Cray Advanced Platform Monitoring and Control (CAPMC)

Steven J. Martin
Cray Inc.
Chippewa Falls, WI USA
stevem@cray.com

David Rush
Cray Inc.
Chippewa Falls, WI USA
rushd@cray.com

Matthew Kappel
Cray Inc.
St. Paul, MN USA
mkappel@cray.com

***Abstract*—With SMW 7.2.UP02 and CLE 5.2.UP02, Cray released its platform monitoring and management API called CAPMC (Cray Advanced Platform Monitoring and Control). This API is initially directed toward workload manager vendors to enable power-aware scheduling and resource management on Cray XC-series systems and beyond. In this paper, we give an overview of CAPMC features, applets, and their driving use cases. We further describe the RESTful architecture of CAPMC and its security model. Finally, we preview future enhancements to CAPMC in support of in-band control and additional use cases.**

***Keywords*-Power monitoring; power capping; RAPL; energy efficiency; power measurement; Cray XC30; Cray XC40**

I. Introduction

Cray Advanced Platform Monitoring and Control (CAPMC) was first described in [1] in section IV-A "Workload manager (external) power monitoring and management interface". Initial CAPMC functionality shipped with the SMW 7.2.UP02 and CLE 5.2.UP02 releases in the fall of 2014. At the time of this writing, CAPMC functionality has been or is being integrated into the offerings of workload management (WLM) software products that support Cray XC systems. With the monitoring and control functionality exposed with the CAPMC service, Cray's WLM partners are enabled to develop innovative solutions to tackle customers' power and energy problems and provide them with powerful telemetry data from their systems for system administrators and users alike.

This work is organized as follows: In section II, we outline some of the use-case drivers considered during the initial design and development of CAPMC. These use cases provide the base functionality needed by WLMs to deliver high value power monitoring and management functionality on Cray systems. In section III, we describe the backend infrastructure, security framework, and client frontend of the CAPMC service. Section IV reviews the features of the client frontend and their parameters and output. Finally, in section V, we describe the future direction of CAPMC.

II. Capabilities and Use Cases

The development of CAPMC was initially driven by the need to enable WLM software partners to provide customers with advanced power monitoring and management capabilities. In [2], the authors present a detailed set of power/energy related use cases for HPC systems. In [3], the authors describe "Power/Energy Measurement and Control Scenarios" of interest for the Trinity system.

In this section we will first describe some basic use cases and then look at how they may be combined to support higher level use cases.

In all of the following use cases the actor is the WLM. The text will focus on a brief description of the use-case scope, the basic flow that the WLM software would be expected to take, and other considerations where appropriate. There are likely other actors that could drive some of these use cases.

A. System power monitoring

Capability: The CAPMC ability to access minimum, maximum, and average system- and cabinet-level power data for a specified time period.

Use case: WLM periodic collection of system-level power consumption for workload-aware power utilization tracking, capacity planning, and WLM driven analytics and logging.

Code flow for periodic system power monitoring:

1) The WLM selects a timing interval, likely in the range of five minutes to an hour.
2) The WLM sets up an internal timer, or otherwise schedules periodic polling of system power usage data at the chosen interval.
3) The WLM calls CAPMC and requests system power data for the chosen interval.
4) CAPMC validates the caller's credentials and returns the requested data in a JSON-formatted data structure.
5) The WLM validates the return code in the response, then extracts data from the JSON-formatted data structure.
6) The WLM updates internal logs, data structures, and external clients with updated system power information.

Alternate flows or considerations: The WLM may want or need to track power at the cabinet level, for all or selected sets of cabinets in the system. CAPMC provides cabinet-level power data with the same infrastructure and calling

convention as used to query system power. The cabinet-level call returns minimum, maximum, and average power data for each cabinet in the system.

B. Node power/energy monitoring

Capability: CAPMC supplies the ability to collect node-level energy data for selected nodes in the system for a specified time period. CAPMC also provides flexibility to WLMs in how the nodes are specified, how the time period is specified, and what level of abstraction the data is returned in.

Use case: WLM on-demand collection of power and energy statistics for a running or recently completed application.

Code flow for responding to a user request for application energy statistics from a generic WLM:

1) The WLM receives a request for power/energy statistics data from a user for a running application.
2) The WLM validates that the user making the request is authorized to access data for the requested application.
3) The WLM calls CAPMC requesting application-level energy statistics for all of the nodes assigned to the application.
4) CAPMC validates the request, then gathers the required data and calculates the requested energy statistics, returning data to the WLM in a JSON response data structure.
5) The WLM validates the return code in the response, then extracts data from the JSON formatted data structure and returns relevant data to the user in whatever format is specified in the WLMs API.

Alternate flows or considerations: In the example above, the WLM may be calling CAPMC on behalf of an unprivileged user. It is the WLM's responsibility to implement policy on behalf of the site with respect to who should or should not have access to the data. We expect WLM software to excel at handling such site-level policy. CAPMC provides the capability to access the data and is not intended to implement policy.

