Power capping of heterogeneous systems

1st Andrew Nieuwsma
HPC, AI, and Labs
Hewlett Packard Enterprise
Minneapolis, United States
andrew.nieuwsma@hpe.com

2nd Dr. rer. nat. Torsten Wilde
HPC, AI, and Labs
Hewlett Packard Enterprise
Munich, Germany
wilde@hpe.com

***Abstract*—The landscape of HPC is changing rapidly because of rising energy prices and concerns of increased OPEX, regulatory concerns around data center sustainability (reduction of carbon footprint, total power burden on the grid), and the expected increase in system power consumption as systems get larger and component power requirements increase. Customers are asking for solutions that help them manage the changing landscape. Distributing a system power cap for the different nodes of a heterogeneous system is challenging. Providing different power distributions based on different allocation policies is challenging without an automated tool. The System-Power-Capping tool presented in this paper is part of HPE's vision toward holistic system power management software stack. The tool allows HPE and the customer to set a system power cap and to allocate node power based on different distribution algorithms and customer allocation policies, simplifying the process of setting power caps on a heterogeneous system. This paper presents experimental results that demonstrate the efficiency of various distribution algorithms using a customer hardware configuration as reference. The recursive applicability of the approach (from data center to an individual node) can potentially help customers to improve the overall power allocation in the future.**

***Index Terms*—power capping, HPC system management, holistic system power and energy management**

I. Introduction and Background

With the transition into exascale computing, system power consumption is becoming a more critical component of system operation to manage. The first Exascale system (called Frontier) at the Oak Ridge National Laboratory (ORNL) consumed around 21MW when running the Linpack benchmark (number 1 system on the Nov 2022 Top500 list [8]). An analysis of the previous supercomputing system at ORNL (called Summit) showed that the systems average power consumption was around 50% of Linpack power [9]. The difference between normal operating power consumption and the provisioned power and cooling capabilities by the facility for a specific HPC system is becoming an operational concern. One way to reduce stranded and/or trapped power and cooling capacity in HPC data center is hardware over-provisioning (installing more compute nodes than could be operated when running Linpack) requiring a trusted enforcement of a system power cap.

Setting a power cap on homogeneous systems in the most basic way is a relatively straightforward process since all nodes will get the same power share. With the explosion of different accelerators and the need for customers to support a wide variety of workloads as efficiently as possible, heterogeneous systems will become a standard HPC system architecture. In addition, node architectures are moving towards heterogeneous compute devices as well. The implications are that: different node architectures with different processing components will have different min/max power boundaries; different power cap values will have different impacts on node performance; and a system power cap cannot just be split according to the number of nodes.

$$SystemPower = \sum_{i=1}^{S} notControllableConsumers_i + \sum_{j=1}^{N}(BaseNodePower_j + \sum_{k=1}^{C} CPU_{jk} + \sum_{l=1}^{A} Accelerator_{jl}) \quad (1)$$

With:

- $\sum_{i=1}^{S} notControllableConsumers_i$ represents the sum of the power consumption of support infrastructure (e.g. power distribution, system cooling) and system components that either cannot be, or should not be controlled such as login nodes, network equipment, system management controllers, I/O subsystem etc. for which the name plate power or any alternative safeguard power value is used.
- $\sum_{j=1}^{N}()$ represents system components whose power consumption can be controlled (e.g. representing mainly the compute nodes for an HPC systems)
- $BaseNodePower_j$ represents the sum of all not controllable consumers on a node
- $\sum_{k=1}^{C} CPU_{jk} + \sum_{l=1}^{A} Accelerator_{jl}$ represent node components whose power consumption can be controlled (e.g. individual compute units on the nodes)

The system power consumption can be expressed as the sum of two terms - Formula (1). One term $\sum_{i=1}^{S} notControllableConsumers_i$ defines the maximum possible power consumption of consumers that cannot be controlled and the other $\sum_{j=1}^{N}(BaseNodePower_j + \sum_{k=1}^{C} CPU_{jk} + \sum_{l=1}^{A} Accelerator_{jl})$ defines the sum of the power consumed by any controllable compute node. Compute node power is defined as the sum of a fixed base power

