh

```
#!/bin/sh
: ${ENDPOINT="https://cmm7"}
: ${ID="Enclosure"}
: ${CURLOPTS="-u root:XXXXXXXX"}
BASE="redfish/v1/Chassis"
ACTION=$1
case ${ACTION} in
  On) echo "Turning On"
    curl ${CURLOPTS} \
      -H 'Content-Type: application/json' \
      -d '{"ResetType":"On"}' \
      ${ENDPOINT}/${BASE}/${ID}/Actions/Chassis.Reset;;
  Off) echo "Turning Off"
    curl ${CURLOPTS} \
      -H 'Content-Type: application/json' \
      -d '{"ResetType":"Off"}' \
      ${ENDPOINT}/${BASE}/${ID}/Actions/Chassis.Reset;;
  ForceOff) echo "Forcing Off"
    curl ${CURLOPTS} \
      -H 'Content-Type: application/json' \
      -d '{"ResetType":"ForceOff"}' \
      ${ENDPOINT}/${BASE}/${ID}/Actions/Chassis.Reset;;
  Status) echo "Checking status"
    curl ${CURLOPTS} \
      ${ENDPOINT}/${BASE}/${ID} ;;
  *) echo "Unknown action: ${ACTION}";;
esac
```

In listing 8, *redfishtool* lists the "Chassis Collection" of the targeted endpoint.

Listing 8. Chassis list

```
sms:~user> redfishtool.py ${RFTOPTS} -r cmm7 Chassis list
{
  "_Path": "/redfish/v1/Chassis",
  "Members@odata.count": XX,
  "Name": "Chassis Collection",
  "Members": [
    {"@odata.id": "/redfish/v1/Chassis/Enclosure", "Id": "
        Enclosure" },
    {"@odata.id": "/redfish/v1/Chassis/M1", "Id": "M1"},
    {"@odata.id": "/redfish/v1/Chassis/M2", "Id": "M2"},
    {"@odata.id": "/redfish/v1/Chassis/B2", "Id": "B2"},
    {"@odata.id": "/redfish/v1/Chassis/B0", "Id": "B0"},
    {"@odata.id": "/redfish/v1/Chassis/B1", "Id": "B1"},
    {"@odata.id": "/redfish/v1/Chassis/M0", "Id": "M0"},
    {"@odata.id": "/redfish/v1/Chassis/B3", "Id": "B3"},
    {"@odata.id": "/redfish/v1/Chassis/M3", "Id": "M3"},
    ...
  ]
}
```

Treatment of Chassis in the Redfish specification enables the script *chassis-test.sh* (shown above in Listing 7) to target any of the chassis in the "Chassis Collection" by simply setting (or overriding) the environment variable "ID". This should help illustrate how the regularity and consistency of the Redfish specification can lead to simplification and code reuse in debug tools and higher level systems management software.

VI. Telemetry

Along with changes to use Redfish as the low level hardware control interface on Shasta, there are corresponding changes with resect to telemetry data collection and publishing.

For data collection, just like on the XC, there are various types of telemetry data available in Shasta. Note that the same data that is accessible today on the XC will also be provided with Shasta. An example of this telemetry data includes:

- Power/energy data
- Thermal data
- Alerts and events

The current DMTF Redfish specification provides two methods of telemetry data collection, synchronous and asynchronous, depending on the APIs supported by the Redfish endpoint.

A. Asynchronous Collection (event subscription based)

For Cray capability class machines, where Cray controls the Redfish software stack within the embedded controllers, all telemetry data can be accessed via either method. To provide a scalable mechanism for streaming telemetry, Cray will provide all data as an asynchronous update available via the Redfish EventService API.