The current release of CAPMC provides three applets that support requests for node-level data at different levels of abstraction. The applets are described in more detail in section IV. All node-level energy data currently supported by CAPMC are collected out-of-band by Cray's Hardware Supervisory System (HSS). Their collection is not a source of jitter or noise that could impact application performance.

C. Node power on / off control

Capability: CAPMC provides the ability to cleanly shut down, then power off and power on, then reboot selected compute nodes. This capability allows for WLM software to manage the number of compute nodes that are ready to run jobs. The WLM can use CAPMC to shut down and power off idle nodes when they will not be needed for some reasonable amount of time, and then request that they be powered on and rebooted in time to be used when needed. In [4] this use case is addressed from the SLURM WLM's point of view.

Use case: The WLM dynamically manages a pool of available idle compute nodes so that when possible there is a configurable number of nodes ready to run new jobs at all times, and that the number of idle nodes in the system is not excessive.

Code flow (simplified) for managing a pool of idle compute nodes:

1) Launch available workload until all pending jobs are started, no idle nodes are available, or some other resource constraint or limit is hit.
2) Place nodes that have just become "idle" back into the idle pool.
3) If the pending-restart or pending-shutdown pools are not empty, call into CAPMC for updated status on nodes in the pending pools. Use the updated status information returned by CAPMC to create a list of nodes that are newly transitioned into the "ready" or "off" states.
4) Place nodes that have just become "ready" into the idle pool, and remove them from the "pending-restart" pool.
5) Place nodes that have just become "off" into the powered-off pool, and remove them from the pending-shutdown pool.
6) If the number of nodes in the idle and pending-restart pools is below a lower limit, and there are nodes available in the powered-off pool, then call CAPMC and request a configurable number of nodes be powered on, placing those nodes into the restart-pending pool.
7) If the number of nodes in the idle pool is above an upper limit, and the scheduler is not waiting for nodes to launch a large pending job (or a job that will launch in the near future) then call CAPMC and request a configurable number of nodes be shutdown and powered off. Moving the requested nodes from the idle pool into pending-restart pool.

Alternate flows or considerations: One consideration worth mentioning with respect to the node power on/off capability is the amount of time it takes to complete the operations, and that latency's impact on that capability's usage. CAPMC provides an applet that allows the WLM to query information about expected latency and other parameters that we expect to be useful when using this feature.

D. Power capping control

Capability: The CAPMC ability to support dynamically managing node-level power capping of compute-nodes.

Use case: WLM support of a system with multiple job queues, where each queue represents nodes that are capped with a different maximum node-level power setting. The

WLM uses CAPMC to dynamically modify node power caps when assign nodes to queues to match demand and site level policy consideration.

Code flow illustrating an implementation with three dynamically managed job queues:

1) Assume a system with a total of 1000 identically configured compute nodes with a worst case power draw of 400 watts and an idle power draw of 50 watts.
2) Assume a job queue named "unlimited" with no power limit and an initial allocation of 0 compute nodes.
3) Assume a job queue named "300W" with a 300 watt power limit and an initial allocation of 0 nodes.
4) Assume a job queue named "200W" with a 200 watt power limit and an initial allocation of 1000 nodes.
5) Assume a site-level policy that charges users based on total energy allocated to compute nodes assigned to their jobs. That is, they get charged more for a 10 node job that runs for 10 minutes on the "unlimited" queue than the same job running for 10 minutes on the "300W" queue even if the actual power consumption is the same.
6) Assume a site-level policy that gives preference to jobs in the "200W" queue between 8:00 AM and 5:00 PM on week days.
7) Each time the scheduler runs it can adjust the number of nodes assigned to each of the three queues to meet demand by calling CAPMC to adjust the power cap settings for compute nodes that are reassigned.
8) The WLM calls CAPMC and collects job statistics including energy used and average power per node, then provides feedback to the user to help them select the most cost effective queue for their next job run.
9) Over time data collected can be used to create more queues, or to adjust the power cap settings of available queues to match the needs of the site and it's users.

Alternate flows or considerations: While the above example is simple, it may be worth more investigation with respect to making users more aware of job energy usage and how awareness and pricing might affect system throughput and total cost of ownership.

E. Power aware scheduling of an over-provisioned system

Capability: CAPMC enables WLMs to leverage dynamic system power monitoring and control capabilities to manage an over-provisioned system. In [5] and [6], the authors describe over-provisioned systems and their related research.

Use case: The WLM schedules jobs in a power aware manner that maximizes system productivity while preventing total power consumption from exceeding a site imposed maximum power limit. The maximum power limit in this example is static, but in a production implementation the limit would be expected to be adjusted over time. This use case uses CAPMC system and node level power monitoring and node-level power capping. The goal for power capping in this example is to allow the scheduler to treat power as a consumable resource. Power capping in this example is not intended to limit job or workload performance.