	Node Type 1	Node Type 2
Node Architecture	Homogeneous	Heterogeneous
Node Composition	2 CPU, 0 GPU	1 CPU, 4 GPU
Min Power Cap in Watts	350	764
Max Power Cap in Watts	925	2754
Max - Min Power Cap (Delta) in Watts	575	1990
# nodes in system	1536	2560

TABLE I: Example system: heterogeneous hardware power capping ranges

Total Node Count	4,096
Sum_Max	8,471,040 watts
Sum_Min	2,493,440 watts

TABLE II: Example system: summary

consumption (which, depending on the compute unit design, could include memory power but this could change for HBM) and the power consumption of the different compute units on the node.

Setting a useful power cap on a heterogeneous system with heterogeneous nodes is challenging. For example, considering the power ranges for heterogeneous hardware in Table I, the standard approach would take a system power cap (representing the allowed combined power consumption of all compute nodes in the HPC system), divide it by the number of compute nodes, and set the uniform power cap on all compute nodes. As can be seen, depending on the system power cap, there is a potential of little to no overlap between the allowed node power ranges for different hardware architectures. It is very hard to find a 'universal' power cap that could be applied to all hardware in a heterogeneous system. Furthermore, the likelihood that a uniform node power cap calculated from a system power cap would fall within the allowed node power limits becomes smaller the more diverse a heterogeneous HPC system becomes.

II. Existing Solutions

Current solutions for managing the power and/or energy consumption of HPC systems are focused on managing only the "compute nodes" and do not interact with any existing out-of-band (OOB) power management system. Therefore, it is challenging to use those solutions to manage any system level constraints in a trusted manner.

The open source software EAR (Energy Aware Runtime [1] [2]) is a library for optimizing the energy efficiency of a job by optimizing the energy efficiency of the nodes of that job. Currently, most features are aimed towards the support of Intel CPUs but limited support for AMD CPUs and NVIDIA GPUs is provided. The main features are: energy accounting, energy/power control, energy optimization, and active system power capping. EAR intercepts MPI calls, identifies the outer loop and tries to compute a signature. It matches this signature with a time and power model, and depending on specified energy policy selects a frequency for the loop. EAR provides very limited functionality for non MPI/openMP workloads (reporting but not optimization). Currently, EAR does not support energy optimization together with system level power management.

GEOPM (Global Extensible Open Power Manager [3] [4]) is an open source framework for exploring power and energy optimization of HPC jobs. Currently, GEOPM targets only MPI and OpenMP application. It identifies loops and determines an optimal RAPL (Running Average Power Level) setting for the loop according to a set optimization policies. GEOPM can optimize power usage, performance under a given power cap, and trade off performance for energy savings. As GEOPM does not consider job histories, it has to run the detection for every loop again leading to the recommendation to run GEOPM on a dedicated core. Currently, GEOPM has no feature to manage power at a system level. It expects a given application power cap per job and manages the job inside this cap.

Another competitive approach is the Bull Dynamic Performance Optimizer (BDPO [5] [6]) by ATOS. It aims to optimize an application's energy consumption at runtime. BDPO uses a simple hardware performance counter threshold to set a low or high execution frequency depending on compute load. The low and high frequency to use are defined in a configuration file. Currently, BDPO cannot manage a job according to a power constraint and is not able to manage a system power budget.

III. Solution and Distribution Algorithms

The approach presented in this paper enables a system administrator (SysAdmin) to set a system power cap without needing to understand the intricacies of heterogeneous system architecture. It provides an automatic mechanism to intelligently set node power caps according to a specified system power cap and power allocation policy. The system power cap is distributed into individual node power caps according to system/node characteristics and SysAdmin defined trade-offs providing the best possible node power distribution for homogeneous and heterogeneous system using the out-of-band (OOB) system control.

A. High level solution

Figure 1 shows the high-level representation of the prototype solution. The solution is able to determine the best node power cap distribution for homogeneous and heterogeneous HPC systems with a homogeneous or heterogeneous node architecture according to user definable policies and extensible distribution algorithms. The system can apply the power cap to each individual compute node inside a HPC system. The proposed system and solution can be used by in-band application aware power and energy management software, e.g. GEOPM , in combination with hardware provided power control interfaces and hardware-based node power distribution logic (static and dynamic), to set optimal node power guardrails according to a system power cap and application power requirements.