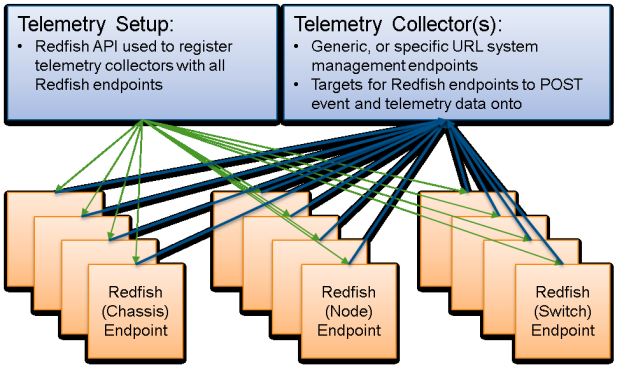

Figure 1: Redfish telemetry setup

The event service is an alert mechanism used in Redfish allowing consumers to subscribe to updates from the endpoint. This event data will be sent out through HTTPS to a user supplied target (Cray-provided for capability class systems). This allows telemetry data to be pointed dynamically to the desired destination.

Figure 1 illustrates a telemetry setup and delivery pattern using the Redfish EventService subscription method. The "Telemetry Setup" and "Telemetry Collector(s)" represent higher level management software. The setup is persistent and will generate a very low volume control API traffic pattern. Collectors for the streaming telemetry case see steady, continuous, and relatively constant load. System size and telemetry collection rates will be used to scale-out the number of collectors needed on large systems. Provisioning of collectors dedicated for asynchronous (less predictable) events will be possible to help insure that error events get adequate priority. In addition to possible scaling benefits, specialization of collectors by the type of telemetry they handle may lead to better efficiency.

The Redfish EventService is clearly documented [14].

The EventService examples in this section all target the system "COTS-B", because unlike the examples in Section V the APIs used in this Section are not provided by the Redfish implementation on the "COTS-A" system.

The code in Listing 9 is using the *redfishtool* to look at the "EventService" base REST URL for the "COTS-B" system.

Listing 9. redfishtool: EventService GET

```
sms:~user> redfishtool.py -S Always -u root -p XXXXXXXX \
        -r COTS-B raw GET /redfish/v1/EventService
{
  "DeliveryRetryAttempts": 3,
  "Subscriptions": {
    "@odata.id": "/redfish/v1/EventService/Subscriptions"
  },
  "EventTypesForSubscription": [
    "StatusChange",
    "ResourceUpdated",
    "ResourceAdded",
    "ResourceRemoved",
    "Alert"
  ],
  "Actions": {
    "#EventService.SubmitTestEvent": {
      "EventType@Redfish.AllowableValues": [
        "StatusChange",
        "ResourceUpdated",
        "ResourceAdded",
        "ResourceRemoved",
        "Alert"
      ],
      "target": "/redfish/v1/EventService/Actions/
          EventService.SubmitTestEvent"
    }
  },
  "@odata.type": "#EventService.v1_0_2.EventService",
  "Id": "EventService",
  "@odata.id": "/redfish/v1/EventService",
  "DeliveryRetryIntervalInSeconds": 30,
  "Description": "Event Service represents the properties
       for the service",
  "Status": {
    "State": "Disabled",
    "Health": "Ok",
    "HealthRollUp": "Ok"
  },
  "Name": "Event Service",
  "@odata.context": "/redfish/v1/$metadata#EventService.
       EventService",
  "EventTypesForSubscription@odata.count": 5,
  "ServiceEnabled": false,
  "IgnoreCertificateErrors": "Yes"
}
```

To add a subscription, send an HTTPS POST to the endpoint with a correctly formatted a JSON payload as in the example provided in Listing 10.

Listing 10. Curl: EventService Subscription

```
sms~user> curl -u root:XXXXXXXX -X POST \
     https://COTS-B/redfish/v1/EventService/Subscriptions/
{
  "Destination": "http://SMS-Telemetry/TelemetryEndpoint",
  "Context": "power",
  "EventTypes": [
    "Alert",
    "Status Change",
      Value
  ],
  "Protocol": "Redfish"
}
```

A subscription ID (GUID) for this event subscription is returned upon successfully subscribing to the event, of the form: 74c8eca0-28b9-11e8-a4c7-74e6e2fb6114.

Once the subscription has been set up with a target endpoint, any time there is a change of the type specified in the JSON payload, a Redfish event will be sent via an HTTPS POST command to the target endpoint specified in the subscription.