Code flow example for power aware scheduling of an over-provisioned system:

1) Assume a system operating under a 2 megawatt power cap that has 8000 identical nodes with:
   - Maximum application observed un-capped power draw of 400 watts per node.
   - Minimum usable node-level power cap of 200 watts per node.
   - Idle power draw of 50 watts per node.
   - Zero power draw when powered off.

   The scheduler manages pools of idle and powered-off nodes.
2) Assume a starting system state where:
   - New jobs can be submitted with a requested power cap (call it UPCAP). UPCAP is then not increased by the scheduler.
   - 2125 nodes are powered off.
   - 1000 nodes are idle.
   - 4875 nodes are allocated to running jobs.
   - All running jobs are launched with nodes capped at 400 watts.
   - All available power is allocated.
3) Assume that the scheduler has access to system- and job-level power data collected by periodically calling CAPMC. The following power statistic are assumed to be available to the scheduler:
   - System-level minimum, average, and maximum power data for rolling 24, 4, 1, and .25 hour intervals.
   - Per-job minimum, average, and maximum power data for the job lifetime, and for 60, 30, 15, and 1 minute intervals.
   - Per-job maximum remaining runtime (call it RRT), likely via a policy limit at job launch time.
4) Top-of-loop:
5) Look at pending work and determine how aggressive to be in lowering per-job power caps.
6) Calculate new power cap targets for all jobs using rules like the following:
   - If RRT $\leq$ 1 minute, do not update the job power cap.
   - If the job has been running $\leq$ 1 minute, do not update the job power cap.
   - If the job's power cap has been updated in the past minute, do not update the job power cap.
   - Target $\leq$ 400 watt maximum.
   - Target $\leq$ UPCAP.
   - Target $\leq$ current power cap + 7 watts.
   - Target $\geq$ current power cap - 5 watts.
   - Target $\geq$ job lifetime average power + 10 watts.

- Target $\geq$ job 60 minute average power + 7 watts.
- Target $\geq$ job 30 minute average power + 5 watts.
- Target $\geq$ job 15 minute average power + 4 watts.
- Target $\geq$ job 1 minute average power + 4 watts.
- Target $\geq$ 200 watt minimum.
- If aggressive power capping is selected, choose the lowest valid target, otherwise choose the highest valid target power cap.

7) Apply updated power capping targets for all jobs. Note that all nodes in multi-node jobs get the same power cap. This step is also complicated in that any adjustments that increase power cap levels must not cause a violation of the system level power cap.
8) Manage idle and powered-off node pools.
9) If there is no new work that can be scheduled, goto Top-of-loop.
10) Assign nodes to new jobs with the default 400 watt power cap unless the user requested a lower power cap.
11) Goto Top-of-loop.

Alternate flows or considerations: There are many policy-driven choices to consider when managing over-provisioned HPC systems. As the cost of compute-node hardware drops in comparison with other costs in operating HPC systems, the number and scale of over-provisioned systems is expected to increase. This increase will drive continued need for innovation in both WLM software and the monitoring and control capabilities provided by CAPMC.

III. Architecture

To the outside world, CAPMC appears much like a typical web services API operating over HTTP. Clients POST HTTP requests containing payloads formatted in JavaScript Object Notation (JSON) [7]. Internally CAPMC translates HTTP API requests into platform monitoring and control operations such as node boot, shutdown, power-cap set or get, job profiling, or job-energy reporting.

CAPMC, the service, is implemented using standardized off-the-shelf technology where applicable. Additionally, it contains a custom bridge interface into the proprietary Cray HSS. *capmc*, the script, is the remote command line interface to CAPMC, the service. The client access script is implemented in Python with the only dependency being the Python 2 standard library. The high-level architecture of the CAPMC service is pictured in Figure 1.

A. Remote Access

The officially supported client interface is the capmc utility. However, the following shell command most clearly demonstrates access control and the end-to-end communication path. The example presented in Figure 2 demonstrates how a JSON-formatted object containing a node ID (nid) list is posted via a third-party tool to the CAPMC service handler which returns component state and result status.

```
shell:~> curl \
  -H 'Content-type: application/json' \
  -X POST --cacert certificate_authority.crt \
  --cert client.crt --key client.key  \
  -d '{"nids":[1,2,65]}' \
  https://<smw-hostname>:8443/capmc/get_node_status
{
    "e": 0,
    "err_msg": "",
    "disabled": [65],
    "ready": [1,2]
}
```

Figure 2: Example API call using curl

Additionally, the example presented in Figure 2 demonstrates several key concepts. The "Content-type" header and HTTP "POST" verb indicate to the server that there will be an application-specific JSON payload in the request body. The client side SSL library will verify that the server host certificate was issued by the X.509 [8] authority identified in the `certificate_authority.crt` PEM file, and that the server certificate's common name field matches the DNS host name specified in the URL. Because a client certificate and private key are also specified as command line options, the server side SSL library will perform the same type of validation on the specified `client.crt` PEM file.