While using the Out-of-Band (OOB) interfaces, the potential rate of change is limited by the speed and quality of service provided by the OOB interfaces, for the systems in question, Redfish. While Redfish-based controls can be employed multiple times per minute, the exact implementation varies across vendors, and sub-second timing control is usually not feasible, which is common for HTTP-based protocols. Given that the average job duration for HPC applications exceeds an hour [7], a constraint on the order of seconds is considered acceptable.

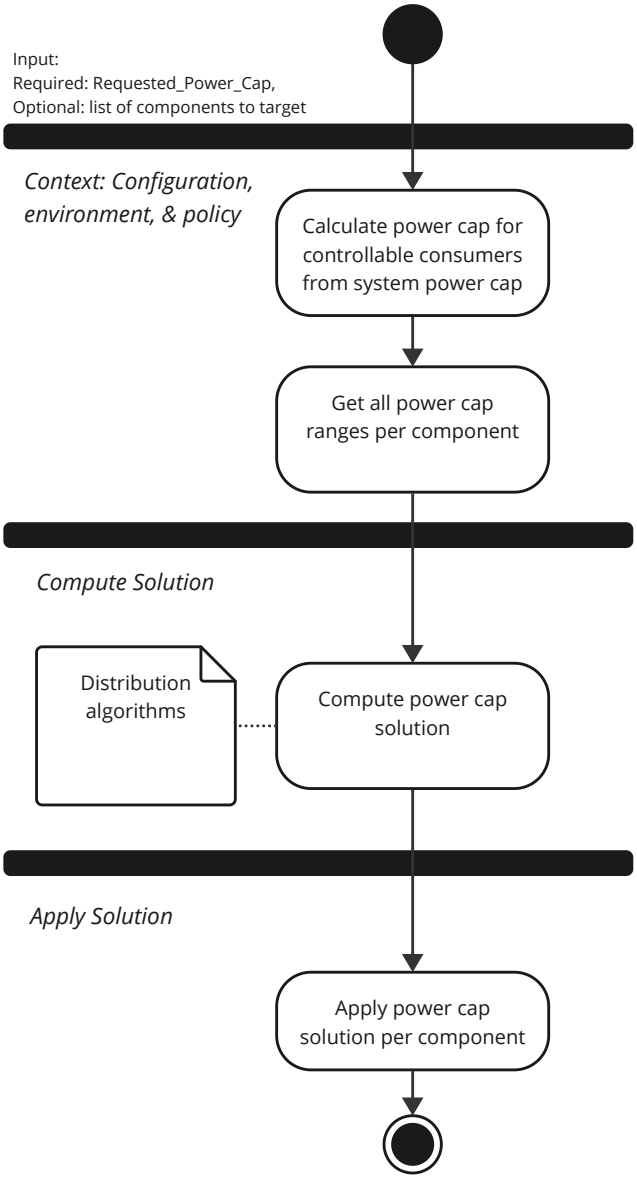

Fig. 1: overall solution algorithm

The overall solution algorithm (Figure 1), works as follows:

1) From the input of *Requested_Power_Cap* and an optional list of components to target
2) Determine the execution context: configuration, customer distribution policy, and the current environment (universe of not controllable consumers & controllable consumers)
   a) Enumerate the individual power cap ranges for each component and calculate the *Sum_Min* (sum of minimum allowed power caps) & *Sum_Max* (sum of maximum allowed power caps) for the system
3) Compute the solution, using the various compute power cap algorithms (described subsequently)
4) Apply the solution to the set of controllable consumers within the list of components target

This algorithm has the potential to be applied to the whole data center recursively down to an individual compute node. All levels can be broken down into a component that represents the maximum power consumption of not controllable consumers and a component that represents all controllable components. Therefore, solutions for one level have the potential of being applied recursively to other levels of the power management hierarchy.