To see a list of all subscriptions, send an HTTPS GET command to this same endpoint with no JSON payload, as in the following example:

Listing 11. Curl: EventService Subscription GET

```
sms:~user> curl -u root:XXXXXXXX \
    https://COTS-B/redfish/v1/EventService/Subscriptions
{
  "@odata.context": "/redfish/v1/$metadata#
       EventDestinationCollection.
       EventDestinationCollection",
  "@odata.id": "/redfish/v1/EventService/Subscriptions",
  "@odata.type": "#EventDestinationCollection.
       EventDestinationCollection",
  "Description": "List of Event subscriptions",
  "Members": [
    {
```

```
      "@odata.id": "/redfish/v1/EventService/Subscriptions
           /74c8eca0-28b9-11e8-a4c7-74e6e2fb6114"
    }
  ],
  "Members@odata.count": 1,
  "Name": "Event Subscriptions Collection"
}
```

To remove a subscription and stop receiving telemetry data, send an HTTPS DELETE command to the endpoint specifying the subscription ID, as in the following example:

Listing 12. Curl: EventService Subscription Remove

```
curl -u root:XXXXXXXX \
     -X DELETE https://COTS-B/redfish/v1/EventService/
          Subscriptions/dad9b3fe-174c-11e8-a98f-64006
          ac4a60a
```

B. Synchronous Collection (polling based)

For COTS white box servers in Cray cluster solutions, the implementation is constrained to use whatever the vendor provides in their Redfish stack. Where asynchronous telemetry collection is supported, it will function in the DMTF specified manner as shown. However in cases with COTS servers where asynchronous telemetry collection is not supported, the various pieces of telemetry data can be collected synchronously by polling the Redfish endpoint, as shown below.

Listing 13. Curl: Poll (GET) COTS-A Server Power

```
sms:~user> curl -u root:XXXXXXXX \
     https://COTS-A/redfish/v1/Chassis/Baseboard/Power#/
          PowerControl/0
{
  "@odata.context": "/redfish/v1/$metadata#Power.Power",
  "@odata.id": "/redfish/v1/Chassis/Baseboard/Power",
  "@odata.type": "#Power.v1_2_1.Power",
  "Id": "Power",
  "Name": "Power",
  "PowerControl": [
    {
      "@odata.id": "/redfish/v1/Chassis/Baseboard/Power#/
           PowerControl/0",
      "MemberId": "0",
      "Name": "Server Power Control",
      "PowerConsumedWatts": 19,
      "PowerMetrics": {
        "AverageConsumedWatts": 149,
        "IntervalInMin": 18250,
        "MaxConsumedWatts": 341,
        "MinConsumedWatts": 2
      },
      "RelatedItem": [
        {
          "@odata.id": "/redfish/v1/Systems/BQWF72600788"
        },
        {
          "@odata.id": "/redfish/v1/Chassis/Baseboard"
        }
      ]
    }
  ],
  ...
}
```

Listing 14. Curl: Poll (GET) COTS-B Server Power

```
curl -u root:XXXXXXXX \
     https://COTS-B/redfish/v1/Chassis/System.Embedded.1/
          Power
{
  "@odata.context": "/redfish/v1/$metadata#Power.Power",
  "@odata.id": "/redfish/v1/Chassis/System.Embedded.1/
       Power",
  "@odata.type": "#Power.v1_0_2.Power",
  "Description": "Power",
  "Id": "Power",
  "Name": "Power",
  "PowerControl": [
    {
      "@odata.id": "/redfish/v1/Chassis/System.Embedded.1/
           Power/PowerControl",
      "MemberID": "PowerControl",
      "Name": "System Power Control",
      "PowerAllocatedWatts": 750,
      "PowerAvailableWatts": 0,
      "PowerCapacityWatts": 980,
      "PowerConsumedWatts": 11,
      "PowerLimit": {
        "CorrectionInMs": 0,
        "LimitException": "HardPowerOff",
        "LimitInWatts": 180
      },
      "PowerMetrics": {
        "AverageConsumedWatts": 26,
        "IntervalInMin": 60,
        "MaxConsumedWatts": 26,
        "MinConsumedWatts": 26
      },
      "PowerRequestedWatts": 510,
      "RelatedItem": [
        {
          "@odata.id": "/redfish/v1/Chassis/System.
               Embedded.1"
        },
        {
          "@odata.id": "/redfish/v1/Systems/System.
               Embedded.1"
        }
      ],
      "RelatedItem@odata.count": 2
    }
  ],
  ...
}
```