B. Security Considerations

The CAPMC HTTP server, nginx [9], is configured by default to require SSL and reject connection requests which do not supply a valid X.509 client certificate. All X.509 certificates must be validated against the preconfigured authority. Therefore, only if both sides of the connection successfully validate each other will the request be allowed to proceed. This type of two-way authentication is a standard capability build into most HTTP servers and clients such as nginx, curl, Python's standard library, or a typical web browser.

The X.509 certificate authority is automatically generated and configured at SMW install time using several invocations of OpenSSL and Cray specific setup scripts. The files are unique to each installation. An additional host certificate and private key are created for use with nginx, enabling encrypted communications. A client access certificate and private key are generated as well, for use with the remote command line utility.

Ultimately it is the duty of the system administrator to take appropriate security precautions when distributing and configuring the client access private key file. For example, a pass-phrase may be added to the private key before installation to a remote system if the files are to be transported via an insecure medium. Upon deployment appropriate file permissions must be set on the private key such that only the intended user may have read access. All private keys must remain private.

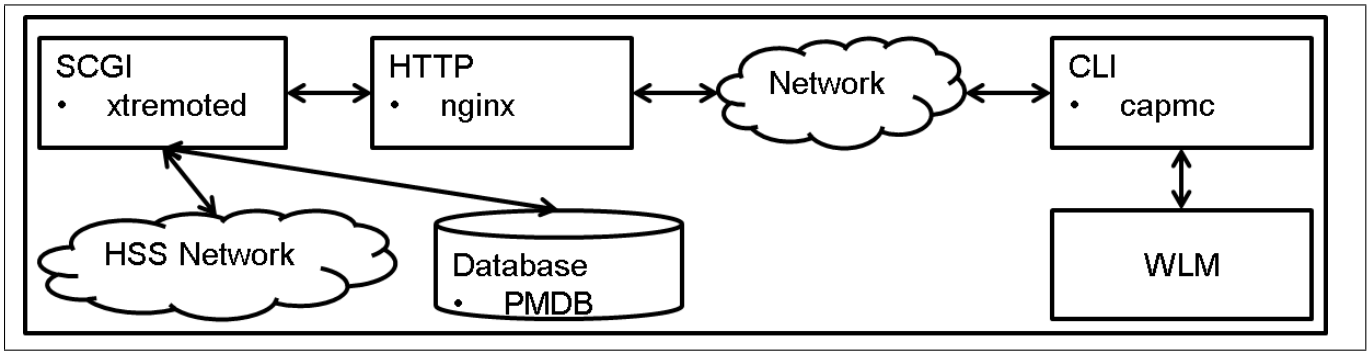

Figure 1: CAPMC Architecture

C. Command Line Interface

As previously mentioned, the officially supported command line interface is called capmc. Internally it operates much as the command example using curl. It performs all the same actions with respect to X.509 certificate validation, encrypted communication, and JSON data structure passing. The primary difference is that it can construct and transmit appropriate JSON data structures based on user specified command line arguments. The example presented in Figure 3 accomplishes the same task as in Figure 2, only it does so with a much simpler syntax.

```
shell:~> capmc node_status --nids=1-2,65
{
    "e": 0,
    "err_msg": "",
    "disabled": [65],
    "ready": [1,2]
}
```

Figure 3: Example API call using capmc

The simpler syntax is a direct result of having preconfigured parameters with respect to X.509 certificate path locations, the SMW service URL, and operation specific command line argument parsing. The configuration syntax demonstrated in Figure 4 conforms to JSON. All parameter values are represented using the string data type.

```
shell:~> cat /etc/opt/cray/capmc/capmc.json
{
"os_key":
    "/etc/opt/cray/capmc/client.key",
"os_cert":
    "/etc/opt/cray/capmc/client.crt",
"os_cacert":
    "/etc/opt/cray/capmc/certificate_authority.crt",
"os_service_url":
    "https://example-smw.us.cray.com:8443"
}
```

Figure 4: Example capmc client configuration file

For more advanced use cases or cases where third party integrators may find it simpler to construct data structures themselves, capmc supports reading the request payload via standard input. The only required argument identifies the resource which will handle the request. The example in Figure 5 demonstrates the same request as in Figures 2 and 3 using standard input redirected from a string.

```
shell:~> capmc json \
  --resource=/capmc/get_node_status <<< \
  '{"nids":[1,2,65]}'
{
    "e": 0,
    "err_msg": "",
    "disabled": [65],
    "ready": [1,2]
}
```

Figure 5: Example API call using capmc standard I/O

D. Application Server

The CAPMC server-side components utilize a widely popular HTTP server, nginx. The CAPMC infrastructure uses nginx in one of its common deployment roles. It provides encryption and user authorization capabilities to an independent, application-specific server. In this case, that application-specific server is called *xtremoted*. It is the bridge between the external world and the proprietary Cray HSS.