Equations 2, 3, 4 depict that fundamentally each power distribution is comprised of controllable consumers and not controllable consumers. Furthermore the equations describe that power control is nested across levels of granularity and controlability. The facility management can control the power of the installed systems, the system management can control the power of the compute nodes in the system, and the compute node management can control the compute unit power limit. The approach described in this paper is generic enough that it could be applied at any defined level in the power hierarchy. The prototype implementation targets system power and compute node power.

$$FacilityPower = \sum_{i=1}^{C} notControllableConsumers_i + \sum_{j=1}^{S} SystemPower_j) \tag{2}$$

$$SystemPower = \sum_{i=1}^{C} notControllableConsumers_i + \sum_{j=1}^{N} ComputeNodePower_j) \tag{3}$$

$$ComputeNodePower = \sum_{i=1}^{C} notControllableConsumers_i + \sum_{j=1}^{U} ComputeUnit_j) \tag{4}$$

The compute solution decision graph (Figure 2) to determine power distribution works as following:

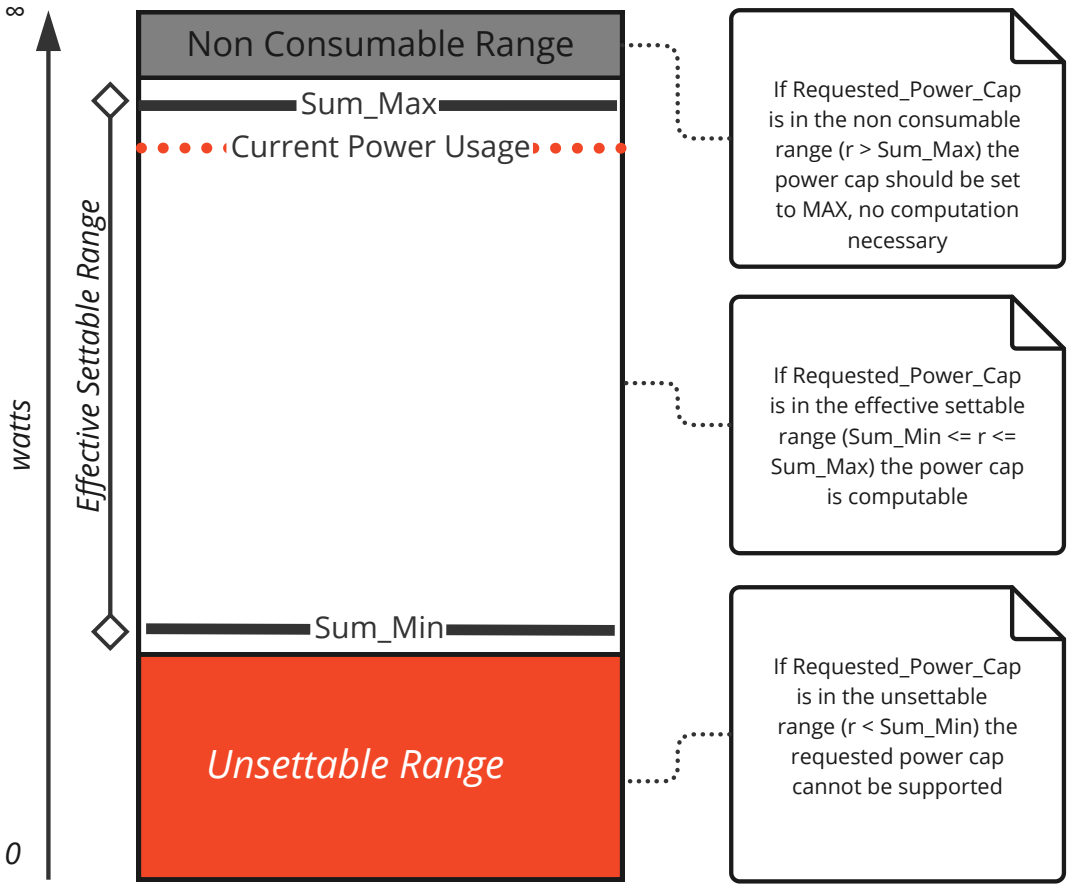

Fig. 2: Compute solution decision graph

1) IF *Requested_Power_Cap* is greater than *Sum_Max*: Set to max for all components
2) ELSE IF *Requested_Power_Cap* is less than *Sum_Min*: Return an error because minimum power for all components is not available.
3) ELSE *Requested_Power_Cap* is between *Sum_Max* and *Sum_Min*: For the set of components, compute a power cap value. Possible power distribution varies by algorithm and policy.