C. Telemetry Publication

Along with the changes in how telemetry data is collected using Redfish APIs and JSON data formats, there are also changes to the way it is published in Cray platform software. The telemetry collector pulls the data from the hardware via communications with the Redfish endpoint, whether via synchronous or asynchronous method, and then this data is published onto a streaming telemetry bus. This telemetry bus is used by the Cray capability class systems management to make this data available internally, as well as to customer provided systems. Figure 2 shows a high level view of telemetry collection from Redfish onto a telemetry bus.

D. Upcoming DMTF Telemetry Model Specification

To expand the capability of the Redfish telemetry specification, the DMTF has a work-in-progress project that is nearing approval for a new telemetry model [15].

This specification proposes a general model for describing metrics characteristics, metrics reports, and triggers that will work with existing metrics properties.

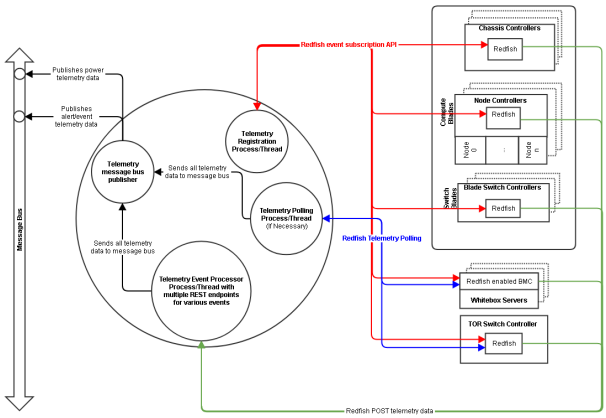

Figure 2: Redfish telemetry onto a telemetry bus

The proposal provides for a new Redfish root level service called the TelemetryService in the Redfish specification, as outlined below:

Listing 15. Proposed TelemetryService

```
{
  "@Redfish.Copyright": "Copyright 2014-2016 Distributed
      Management Task Force, Inc. (DMTF). All rights
      reserved.",
  "@odata.context": "/redfish/v1/$metadata#
      TelemetryService.TelemetryService",
  "@odata.type": "#TelemetryService.1.0.0.TelemetryService
      ",
  "@odata.id": "/redfish/v1/TelemetryService",
  "Id": "TelemetryService",
  "Name": "Telemetry Service",
  "Status": {
    "State": "Enabled",
    "Health": "OK"
  },
  "MetricDefinitions": {
    "@odata.id": "/redfish/v1/TelemetryService/
        MetricDefinitions"
  },
  "MetricReportDefinitions": {
    "@odata.id": "/redfish/v1/TelemetryService/
        MetricReportDefinitions"
  },
  "MetricReports": {
    "@odata.id": "/redfish/v1/TelemetryService/
        MetricReports"
  },
  "Triggers": {
    "@odata.id": "/redfish/v1/TelemetryService/Triggers"
  }
}
```

This proposal also provides for common metric definitions and statistics, metric report definitions, metric reports, and triggers.

VII. Redfish & Network Boot

Redfish does not define a boot protocol. It does provide a mechanism to select a boot source and then trigger a (re)boot. Capabilities are similar to those provided by IPMI, only more flexible. Those familiar with IPMI will find the standard *PXE*, *HDD* boot options work as expected. The boot source can be overridden once or permanently. A major enhancement over IPMI which Redfish provides is the ability to specify a one time boot option by Unified Extensible Firmware Interface (UEFI) [16], [17] device path. The device path can identify any bootable component supported by firmware. If a system has multiple network interfaces, the desired interface can be selected directly via UEFI device path and reset. This is an improvement over the *PXE* option, which doesn't allow specifying specifically which network device to use. Selecting a specific network boot interface is a requirement when a computer system is connected to more than one PXE capable network.

Not all Redfish enabled BMCs support an identical list of boot options. As can be seen in Listing 16, supported boot source options must be discovered.