The two components, xtremoted and nginx, communicate with one another over a UNIX domain stream socket using an open protocol known as the Simple Common Gateway Interface or SCGI [10]. The protocol allows interoperability between off-the-shelf HTTP servers and long running stateful servers that don't natively support HTTP, encryption, or one of the many standardized HTTP authorization schemes. Internally xtremoted is a simple command dispatcher. It receives a request, parses the input parameters, completes or rejects the request, and returns a response.

IV. Applets

The CAPMC service implements multiple applets to meet the use cases described in section II among others. In the following we review the available applets in terms of their

usage in *controlling* and *monitoring* the Cray system. As discussed in the section III, the client for CAPMC service is the capmc command [11]. Through the capmc command, WLMs and other authorized software or users can execute the following applets. Upon successful transmission of a request to the CAPMC service, the capmc command awaits a JSON-formatted response. That response always includes an "e" number and an "err_msg" string. An "e" value of zero represents success, while any non-zero values are accompanied by a non-empty "err_msg" string describing the failure or error.

A. Control Applets

Cray presents node-level control for powering on or off compute nodes, setting node-level power caps, querying node state information, and Cray- and site-specific CAPMC service usage rules.

1) Node-level Power Controls: Given its knowledge of the system's job queues, WLMs may intelligently manage power using node-level power controls. The simplest management may be to turn off compute nodes which may be idle for a significant time interval. For this purpose, CAPMC presents the WLMs with two applets: *node_on* and *node_off*.

The *node_off* applet cleanly shuts down and powers off a specified set of nodes, while the *node_on* applet powers on and boots a specified set of nodes into a usable state. The *node_off* and *node_on* applets are implemented as non-blocking operations, in that the service completes after communication of the request to the system. These applets only fail if that communication fails or invalid parameters are detected. Only compute nodes may be controlled in this way. Example invocations of *node_off* and *node_on* are shown in Figures 6 and 7, respectively.

```
shell:~> capmc node_off -n 1-5,99
{
    "e": 0
    "err_msg": ""
}
```

Figure 6: Example usage of *node_off*: Shutdown and power off nodes 1, 2, 3, 4, 5, and 99.

```
shell:~> capmc node_on -n 1-5,99
{
    "e": 0
    "err_msg": ""
}
```

Figure 7: Example usage of *node_on*: Power on and boot nodes 1, 2, 3, 4, 5, and 99.

2) Node-level State: Because the *node_on* and *node_off* applets are asynchronous in nature, Cray provides the *node_status* applet which can be used to return node states for the system or a subset of specified nodes. Additionally, given a filter, nodes may be queried by state. The states available through this applet mirror those in the Cray HSS: 'on', 'off', 'halt', 'standby', 'ready', 'diag', 'disabled'. Once a node is gracefully shutdown and powered off using the *node_off* applet, that node is in the 'off' state. Once a node is powered on and booted into a useable state, that node is in the 'ready' state. An example *node_status* invocation is shown in Figure 3.

The *node_rules* applet informs the third party software about hardware (and perhaps site-specific) rules and timing constraints that allow for efficient and effective management of idle node resources. The data returned by the *node_rules* command informs the caller of how long "on" and "off" operations should be expected to take, the minimum amounts of time nodes should be left off to reasonably save energy, and (optionally) limits on the number of nodes that should be turned on or off. Cray supplies values for these rules where appropriate, specifically the expected latencies for *node_on* and *node_off* and the minimum amount of time a node should be off. Other values like the maximum node counts for *node_on* or *node_off* and the maximum amount of time a node should be left off are left unset. Site administration may customize these values by editing the `rules.ini` file on their SMW under `/opt/cray/hss/default/etc/xtremoted`. It is important to note that these rule values are not strictly enforced by the software. Instead, they are meant to provide guidelines for authorized callers in their use of the CAPMC service. An example *node_rules* invocation is shown in Figure 8.

```
shell:~> capmc node_rules
{
    "e": 0,
    "err_msg": "",
    "latency_node_off": 60,
    "latency_node_on": 600,
    "max_off_req_count": -1,
    "max_off_time": -1,
    "max_on_req_count": -1,
    "min_off_time": 900
}
```

Figure 8: Example usage of *node_rules*: Query the rules for node-level power control.

3) Node-level Power Capping Controls: The *get_power_cap_capabilities*, *get_power_cap*, and *set_power_cap* applets allow for third-party software management of node level power capping. These three applets enable flexible, efficient node-level and accelerator-level capabilities that can support multiple use cases without enforcing policy.