B. Distribution Algorithms

The prototype has an extensible system power distribution algorithm. The prototype has eight different algorithms implementations, some with multiple configurations, for a total count of twelve unique ways to distribute system power cap. The algorithms are modular so they could be modified or extended as necessary. Table III describes in detail the various algorithms developed for the prototype. The various algorithms have a similar base, they all start with the full list of the controllable consumers selected for power capping, and knowledge of the individual node *MaxW* & *MinW*. From this, the algorithms can calculate the *Sum_Max* & *Sum_Min* and determine if any base solution is possible (Figure 2).

After a power cap distribution has been calculated for the system and validated that it is within specification, the prototype calls the power control service provided as part of the system management layer. The power control service is the entity actually responsible for communicating with the OOB hardware controls (via Redfish). The prototype will confirm with the power control service that the power caps have been set on the individual nodes.

IV. Methodology and Results

The prototype implementation has multiple distribution algorithms that it can use to compute potential node power distributions. As part of developing the prototype, the multiple distribution algorithms were evaluated and tested.

The optimal distribution algorithm was determined by calculating different power distributions and selecting the algorithm that provided the best total solution utilization. For the prototype implementation total solution utilization was defined by algorithm solution utilization, which is defined as how close to *Requested_Power_Cap* the sum of the node power

Name	Description	Notes
base_solution	This algorithm mirrors the *compute solution decision graph*. It determines if any solution is possible (is in range between *Sum_Min* and *Sum_Max*).	This is the default algorithm that all other algorithms first call. If this algorithm identifies no valid solution, then no other algorithms could find a valid solution.
even_split	This algorithm take the difference between *Requested_Power_Cap* and *Sum_Min* and divides it evenly among all nodes.	This assumes all components are homogeneous, or that their power cap ranges overlap significantly.
equal_percentage	For each node type calculate the range (max – min) and split it up into 10,000 discrete steps. Then starting from Max for each node, decrease all nodes values by 1/10,000th until the sum of the power caps is less than or equal to *Requested_Power_Cap*. It is likely the value will be a decimal, which is then truncated to an integer, which is required for the hardware setting.	The hardware implementation only allows whole watt settings, in increments of one. 10,000 discrete steps, also referred to as the 'decrease quantum' was chosen to ensure a high enough resolution such that all discrete steps for all 'likely to be encountered' hardware types would be not larger than 1W. If the stepping was larger than 1W, e.g. decrease by 2W the application would most likely not be able to consume total available watts. This algorithm is the default algorithm selected for the prototype implementation (called *cray-power-capping*) of the presented solution.
count_down	For each node, decrease power cap value by 1W from Max until the sum of the power caps is less than or equal to *Requested_Power_Cap*.	This is similar to *equal_percentage*, but instead of all hardware types having an equal number of discrete steps the hardware has different numbers of available steps (equal to their delta), such that hardware with smaller ranges get to *MinW* before those with larger ranges. In our example the homogeneous node with a delta of 575W is exhausted, set to min, before the heterogeneous node with a delta of 1990W.
*delete_by_**	A collection of algorithms that group the nodes by power capping characteristics and then systematically set each group to minimum until an overall solution is found.	The nodes are all grouped by similar power cap controls (same Max, Min). Then the groups can be processed by one of several characterizations: count of components that have similar controls (e.g. same Min Max), delta for component (i.e. Max - Min), maximum power cap, minimum power cap. The characterized groups can be 'consumed', i.e. set to minimum, by either the largest or smallest group first.

TABLE III: Partial list of distribution algorithms developed for the proof of concept.