Listing 16. Boot Source Selection

```
...
    "Boot": {
        "BootSourceOverrideEnabled": "Once",
        "BootSourceOverrideTarget": "None",
        "BootSourceOverrideTarget@Redfish.AllowableValues"
            : [
            "None",
            "Pxe",
            "Cd",
            "Floppy",
            "Hdd",
            "BiosSetup",
            "Utilities",
            "UefiTarget",
            "SDCard"
        ],
        "UefiTargetBootSourceOverride": ""
    },
...
```

In contrast to the Cray XC Series, next generation compute nodes include a node-visible management Ethernet interface. This additional standard Ethernet interface, combined with Redfish based node management, and support for standardized UEFI Network Bootstrap Programs (NBP) enable a standardized network boot paradigm on Cray's custom, density optimized, hardware platform.

A popular NBP, iPXE [18], allows downloading boot image content using multiple transfer protocols including HTTP. Additionally, iPXE may be scripted. Those two properties make implementing a dynamic boot server straight forward. One such possibility involves scripting iPXE such that it submits, via HTTP, the boot device hardware MAC address as part of a chain loading procedure. A boot script server can use the submitted MAC address as a database key, look up presently assigned boot content, and respond with the final boot script.

Listing 17. iPXE Stage 1 Boot Script

```
#!ipxe
chain http://192.168.10.10:8000/chainload?mac=${net0/mac}
```

The initial chain loading script (see Listing 17) is a static two line file that can execute on all nodes. iPXE will generally download and execute it via TFTP. When using DHCPv4, this script is listed in the "filename" parameter. In this example iPXE is instructed to chainload another script

from an HTTP server located at IP address 192.168.10.10. The MAC address of network device 0 is sent along as a URL-encoded form.

Upon receiving the form submission, a boot script server may dynamically generate relevant parameters and construct a second stage boot script such as the one show in Listing 18. Lines beginning with *kernel* and *initrd* inform iPXE where kernel & initrd content should be downloaded along with any applicable kernel command line arguments. File transfer protocol, host, and port number are assumed by iPXE to be relative to those specified in the stage 1 boot script if omitted. The additional parameter *bootmac*, while not required, is a convenient hint informing the booted operating system which network device it booted from.

Listing 18. iPXE Stage 2 Boot Script

```
#!ipxe
kernel /image/default/vmlinuz initrd=initrd.gz bootmac=02:
    E0:1D:96:4C:00
initrd /image/default/initrd.gz
boot
```

The use of iPXE is only one of several options available for booting Cray next generation compute platforms, the intent with Shasta is to allow for supporting a wide range of industry standard boot mechanisms.

VIII. Cray And 3rd Party Tool Integration

While Redfish offers a rich set of control and monitoring functionality, it does so at a fairly low level. All Redfish control and query operations can be done using the *curl* command, for example, from the command line. While useful, this usage model is less than ideal for administering a large system.

The Redfish industry-standard REST interface provides excellent opportunities for integration with tools which can take care of the underlying operations, hiding details from the administrator, as well as allowing for scaling in larger systems.

Cray will implement tools to simplify administration tasks. These tools will provide for various operations such as powering nodes on and off, checking states of various hardware components, setting up monitoring and telemetry, etc.

Cray Redfish based administration tools can be used to do the following:

- Discover Redfish endpoints/URLs
- Hide the details of the underlying Redfish operations
- Fanning out and parallelizing multiple Redfish operations for scaling purposes
- Setting up asynchronous eventing and streaming telemetry collection parameters

Endpoint discovery was covered in detail in Section IV of this paper. Data from the endpoint discovery process will be stored in a Cray management database table for use by higher level systems management software.

Generally a control operation performed by a system administrator, such as turning nodes on, will target multiple devices. One example of this is turning all nodes of the system on for the first time after a site wide power outage, and booting the system. The Cray hardware management system will parallelize these operations rather than doing them serially. This allows for scaling to very large systems. Cray software will also parallelize operations such that site level constraints for power and cooling are not violated when managing systems with ramp rate constraints.

