Powersched: A HPC System Power and Energy Management Framework

Marcel Marquardt
Hewlett Packard Enterprise
Böblingen, Germany
marcel.marquardt@hpe.com

Jan Mäder
Hewlett Packard Enterprise
Böblingen, Germany
jan.maeder@hpe.com

Tobias Schiffmann
Hewlett Packard Enterprise
Schwalbach a.T., Germany
tobias.schiffmann@hpe.com

Dr. Christian Simmendinger
Hewlett Packard Enterprise
Böblingen, Germany
christian.simmendinger@hpe.com

Dr. Torsten Wilde
Hewlett Packard Enterprise
Munich, Germany
wilde@hpe.com

***Abstract*—Modern processors and thus HPC systems consume huge amounts of power. Energy-efficiency is an important topic, not only due to rising energy prices. Some HPC sites are even restricted by the energy infrastructure and run systems which can at peak utilization consume more than is available. This overprovisioning is tolerable, as many applications do not cause the system to draw peak power. However, the current common strategy of using static power limits for all compute nodes is suboptimal, as applications are impacted differently by it. In this paper, we present the Powersched framework, which is part of HPEs vision towards a holistic system power management software stack. It continuously profiles applications using hardware performance counters, and runs an energy optimization algorithm to steer them into an energetic sweetspot using mean-shift clustering and in-band power limits. The state of the entire cluster is supervised to ensure that shared power budgets of overprovisioned systems are not exceeded. A test run using a selection of benchmark applications could show an average energy saving of around 14% with an average runtime increase of less than 2%.**

***Index Terms*—Power, Energy, Overprovisioned, HPC, ML**

I. Introduction

A. Motivation

While supercomputers, beyond theory and experiment, have become the third pillar of academic and industrial research, they can consume huge amounts of energy. For example, the fastest supercomputer Frontier (according to the Top500 list, Nov 2022) required more than 20 Megawatts of power during its HPL benchmark run. This rising power consumption already has led to a situation where large HPC sites want to reduce stranded or trapped facility power and cooling capacity to the deployment of hardware. Over-provisioned supercomputers, where the peak power demand of all compute nodes in the system can exceed the power available at a given site. In addition, energy prices have dramatically increased over the last year, especially in Europe. One strategy is to reduce power consumption by static power capping, but this leads to a substantial drop in overall application performance. However, not all applications are equal. Compute bound codes may suffer from lower CPU power limits and the resulting clock speed reduction, memory bound codes however can be limited without impacting performance significantly. This opens up an opportunity to significantly reduce energy-consumption without a drastic increase in runtimes, depending on the application profile. Over-provisioned supercomputers can stay within their power budget without sacrificing performance if power is shifted from application that don't need it to applications that benefit from additional power.

B. Problem Statement

Leveraging these potential power-savings is not trivial, especially for over-provisioned systems. As the applications are submitted by users, static analysis can not be used, thus requiring runtime profiling in the background. An optimal power limit then has to be determined based on the profile. On over-provisioned systems, the global power budget additionally cannot be exceeded, requiring system-wide coordination. Any solution also has to integrate with the scheduler to not exceed the power budget by starting too many jobs. One big challenge is to develop a framework that can support the widely different system configurations with different CPUs and even heterogeneous systems with GPUs.

C. Our Solution

The Powersched framework presented in this paper is part of HPE's vision towards a holistic system power management software stack. It implements a reliable and extensible framework for in-band application aware power and energy management. It can manage over-provisioned systems while simultaneously steering HPC workloads into their energetic sweet-spot. Powersched records CPU profiling data while changing system runtime parameters, such as the available power per CPU package. Using Machine Learning, it derives an optimal sweet-spot for the given workload and its profiling counter footprint.

D. Structure of this paper

This paper is split in four parts: in section II, related existing projects are discussed which fall in the scope of

the given problem. Section III then covers the architecture of the Powersched framework, while section IV goes into detail on the current energy-optimization algorithm based on mean-shift clustering. Our early results using this strategy are then presented in section V.