Algorithm	Utilization Mean	Utilization STDDEV	Utilization Variance
base_solution	6.35e-01	2.64e-01	6.97e-02
count_down	1.00e+00	2.04e-04	4.18e-08
delete_by_component_count_least-to-most	6.02e-01	2.17e-01	4.73e-02
delete_by_component_count_most-to-least	6.49e-01	1.78e-01	3.17e-02
delete_by_delta_largest-to-smallest	6.02e-01	2.17e-01	4.73e-02
delete_by_delta_smallest-to-largest	6.49e-01	1.78e-01	3.17e-02
delete_by_max_power_cap_largest-to-smallest	6.49e-01	1.78e-01	3.17e-02
delete_by_max_power_cap_smallest-to-largest	6.02e-01	2.17e-01	4.73e-02
delete_by_min_power_cap_largest-to-smallest	6.49e-01	1.78e-01	3.17e-02
delete_by_min_power_cap_smallest-to-largest	6.02e-01	2.17e-01	4.73e-02
equal_percentage	1.00e+00	2.71e-04	7.37e-08
even_split	9.99e-01	3.40e-04	1.16e-07

TABLE IV: Per algorithm solution utilization metrics across all tested percentages of system power limit (29% - 101%)

Algorithm	Node Type 1 (W)	Node Type 2 (W)
base_solution	350	764
count_down	350	1444
delete_by_component_count_least-to-most	925	764
delete_by_component_count_most-to-least	350	764
delete_by_delta_largest-to-smallest	350	764
delete_by_delta_smallest-to-largest	925	764
delete_by_max_power_cap_largest-to-smallest	350	764
delete_by_max_power_cap_smallest-to-largest	925	764
delete_by_min_power_cap_largest-to-smallest	350	764
delete_by_min_power_cap_smallest-to-largest	925	764
equal_percentage	517	1343
even_split	775	1189

TABLE V: Comparison of solution utilization algorithm at 4.24MW (50%) system power limit

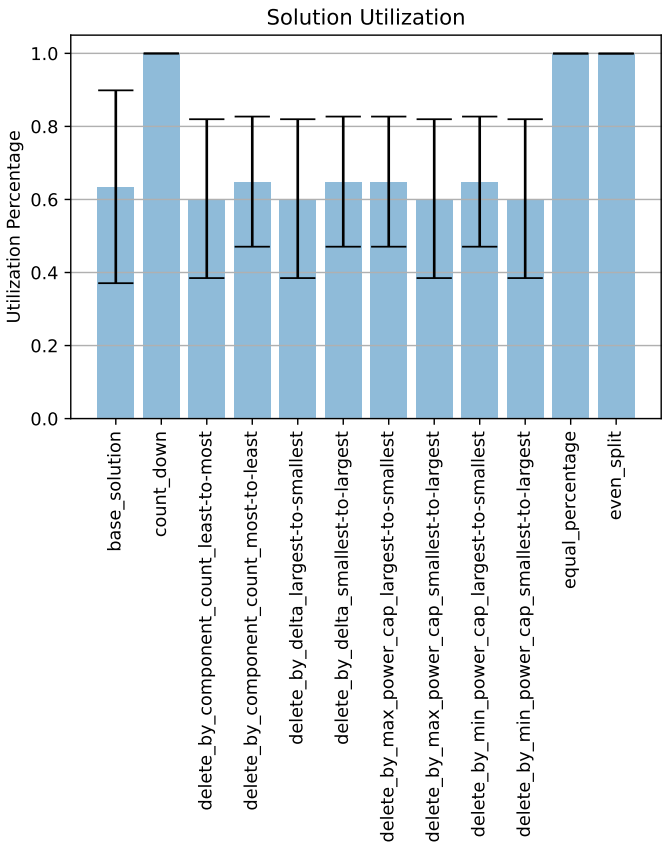

Fig. 3: Solution utilization

caps can be allocated without exceeding. The reasoning being that customers would like to take maximum advantage of the *Requested_Power_Cap*. In the future, total solution utilization could be defined according to the customer power management goal (for example, as a trade off between efficiency and performance), the customer distribution policy (for example, prioritizing high power consumers over low power consumers), and the nature of the hardware (for example, provide more power to better utilized node architectures).