Given a list of nodes, the *get_power_cap_capabilities* applet returns information about power capping capabilities, controls, and valid ranges. These capabilities are returned in a structured way where information is grouped for all

cases where the hardware is common. Thus, even though the call may request capabilities for all of the compute nodes in a system, the maximum response size is limited to one group for each hardware node configuration in the system, and all nodes in each group are listed. On current two socket XC nodes, there is a single 'node' control that has minimum/maximum range information in watts. Cray XC nodes with accelerators have an additional 'accel' control that also has minimum/maximum range information in watts. An example invocation of *get_power_cap_capabilities* is shown in Figure 9.

```
shell:~> capmc get_power_cap_capabilities \
         -n 761-763
{
  "groups":[{
    "name":
    "01:000d:306f:00f0:0018:0080:0855:0000",
    "desc":
    "ComputeANC_HSW_240W_24c_128GB_2133_NoAccel",
    "supply":425,
    "host_limit_min":180,
    "host_limit_max":360,
    "static":0,
    "powerup":140,
    "controls":[{
            "name":"node",
            "desc":"Node manager control",
            "min":180,
            "max":360
        }],
    "nids":[
        761,
        762,
        763
    ]
  }],
  "e":0,
  "err_msg":""
}
```

Figure 9: Example usage of *get_power_cap_capabilities*: Query power capping capabilities, controls, and valid ranges for nodes 761 through 763.

The *get_power_cap* applet returns the power-capping control(s) and current settings for all requested compute nodes. Note that a power cap setting (value) of zero has a special meaning of *'not-capped'*. The *set_power_cap* applet allows the authorized caller to set the same controls that are returned by *get_power_cap* within the minimum/maximum constraints returned by get_power_cap_capabilities. Example invocations of *get_power_cap* and *set_power_cap* are shown in Figures 10 and 11, respectively. If setting multiple different power caps is desired, it is recommended that those be set programmatically using the *json* applet as shown in Figure 5, which would allow third-party software to pass its own JSON-formatted power cap request in a single transaction with the CAPMC service.

```
shell:~> capmc get_power_cap -n 761
{
    "nids":[{
            "nid":761,
            "controls":[{
                    "name":"node",
                    "val":0
                }]
        }],
    "e":0,
    "err_msg":""
}
```

Figure 10: Example usage of *get_power_cap*: Query the existing power cap on node 761. A node power cap value of zero means there is no cap set.

```
shell:~> capmc set_power_cap -n 761 --node 220
{
    "e":0,
    "err_msg":""
}
```

Figure 11: Example usage of *set_power_cap*: Set the node power cap on node 761 to 220 watts.

B. Monitor Applets

Cray provides a number of flexible applets meant to query useful power and energy statistics from the Power Management Database (PMDB) [1] [12]. System-level applets report power at system and cabinet levels. Node-level applets can report energy statistics at node, job, and application levels. Power is reported in watts, and energy is reported in joules.

1) System-level Monitoring Applets: WLMs and other authorized software or users may want to check the overall system's use of power. WLM software might poll for this type of system power data on each scheduling cycle, at regular intervals, or on-demand from an interactive system workload administrator. To facilitate this, the CAPMC service offers two applets. The *get_system_power* applet is used to access minimum, average, and maximum power for the system over a window of time in the past. The *get_system_power_details* applet is used to request the same statistics as get_system_power but for each of the cabinets in the system. For both applets, the caller may supply a starting time and window length. In the absence of the starting time parameter, a default starting time of "now" minus the window length is used. If no window length is specified, default window length is ten seconds. The ability to access historical data is limited by the size of the system and the amount of resources dedicated to the backing database (PMDB). The maximum valid time window is one hour. Examples of get_system_power and get_system_power_details calls are shown in Figures 12 and 13, respectively.

2) Node-level Monitoring Applets: The *get_node_energy_stats*, *get_node_energy* and *get_node_energy_counter* applets allow flexible querying to node energy data by node list, job ID, or ALPS application

```
shell:~> capmc get_system_power -w 600
{
    "start_time":"2015-04-01 17:02:10",
    "avg":5942,
    "min":5748,
    "max":6132,
    "window_len":600,
    "e":0,
    "err_msg":""
}
```

Figure 12: Example usage of *get_system_power*: Query the average, minimum, and maximum system power values over the last five minutes.

```
shell:~> capmc get_system_power_details \
         -w 600 \
         -s "2015-04-01 17:00:00"
{
    "cabinets": [
        {
            "avg": 5941.3316666666669,
            "max": 6170,
            "min": 5695,
            "x": 0,
            "y": 0
        }
    ],
    "e": 0,
    "err_msg": "",
    "start_time": "2015-04-01 17:00:00",
    "window_len": 600
}
```

Figure 13: Example usage of *get_system_power_details*: Query the average, minimum, and maximum power values for each cabinet in the system over the five minutes prior to 2015-04-01 17:00:00. Cabinets are indexed by their x and y coordinates.