II. Background and Related Work

The Energy Aware Runtime (EAR) library [1] can optimize the energy efficiency of an application by using node-local energy optimization techniques. EAR works mainly for MPI applications. It intercepts MPI calls and detects the outer loop. EAR then generates an application signature (using iteration time, average DC node power, main memory transactions per instructions, and cycles per instruction) and uses this signature together with a system signature to project time and power according to a power model. The best fitting projection is applied to fulfill a specified energy policy. This is done independently on each node of a job. EAR does not manage any power constraints. Our approach uses deterministic measurements to determine the optimum setting for each cluster, and applies the optimal setting automatically by matching an application to a cluster.

The Bull Dynamic Power Optimizer (DPO) [2] uses a hardcoded frequency to set either a low or high processor frequency based on an Instruction per Cycle (IPC) threshold. For example, if IPC is below 12.5, a frequency of 1.5GHz is applied. If IPC is above 12.5, a high frequency of 2.0GHz is applied. DPO optimization is node-local, and DPO does not manage any power constraints.

The Global Extendible Open Power Manager (GEOPM) [3] works by intercepting MPI or OpenMP calls. GEOPM identifies the out loops and then uses a full frequency sweep to identify the optimal frequency or power limit that fulfills the requested optimization policy. This frequency or power limit is then applied till the next loop is identified. GEOPM can manage a job power cap, but does not manage system-wide power budget or a job which is running across multiple power domains. The full frequency sweep for every loop leads to high resource requirements for GEOPM.

III. Powersched Framework

A. Requirements

The problem statement leads to the following requirements, which are divided into the areas of heterogeneous systems, overprovisioned systems and energy optimization.

The requirement to support overprovisioned systems dictates that the framework must work at a holistic level, as a simple node locale solution would not be sufficient. To work on a holistic level the framework needs the ability to interact with the job scheduler to interact with the cluster and take control of the resource allocation. Additionally the framework must be able to move power between nodes, to shift power to workloads that can utilize it most efficiently. To enable the users to some level of control over the management mechanisms configurable power distribution policies is required.

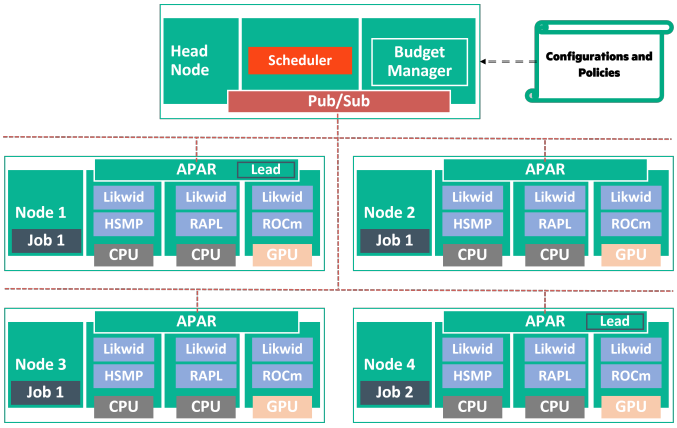

Fig. 1. Architecture diagram

To support modern computing clusters and make the framework adaptable to future system architectures, the framework must support heterogeneous systems. With the proliferation of accelerators such as GPUs and the imminent development of APUs, support for multiple components in one node is required.

The third key requirement is an energy optimization mechanism, that works without additionally user input our static code analysis and instead relays only on system metrics and hardware counters. Utilizing these counters workload profiles and application phase changes should be dynamically identified. Based on these identified workload profiles the optimizer should assign a power cap representing the most energy efficient run time configuration to the compute nodes. To work within a given over all cluster energy budget the optimizer should furthermore be able to shift power between jobs to make the best use of the limited power.

B. Architecture

Powersched addresses these requirements with the architecture shown in figure 1. It consists of three main components: a Budget Manager daemon, one APAR daemons per node and a scheduler integration. These communicate through a Publish/Subscribe messaging system.

The **Budget Manager daemon** is the central authoritative entity. It keeps track of the current state of the system, including which jobs are running on which nodes, what power cap is set on them and how much of the power budget is remaining. Using this holistic view on the system, it decides the power level for all existing jobs and tells the scheduler how many nodes it can use for scheduling new jobs. When it receives measurements from the APAR daemons, it uses the energy optimization algorithm to determine the best power level. The budget manager then attempts to satisfy this request with the available budget. Any left-over budget is used to facilitate new jobs.