For the prototype implementation, the algorithm solution utilization (ratio of how close the computed power cap is to the *Requested_Power_Cap*) used the node profiles enumerated in Table I. The test methodology was to generate a series of tests across the range of system power cap limits (*Sum_Min* to *Sum_Max*), from just below the minimum valid power cap solution (all nodes set to min, aka *Sum_Min*) to just above the maximum valid power cap solution (all nodes set to max, aka *Sum_Max*) split into whole percentages leading to 70 separate tests cases (70 unique possible *Requested_Power_Cap* values). Then each algorithm was executed using the 70 test cases and the per algorithm solution utilization (per test case) was determined. Finally for each algorithm the mean, standard deviation (STDDEV), and variance across the population of solution utilization for the test cases were calculated (see Figure 3 and Table IV for more details). Mean demonstrates how effective an algorithm was across all test cases, a score of 1.0 would mean that the algorithm allocated 100% of the *Requested_Power_Cap*. The STDDEV is the spread of the solution utilization from the mean. The variance measures the average degree to which each solution utilization varies from the mean. For both STDDEV and Variance, a lower score is better.

The conclusion of the prototype implementation was that *equal_percentage* or *count_down* are the algorithms that provide the most complete (efficient) algorithm solution utilization (Table IV). Even though both distribution algorithms provide almost the same utilization of the available power they have different application performance implications (Table V). The *equal_percentage* distribution provides a more even performance reduction across all node types since the power reduction is based on the same percentage from the maximum node power limit . Whereas the *count_down* method reduces the power by a fixed quantity independent of the node maximum power limit. *count_down* will exhaust Node Type 1 power cap before Node Type 2, setting it to its minimum performance earlier than in the *equal_percentage* distribution. Therefore *count_down* favors Node Type 2. Alternatively, *equal_percentage* will always ensure that all computes (Node Type 1 and Node Type 2) will have at least some power above minimum, unless the *Requested_Power_Cap* is below *Sum_Min*. Furthermore, while the prototype implementation optimized for maximum solution utilization, it is possible that sites may optimize for different criteria, such as providing maximum power to nodes with accelerators versus nodes with CPUs only (see Conclusion and Future Work for a more in-depth discussion on possible customer optimizations). Ultimately, it is likely ideal to optimize for some combination of performance and power trade-off between the different system nodes as well as the power distribution inside a node.

V. Future Work

The current prototype implementation, called *cray-power-capping*, is a standalone application that runs to completion and exits providing a static solution for system power capping. The future version, called *System Power Capping*, will be a RESTful API micro-service that, first, can be triggered by system events (such as job start), and second, will run continually to enable dynamic system power management.

System Power Capping introduces a new concept, called *pools*. A pool is a grouping of nodes that should share a power limit. Pools may be created by SysAdmins, but would most likely be created by workload managers to represent the set of nodes that are part of batch queues or jobs. Compute jobs are likely to have different relative priorities compared to each other on the system. Therefore it is likely that it is desirous to distribute power among different jobs differently, and to 'favor' some jobs more highly than others. By utilizing the pool concept the system can group nodes by shared purpose, imply or define a higher priority for some pools than others, and have much more fine grained control of the total system power limit and individual component limits.

System Power Capping will reduce the total number of distribution algorithms from eight to four, removing the *Delete_By_** algorithms. The overall solution utilization for the *Delete_By_** algorithms is significantly lower than the *count_down*, *equal_percentage*, *even_split* algorithms. The *Delete_By_** algorithms are close to the performance of *base_solution*, but are not as useful, because *base_solution* is called by every other algorithm to determine if any solution, regardless of performance, is possible. With the introduction of pools *System Power Capping* can compensate better for heterogeneous or disparate counts of hardware, removing the need for the *Delete_By_** algorithms.

System Power Capping introduces several other new concepts *rationing*, *upper and lower limits per pool*, and the notion of *starvation*.

Rationing is the concept that if there is insufficient power available, which is always the case when the *Requested_Power_Cap* is less than *Sum_Max*, the system actors may want to set a priority order for which pools (and nodes) are impacted first. This concept could be used to help ensure that a *high-priority* pool is the last pool to experience rationing, thereby retaining maximum available performance for the pool.

Upper and lower limits allow system actors more fine-grained control on the amount of available power to distribute to a pool. The upper and lower limits are within the max min window for the pool. An upper limit requires that even if more power is available, do not allow this pool to exceed the upper bound; this may be particularly useful if running hardware at TDP violates resource constraints. A lower limit requires that even if the pool could sacrifice more power, that it should not do so; this may be particularly useful if the system actor knows that a minimum power limit above *Sum_Min* is needed to get necessary job performance.

Starvation is the concept that if there is still insufficient power available, after components have been set to *MinW*, that the only remaining alternative is to power down nodes. This extreme action may be needed to keep the data center operational in a degraded power event. The concept of starvation is an extreme response, which could be needed, but it is ultimately up to the system actors to define if pools should be allowed to be starved.

cray-power-capping is a 'one shot' application; either the solution works and is applied, or is invalid. However *System Power Capping* is a continually processing solution with eventual consistency. As pools are created or deleted, as new hardware is added to the system, or as the overall site power limit is adjusted, *System Power Capping* will respond to total environment (the model) and take a best effort to reconcile the model of the system to the world. It is likely that at some points in time, the requested world state will not be possible, this may be due to user error, or because the system environment is too severely constrained, e.g. the data center needs to reduce power below *Sum_Min* because of a power failure, in this case *System Power Capping* will return a model validation failure, which indicates that despite best efforts the system cannot completely conform to the desired state of the world. In such cases the SysAdmins should take the necessary steps to either reduce load or increase capacity.

VI. Conclusion

The presented solution is the first step towards an optimal management of the system power consumption of heterogeneous HPC systems that use OOB management system. The solution enables hardware over-provisioning in order to make stranded and/or trapped power and/or cooling capabilities available for use to an HPC data center. In its first implementation it uses different power distribution algorithms to select the best static power distribution according to customer needs.

The next version, called *System Power Capping*, will be used to set compute node guard rails. Those guard rails can be considered starting node power set-points if an application aware in-band component is available. A combination of OOB control and in-band application awareness could be used to, for example, manage node power caps according to running application needs (e.g., application needs only two GPUs from the four on each job node – shift power from those nodes of the job to other nodes in the system).

Acknowledgment

The authors would like to thank Larry Kaplan and Andy Warner for reviewing this paper and providing feedback.

The authors would also like to thank Michael Jendrysik for helping with the first prototype implementation of the solution.

References

[1] Corbalan, Julita, and Luigi Brochard, "EAR: Energy management framework for supercomputers.", Barcelona Supercomputing Center (BSC) Working paper. 2019, https://www.bsc.es/sites/default/files/public/bscw2/content/software-app/technical-documentation/ear.pdf.

[2] Barcelona Supercomputing Center, "EAR: Energy management framework for HPC", https://www.bsc.es/research-and-development/software-and-apps/software-list/ear-energy-management-framework-hpc, as of October 2021.

[3] GEOPM Consortium, "GEOPM Service Documentation", https://geopm.github.io/, as of October 2021.

[4] Eastep, Jonathan, et al., "Global extensible open power manager: a vehicle for HPC community collaboration on co-designed energy management solutions.", https://dl.acm.org/doi/10.1007/978-3-319-58667-0_21 International Supercomputing Conference. Springer, Cham, 2017.

[5] Ferrero, Fabio; Reorda, Matteo Sonza, "Analysis and dynamic optimization of energy consumption on HPC applications based on real-time metrics.", https://webthesis.biblio.polito.it/6423/1/tesi.pdf, doctor thesis, 2017.

[6] ATOS, "Atos High Performance Computing Software Suites", https://atos.net/wp-content/uploads/2020/11/HPC-Software-Suite-position-paper.pdf, as of October 2021.

[7] Tang, Desai, Buettner, Lan. "Job scheduling with adjusted runtime estimates on production supercomputers". 2013, http://www.cs.iit.edu/~lan/publications/jpdc_2013_wei.pdf.

[8] Top500 List (Nov 2022), https://www.top500.org/lists/top500/2022/11/.

[9] Woong Shin, Vladyslav Oles, Ahmad Maroof Karimi, J. Austin Ellis, and Feiyi Wang, "Revealing power, energy and thermal dynamics of a 200PF pre-exascale supercomputer", In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21), Association for Computing Machinery, New York, NY, USA, Article 12, 1–14, https://doi.org/10.1145/3458817.3476188.