ID (apid). For these commands, the flexibility starts with the option of supplying an apid that the CAPMC service can then use to generate node-list, start-time and end-time information. The caller can also supply an apid or job ID with explicit start time and end time options to get information on a running application or job.

The get_node_energy_stats applet returns total energy for the selected nodes, average energy for nodes in the set, standard deviation of energy for nodes in the set, two ordered pairs of (node, energy) for the minimum and maximum energy nodes, the duration of the interval in seconds, and the node count. This output format is intended to be very useful and efficient when dealing with large node counts, as it can fully leverage the capabilities of the CAPMC service to generate statistics. An example invocation of get_node_energy_stats is shown in Figure 14.

The get_node_energy applet takes the same user inputs as get_node_energy_stats but rather than returning aggregate statistics, it returns the time window in seconds for the queried set, the total node count, and an array of node/energy data pairs with one element for each selected node. This

```
shell:~> capmc get_node_energy_stats \
          --jobid 627929.sdb
{
    "e": 0,
    "energy_avg": 88039.40178571429,
    "energy_max": [
        59,
        102988
    ],
    "energy_min": [
        127,
        15118
    ],
    "energy_std": 24871.40549279,
    "energy_total": 9860413,
    "err_msg": "",
    "nid_count": 112,
    "time": 293.53360400000003
}
```

Figure 14: Example usage of *get_node_energy_stats*: Query job energy statistics for job “627929.sdb”

output format scales with the number of selected nodes, and allows the caller more flexibility in processing the energy data. Both get_node_energy and get_node_energy_stats applets handling of jobs and application timings account for multiple time intervals for the job or application as would be true in the case of suspended and resumed jobs and applications. An example invocation of get_node_energy is shown in Figure 15.

```
shell:~> capmc get_node_energy --apid 122232
{
    "err_msg": "",
    "nodes": [
        {
            "energy": 615,
            "nid": 12
        }
    ],
    "e": 0,
    "nid_count": 1,
    "time": 7.3095230000000004
}
```

Figure 15: Example usage of *get_node_energy*: Query node energy values for a single-node aprun with apid 122232.

The get_node_energy_counter applet requires an explicit point in time and does not calculate energy used over a time interval like the previous two applets. The data returned by get_node_energy_counter are the raw accumulated energy counter values for each selected node. The raw accumulated energy (snapshot) data for any given node are only useful when compared to another snapshot for the same node. This command places the most amount of work in the hands of the caller but allows for the most flexibility. Using this call, third-party software can track total energy of a set of nodes and the energy usage of long running applications, where the runtime of the application may be longer then the history kept in the PMDB. This also would allow third-party WLM

software to directly deal with other advanced use cases like suspend/resume, job migration, etc. Note that this interface can not be used to access data at granularity finer than one second. An example invocation of get_node_energy_counter is shown in Figure 16.

```
shell:~> capmc get_node_energy_counter \
         -n 1008-1010 \
         -t '2015-04-01 23:45:00'
{
    "e": 0,
    "err_msg": "",
    "nid_count": 3,
    "nodes": [
        {
            "energy_ctr": 3865217,
            "nid": 1008,
            "time": "2015-04-01 23:45:00.39833-05"
        },
        {
            "energy_ctr": 3998809,
            "nid": 1009,
            "time": "2015-04-01 23:45:00.39833-05"
        },
        {
            "energy_ctr": 3919829,
            "nid": 1010,
            "time": "2015-04-01 23:45:00.39833-05"
        }
    ]
}
```

Figure 16: Example usage of *get_node_energy_counter*: Query the node accumulated energy counters on nodes 1008 through 1010 at time 2015-04-01 23:45:00.

V. Roadmap

Roadmap considerations mentioned are subject to change, but at this time are expected to become available in the next nine to eighteen months.

Near-term additions to CAPMC fall into two categories. The first is a set of platform controls targeting the ability for WLM to manage hardware configuration settings in upcoming blades. The ability for the WLMs to manage these settings is very important because it allows customers to maximize the value of their system by configuring hardware to match the needs of diverse applications and work flows. Details on all of the actual hardware controls for new blade types are out of scope for this paper. We can however indicate that one of the new CAPMC applets will allow the WLM to reinitialize a node or list of nodes. This new "reinitialize" applet would function in a similar way to:

1) Call CAPMC node_off.
2) Wait for the node(s) to transition to the off state.
3) Call CAPMC node_on.

The reinitialize applet will enable the WLM to modify one or more node hardware settings that require node BIOS reinitialization and then make them take effect.

The second category is a set of in-band controls that will allow WLMs to dynamically control maximum node level c-states, as well as minimum and maximum node-level p-states. These new in-band controls are planned as the first Cray use of node-level controls based on the HPC PowerAPI [13]. This initial support for the HPC PowerAPI functionality on compute nodes will allow the WLM to query c-state and p-state capabilities, read current c-state and p-state limit settings, and write new c-state and p-state limits.

There are many drivers for dynamic control of p-states. The planned support in CAPMC for managing node-level p-state minimum and maximum frequency will give flexibility to WLM software. This new p-state limiting capability will integrate cleanly with Cray's current ALPS support for setting a p-state via APBASIL or the aprun p-state=$< freq >$ command line options. We also believe this feature will allow the WLM to set boundaries for future application and/or run-time p-state controls, where the local node-level actors should be constrained by the higher authority WLM software. From a system power management prospective, dynamic c-state and p-state limiting is an important control for implementing system power-band management.

VI. Conclusion

In this work, we outlined some of the use cases that drove the design and development of CAPMC. Further, we described the underlying architecture and features – its backend infrastructure, security framework, client frontend, and applets. Finally we described forward-looking, near-term additions to CAPMC.

Cray continues to invest heavily in providing and building up novel power and platform management and monitoring features. CAPMC is one of the fruits of that investment. With this service, WLMs and authorized callers are able to monitor and control Cray systems like never before. Power budgets can be managed. System power can be monitored. WLMs can communicate to users energy statistics for their jobs and applications. It is our hope that this work provides Cray customers and administrators with exciting new tools to implement power policies through their chosen WLMs. We also hope that through their WLMs, Cray users will avail themselves of these capabilities providing novel ways to look at their jobs, applications, and usage of the system moving toward exascale.

Acknowledgments

We would like to thank all of engineers, testers and others who have worked on Cray XC power monitoring and management. Without their hard work and attention to detail, this paper would not have been possible. We would also like to thank the customers who have encouraged us to provide enhanced power monitoring and control capabilities. Ongoing customer input and dialogue is critical to our future success. We would like to especially thank our partners and colleagues at Sandia National Labs for their contribution to

the use cases that informed the design and development of CAPMC.

REFERENCES

[1] S. Martin and M. Kappel, "Cray XC30 Power Monitoring and Management," *Proceedings of CUG*, 2014, (Accessed 23.March.15). [Online]. Available: https://cug.org/proceedings/cug2014_proceedings/includes/files/pap130.pdf

[2] J. H. Laros, III, S. M. Kelly, S. Hammond, and K. Munch, "Power/Energy Use Cases for High Performance Computing," SANDIA REPORT SAND2013-10789 Unlimited Release Printed December, 2013, Accessed 26.March.15. [Online]. Available: https://cfwebprod.sandia.gov/cfdocs/CompResearch/docs/UseCase-powapi.pdf

[3] "Trinity/NERSC-8 Use Case Scenarios," (Accessed 1.Apr.15). [Online]. Available: https://www.nersc.gov/assets/Trinity--NERSC-8-RFP/Documents/trinity-NERSC8-use-case-v1.2a.pdf

[4] "SLURM: Power saving guide," (Accessed 1.Apr.15). [Online]. Available: http://slurm.schedmd.com/power_save.html

[5] A. Langer, H. Dokania, L. V. Kalé, and U. S. Palekar, "Analyzing energy-time tradeoff in power overprovisioned hpc data centers," *Urbana*, vol. 51, pp. 61 801–2302, (Accessed 2.Apr.15). [Online]. Available: http://charm.cs.illinois.edu/newPapers/15-11/paper.pdf

[6] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski, "Exploring hardware overprovisioning in power-constrained, high performance computing," in *Proceedings of the 27th international ACM conference on International conference on supercomputing*. ACM, 2013, pp. 173–182, (Accessed 2.Apr.15). [Online]. Available: https://e-reports-ext.llnl.gov/pdf/752192.pdf

[7] D. Crockford, "RFC 4627: The application/json Media Type for JavaScript Object Notation (JSON)," 2006.

[8] R. Housley, W. Ford, W. Polk, and D. Solo, "RFC 2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile," *Network Working Group–Internet Engineering Task Force*, 1999.

[9] "nginx," (Accessed 2.Apr.15). [Online]. Available: http://nginx.org/en/

[10] N. Schemenauer, "SCGI: A Simple Common Gateway Interface alternative," (Accessed 2.Apr.15). [Online]. Available: http://www.python.ca/scgi/protocol.txt

[11] *capmc(8) System Management Workstation (SMW) 7.2.UP02 Man Pages*, Cray Inc., October 2014.

[12] "Monitoring and Managing Power Consumption on the Cray XC30 system, Cray publication S-0043-72," (Accessed 11.Apr.14). [Online]. Available: http://docs.cray.com/books/S-0043-72/S-0043-72.pdf

[13] J. H. Laros, III, D. DeBonis, R. Grant, S. M. Kelly, M. Levenhagen, S. Olivier, and K. Pedretti, "High Performance Computing - Power Application Programming Interface Specification Version 1.0," SANDIA REPORT SAND2014-17061 Unlimited Release Printed August, 2014, Accessed 27.March.15. [Online]. Available: http://powerapi.sandia.gov/docs/PowerAPI_SAND.pdf