Which **energy optimization algorithm** is actually used is not fixed, however. Powersched defines an abstract interface, so different implementations can be provided and chosen by

the user. In principle, these implementations are a simple black box with the current measurements as input and the new power caps as outputs. Currently, a clustering implementation is provided, which is discussed in section IV in more detail.

On each compute node runs an instance of the **APAR node-local daemon**. It serves as an interface to the system hardware and executes decisions made by the Budget Manager, i.e., setting the given power caps. One APAR daemon per jobs additionally takes on the function of the Job daemon, which is called **APAR lead** in figure 1. In addition to setting power caps, it, also has the task of measuring hardware performance counters. As it is assumed that a job produces similar load across its nodes, Powersched operates currently on a per-job basis, meaning that all nodes of a job have the same power cap. This may change in the future.

The **Scheduler integration** ensures that the Job scheduler starts jobs only on nodes with enough power allocated to them by the Budget Manager. Both Slurm and PBS are currently supported by using dynamic resources (or GRES) that indicate if the node has enough power. Hooks and scripts keep the resource values synchronized with the Budget Manager and also notify Powersched of new and finished jobs.

Publish-Subscribe Messaging is used for communication between the different parts of the framework. This allows sending node, job and system-wide messages easily. As it leverages existing message brokers, Powersched benefits from their robustness and features.

Its **Configuration** is probably the most important part of the framework. As many things within the framework are defined through interfaces and have several possible implementations, the configuration is very powerful and allows tailoring to specific systems. It includes details about the nodes within the cluster, which of them share a given power limit, what hardware components they have, but also system level options like the kind of energy-optimization algorithm to use. To ensure a consistent configuration for all components and prevent having to change node images for each change, only the budget manager reads the configuration files and distributes it via the messaging layer whenever requested.

C. Components

In order to address heterogeneous systems with different hardware components like multiple CPUs and GPUs, another abstraction was introduced. As there are too many components and configurations to support individually, Powersched provides only a set of connectors to system/vendor APIs. Users then define their own components through the configuration by specifying which connectors to use. This is very flexible, as connectors can be reused and adding more is also easy. Each component consists of:

- a *measure interface* implementation
- a *power interface* implementation
- power draw information
- information for energy optimization

The **measure interface** is responsible for obtaining the metrics used for energy optimization. Currently, only LIKWID is supported, but connectors for Perf or PAPI could be easily added. A connector defines a start, stop and poll function where the latter is called by Powersched periodically to get new measurements. This allows connectors for both push and poll based APIs. Which metrics can be measured depends on the implementation and has to be tuned to the optimization algorithm. For the LIKWID connector, the event groups are given through the configuration.

A **power interface** connector basically provides a single method for setting power caps. Currently, an AMD HSMP implementation is supported, but a connector for Intel CPUs could be easily added using the RAPL API, as well as other higher-level APIs.

In addition to these interfaces, some **metadata** is also part of the component configuration. This includes the minimum and maximum values supported for power caps, as well as information necessary for the energy optimizer. The latter can contain for example an expression that calculates the number of retired instructions from measurements. This depends however on the used implementations.

As some values may differ between instances of the same component type, Powersched allows defining **component parameters** which can be used as a placeholder within the component configuration. The actual values are given in the node configuration when specifying its components. A prominent example is the CPU socket index, which is required by the power interface. With parameters, a dual-socket node can use the same component type.

D. Deployment

Apart from the daemons, Powersched consists of measure, power and energy optimization implementations. As the framework evolves, their number will grow, which poses the issue that only a few are actually used. Distributing all of them as well as their dependencies as one unit is not efficient. Instead, Powersched separates them in separate packages that can be installed separately as needed.

IV. Energy Optimization

An integral part of the framework is to move the power budget intelligently between applications. This is achieved by an energy optimization algorithm that decides the best power level for a given application phase based on measurements. For this purpose, we developed a machine-learning-based model that can dynamically predict an optimal power cap for corresponding job nodes based on hardware performance counters. These counters are special registers of a processor in which hardware-related activities are tracked.

A. Algorithm

The Powersched framework currently utilizes 37 of these events to classify application behavior. Since the number of performance counters is limited, the events are divided into four blocks and are each measured over a duration of 2 minutes so that a measurement interval lasts 8 minutes. This duration also ensures a steady solver state.

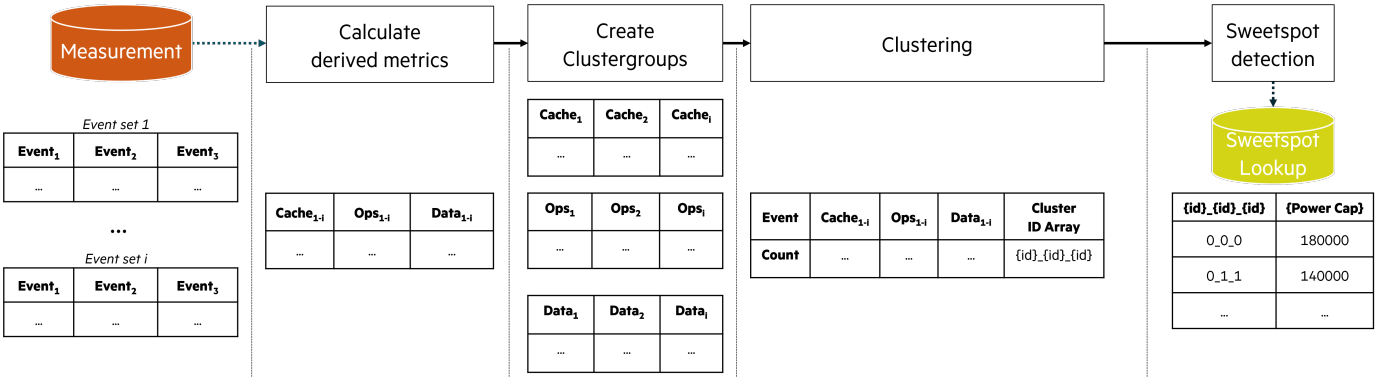

Fig. 2. Graphical representation of the clustering optimization workflow.

TABLE I
COUNTERS AND EVENT SETS USED FOR AMD ZEN2 ARCHITECTURE

	Eventset_1	**Eventset_2**
FIXC1	ACTUAL_CPU_CLOCK	ACTUAL_CPU_CLOCK
FIXC2	MAX_CPU_CLOCK	MAX_CPU_CLOCK
PMC0	RETIRED_SSE_AVX_FLOPS_ALL	DATA_CACHE_ACCESSES
PMC1	CPU_CLOCKS_UNHALTED	L1_DTLB_MISS_ANY_L2_HIT
PMC2	RETIRED_INSTRUCTIONS	CPU_CLOCKS_UNHALTED
PMC3	RETIRED_BRANCH_INSTR	ICACHE_FETCHES
PMC4	RETIRED_MISP_BRANCH_INSTR	RETIRED_INSTRUCTIONS
PMC5	MERGE	DATA_CACHE_REFILLS_ALL
DFC0	DATA_FROM_LOCAL_DRAM_CHANNEL	DATA_OUT_TO_REMOTE_2
DFC1	DATA_TO_LOCAL_DRAM_CHANNEL	DATA_OUT_TO_REMOTE_3
DFC2	DATA_OUT_TO_REMOTE_0	DATA_OUT_TO_REMOTE_4
DFC3	DATA_OUT_TO_REMOTE_1	DATA_OUT_TO_REMOTE_5

	Eventset_3	**Eventset_4**
FIXC1	ACTUAL_CPU_CLOCK	ACTUAL_CPU_CLOCK
FIXC2	MAX_CPU_CLOCK	MAX_CPU_CLOCK
PMC0	ICACHE_L2_REFILLS	LS_DISPATCH_LOADS
PMC1	ICACHE_SYSTEM_REFILLS	DATA_CACHE_REFILLS_LOCAL_ALL
PMC2	REQUESTS_TO_L2_GRP1_ALL_NO_PF	DATA_CACHE_REFILLS_REMOTE_ALL
PMC3	L1_DTLB_MISS_ANY_L2_MISS	CPU_CLOCKS_UNHALTED
PMC4	HWPREF_DATA_CACHE_FILLS_LOCAL_ALL	RETIRED_INSTRUCTIONS
PMC5	HWPREF_DATA_CACHE_FILLS_REMOTE_ALL	LS_DISPATCH_STORES
CPMC0	L3_ACCESS	
CPMC1	L3_MISS	
DFC0	DATA_FROM_LOCAL_DRAM_CHANNEL	DATA_OUT_TO_REMOTE_2
DFC1	DATA_TO_LOCAL_DRAM_CHANNEL	DATA_OUT_TO_REMOTE_3
DFC2	DATA_OUT_TO_REMOTE_0	DATA_OUT_TO_REMOTE_4
DFC3	DATA_OUT_TO_REMOTE_1	DATA_OUT_TO_REMOTE_5

These events are then used as input for a clustering model. In order to evaluate how well different clustering methods can be applied to our data set and which method gives better results, it was decided to implement a partitioning clustering method algorithm and a density-based method algorithm to compare the results. The mean-shift algorithm was chosen as a representative of the partition clustering method because it does not require a predefined number of clusters compared to other algorithms of the partition method. The DBSCAN algorithm was chosen as a representative of the density-based method because this algorithm can detect outliers, the so-called noise, and thus possibly achieve a better result than the mean-shift algorithm, where each point is assigned to a cluster. Clustering is considered good if it has high separation between clusters and high cohesion within a cluster. The silhouette score [4], [5] was used for evaluation. The silhouette coefficient is defined for the interval [-1,1]. Positive values indicate strong separation between clusters, while negative values indicate mixing clusters or overlapping clusters. If the silhouette coefficient is zero, it means that the data is evenly distributed in the Euclidean space. The result of the work was that the DBSCAN achieved a silhouette coefficient of 0.62, whereas the mean-shift achieved a silhouette coefficient of 0.74. The distribution of the applications among the clusters formed was also evaluated. In this regard, the mean-shift algorithm was also clearly ahead of the DBSCAN algorithm. The mean-shift algorithm grouped each application into a single cluster at a time, while the DBSCAN algorithm had applications distributed across multiple clusters. In addition, outlier detection proved to be more of a disadvantage than

an advantage, as the DBSCAN algorithm classified entire applications as noise in extreme cases.

After the mean-shift turned out to be significantly better for application detection, the silhouette score could be further improved through some adjustments. Instead of clustering on the raw event values, we introduced derived metrics, which are calculated from the raw measurements. Additionally, to distinguish more nuanced differences in the individual areas in the clustering, three cluster groups are introduced which are clustered using separate models: *operations*, *caches*, and *data*. Operations include events related to branching and retired instructions. Caches include events related to the translation lookaside buffer (TLB), instruction (ICache), data (DCache), L2, and L3 caches. The last group is *data*, which includes events related to data movements as well as Numa information. Each cluster group yields a cluster ID for each measurement: an operation ID, a caches ID, and a data ID. These three IDs then form a cluster-ID array or fingerprint. Using these two improvements, the algorithm now reaches a silhouette score of 0.85 – an improvement of 0.23.

The third and final step is a lookup table that maps each fingerprint to a sweet-spot, which is the power level on with a workload runs most efficiently in terms of instructions per watt.

B. Initialization

The proposed algorithm requires training before it can be used for inference. The first step of this initialization is data collection. In this step we run a selection of benchmarks to generate different kinds of workload (e.g. compute bound, memory bound or mixed). To determine the most efficient power level for each workload, each benchmark is run several times with different power limits. While running, we periodically take measurements of the 37 events.

After all benchmark runs are completed, the data is prepared for the following steps by calculating the derived metrics and grouping them into the three cluster groups. These form the inputs of the mean-shift models. We also calculate the sweet-spot of each measured application phase here, as it is required later to generate the lookup table. This process starts with calculating the instructions per watt for each measurement. We then group all measurements of the same application phases together, i.e., all first measurements of application XY. Each group should contain measurements for every power level, and the one with the most instructions per watt is selected as the sweet-spot for this application phase.

For each power level, we train the mean-shift clustering models for the three cluster groups with the prepared data measured on that level. Afterward, we generate the sweet-spot lookup table. To generate this table, we collect all measurements with the same fingerprint and power level. For each measurement within such a group, we look up the sweet-spot that was previously determined for the given application phase. The power level that is found most often as a sweet-spot among the measurements is saved as the sweet-spot of that fingerprint.

C. Inference

During inference, the same measurements are performed as in the training phase, as well as calculating the derived metrics and grouping them into the three cluster groups. The trained mean-shift models for the currently applied power level is used to obtain a fingerprint. Afterward, we check if a corresponding sweet-spot is present in our lookup table. If yes, this power level is returned as the sweet-spot of the current application phase. If no sweet-spot is defined for the given fingerprint, no decision can be made and the current power level is retained.

V. Evaluation

In this section, we evaluate the energy optimization algorithm discussed in section IV. As a second step, we also discuss the architectural bottlenecks of the solution.

A. Performance

Before proceeding, it is important to note that the results come from an early prototype and the current implementation may behave differently. Since our solution aims to increase the energy efficiency by optimizing the instructions per watt, Kilowatt-hour (kWh) is used as the evaluation criteria. To evaluate our tool, five applications are used initially:

- the Conjugate Gradient (CG) and Integer Sort (IS) benchmarks from the NAS Parallel Benchmarks [6]
- the MPI versions of Halo3d and Halo3d-26 which are part of the Ember ECP proxy application [7]
- the Swfft ECP proxy application [8]

The NAS Parallel Benchmarks were used to train the models, while the other benchmarks were additionally used for evaluation to investigate how well it works with applications that were not included in the training set. Each application was run across two nodes. As a test platform, a small cluster with an Infiniband fabric is used. Each node is based on the HPE ProLiant DL385 Gen10 and has two 64-core AMD Epyc 7702 CPUs with hyper-threading disabled. To exclude possible hardware influences, each application was executed equally on all available nodes. To smooth out differences between runs, each application was started four times with our dynamic power capping and four times without.

The power consumption with and without dynamic power capping are shown in table II. For better comparability, the mean value of the four measurements per application is shown. In the column ”energy savings”, the percentage reduction in kWh was calculated. This column clearly shows that it was possible to increase the energy efficiency of all five applications. Thus, an energy saving of 14.382% can be achieved on average. Another aspect is the change in runtime, which is shown in the ”runtime extension” column. Here, an increase in the runtime of 1.65% on average can be observed, which is a relatively small value compared to the energy savings.

B. Bottlenecks

Although the performance of the model looks very promising, the clustering approach also has its drawbacks. For one, a large training dataset with diverse application profiles

TABLE II
COMPARISON OF ENERGY CONSUMPTION PER BENCHMARK IN KWH WITH AND WITHOUT DYNAMIC POWER CAPPING (DPC)

	with DPC		without DPC			
Benchmark	**kWh**	**runtime in Sec.**	**kWh**	**runtime in Sec.**	**energy savings in %**	**runtime extension in %**
CG	0,8149	4043	0,8875	3982	8,18	1,53
IS	0,5415	2943	0,6392	2906	15,28	1,27
Halo3d	0,7558	4189	0,9269	4177	18,46	0,28
Halo3d-26	0,8785	4831	1,0504	4733	16,36	2,07
Swfft	0,4193	2267	0,4855	2199	13,63	3,09

is required for the mean-shift to accurately detect clusters. Additionally, measurements on all power levels are required to generate the sweet-spot lookup table. This results in large training times and the difficult choice of which benchmarks to train on. The second drawback is that the measurements which are obtained during operational use (inference mode) cannot be used to improve the models. As the application is run only once there, it is not possible to determine the accuracy of the prediction which is required to adjust the sweet-spot lookup table. A third potential bottleneck could be the reliance on a steady solver state, i.e., that the application contains a loop and each measurement covers all parts of at least one of its iterations. This is the case for most HPC applications, but it limits the frequency measurement can be taken. But this limitation also comes with its benefits, as it switches power limits less often, thus reducing overhead and making it possible for the budget manager to enforce the global power budget.

VI. CONCLUSION

A. Summary

In this paper, we presented a framework that can dynamically optimize applications running on an HPC system at runtime. It is capable of enforcing shared power budgets and is designed to be extensible to support all kinds of cluster configurations, including heterogeneous systems. Each node runs a local daemon, called APAR, that measures system metrics and sets in-band power limits. A system daemon, called Budget Manager, maintains a holistic view of the system, integrates with the job scheduler, and communicates with the APAR daemons to ensure no power budget is exceeded. Additionally, the Budget Manager runs an energy optimization algorithm that determines the optimal power limit for each job without significantly impacting the application performance. This allows saving power, especially for memory-bound jobs, which can be used to start more jobs than otherwise possible on an over-provisioned system.

Our current algorithm for energy optimization is based on mean-shift clustering and uses hardware performance counters collected by LIKWID to determine the optimal power limit of an application phase. It calculates derived metrics, groups them into *operations*, *caches* and *data* related events, which are then clustered separately using a mean-shift model for the given group and power level. The resulting fingerprint is then used to look up the optimal power limit. Using this, we could show an average energy saving of around 14% with an average runtime increase of less than 2%.

However, Powersched is not fixed to LIKWID and this optimization algorithm. It defines plugin interfaces for measuring, power limit setting and energy optimization, so that they can be easily swapped out or extended.

B. Future Work

While the Powersched framework is already very capable, future improvements are planned. For one, alternative algorithms for energy optimization will be investigated, like reinforcement learning or analytical approaches. These may reduce the initial learning overhead and could allow improving during production use.

Another area of interest is the reduction of input events and exploration of other tunable settings of the hardware, like changing frequencies. Especially, out-of-band metrics could be beneficial, as they are less intrusive, but also less specific.

The last and probably most important future research topic is optimizing heterogeneous systems with GPUs. Currently, they are considered in the architecture, but hardware interfaces are lacking, and the energy optimization algorithm also handles different components separately, which could pose problems.

C. Outlook

The Powersched framework is able to manage over-provisioned systems while at the same time steering HPC workloads into their energetic sweet-spot by dynamically changing power limits based on their hardware performance counter footprint. Using this, we could show an average energy saving of around 14% with an average runtime increase of less than 2%.

REFERENCES

[1] J. Corbalan and L. Brochard, "EAR: Energy management framework for supercomputers," en, *Barcelona Supercomputing Center (BSC) Working paper*, 2019.

[2] T. d. L. Magistrale, "Analysis and dynamic optimization of energy consumption on HPC applications based on real-time metrics," en, 2017.

[3] J. Eastep, S. Sylvester, C. Cantalupo, *et al.*, "Global Extensible Open Power Manager: A Vehicle for HPC Community Collaboration on Co-Designed Energy Management Solutions," en, in *High Performance Computing*, J. M. Kunkel, R. Yokota, P. Balaji, and D. Keyes, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2017, pp. 394–412, ISBN: 978-3-319-58667-0. DOI: 10.1007/978-3-319-58667-0_21.

[4] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," en, *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, Nov. 1987, ISSN: 0377-0427. DOI: 10.1016/0377-0427(87)90125-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377042787901257 (visited on 03/30/2023).

[5] J.-O. Palacio-Niño and F. Berzal, *Evaluation Metrics for Unsupervised Learning Algorithms*, arXiv:1905.05667 [cs, stat], May 2019. DOI: 10.48550/arXiv.1905.05667. [Online]. Available: http://arxiv.org/abs/1905.05667 (visited on 03/30/2023).

[6] D. Bailey, E. Barszcz, J. Barton, *et al.*, "The Nas Parallel Benchmarks," *The International Journal of Supercomputing Applications*, vol. 5, no. 3, pp. 63–73, Sep. 1991, Publisher: SAGE Publications, ISSN: 0890-2720. DOI: 10.1177/109434209100500306. [Online]. Available: https://doi.org/10.1177/109434209100500306 (visited on 03/30/2023).

[7] Sandia National Laboratories, *Ember*, en-US, May 2018. [Online]. Available: https://proxyapps.exascaleproject.org/app/ember-communication-patterns/ (visited on 03/30/2023).

[8] Argonne National Laboratory, *SWFFT*, en-US, Apr. 2018. [Online]. Available: https://proxyapps.exascaleproject.org/app/swfft/ (visited on 03/30/2023).