In addition to Cray-supplied tools, sites can implement their own tools and utilities coded to the well documented Redfish APIs. Redfish was design to be "DevOps" friendly with a wide range of options for new code development using modern programming languages.

In addition, there are 3rd party/open source tools available which work with Redfish. For example, DMTF [19] has a selection of tools which utilize Redfish to control hardware, check for conformity to the Redfish standards, monitor Redfish message streams, etc.

This flexibility allows for an environment in which Cray customers can use Cray tools, 3rd party tools, open source tools, in-house developed tools, or any combination thereof to administer and monitor a Cray system. If a site needs a very specific tool, there is no need to ask Cray engineering to implement said tool; the site can potentially use an existing non-Cray tool, or develop their own.

IX. Conclusion

This paper has described the motivation, strategy, and basic concepts of using the DMTF's Redfish standard APIs to deliver an open standard and secure low level system management infrastructure. The Redfish framework Cray is utilizing allows for a very powerful, flexible, and open system for controlling and monitoring the next generation of Cray systems. This paper is intended to show that we are not only thinking about the use of Redfish, and open standards but that we are in fact doing so. We hope that this paper is informative, and that it will encourage future dialog with members of the Cray User Group and the wider HPC community.

REFERENCES

[1] "The DMTF Redfish Developer Hub," (Accessed 3.Apr.18). [Online]. Available: https://redfish.dmtf.org/

[2] "DMTF: The Distributed Management Task Force," (Accessed 3.Apr.18). [Online]. Available: https://www.dmtf.org/

[3] "REST: REpresentational State Transfer," (Accessed 3.Apr.18). [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer

[4] "Red Storm (computing)," (Accessed 8.Apr.18). [Online]. Available: https://en.wikipedia.org/wiki/Red_Storm_%28computing%29

[5] "Sandia National Laboratories and Cray Inc. finalize contract for new supercomputer," (Accessed 8.Apr.18). [Online]. Available: https://share-ng.sandia.gov/news/resources/releases/2002/comp-soft-math/redstorm.html

[6] "Sandia Red Storm supercomputer exits world stage," (Accessed 8.Apr.18). [Online]. Available: https://share-ng.sandia.gov/news/resources/news_releases/red-storm-exits/

[7] "ISO/IEC 30115:2018," (Accessed 26.Apr.18). [Online]. Available: https://www.iso.org/standard/53235.html

[8] "Rfc 2373: Ip version 6 addressing architecture," (Accessed 7.May.18). [Online]. Available: https://www.ietf.org/rfc/rfc2373.txt

[9] "SSDP: Simple Service Discovery Protocol," (Accessed 18.Apr.18). [Online]. Available: https://en.wikipedia.org/wiki/Simple_Service_Discovery_Protocol

[10] "curl," (Accessed 23.Apr.18). [Online]. Available: https://en.wikipedia.org/wiki/CURL

[11] "DMTF Redfishtool: GitHub," (Accessed 3.Apr.18). [Online]. Available: https://github.com/DMTF/Redfishtool/blob/master/README.md

[12] "Rfc 8259: The javascript object notation (json) data interchange format)," (Accessed 3.Apr.18). [Online]. Available: https://tools.ietf.org/html/rfc8259

[13] "Ecma standard: Ecma - 404," (Accessed 18.Apr.18). [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[14] "DMTF Redfish: DSP0266," (Accessed 19.Apr.18). [Online]. Available: http://redfish.dmtf.org/schemas/DSP0266_1.1.html

[15] "DMTF Redfish: DSP-IS0002," (Accessed 19.Apr.18). [Online]. Available: https://www.dmtf.org/documents/redfish/redfish-telemetry-proposal-09a

[16] "Unified Extensible Firmware Interface Forum," (Accessed 7.May.18). [Online]. Available: http://www.uefi.org/

[17] "Redfish Configuration of UEFI HII Settings," (Accessed 7.May.18). [Online]. Available: http://www.uefi.org/sites/default/files/resources/UEFI_Plugfest_Redfish_Fall_2016.pdf

[18] "iPXE," (Accessed 9.May.18). [Online]. Available: https://ipxe.org/

[19] "DMTF: GitHub," (Accessed 3.Apr.18). [Online]. Available: