How to Make Profit: Exploiting Fluctuating Electricity Prices with Albatross, A Runtime System for Heterogeneous HPC Clusters

Timo Hönig, Christopher Eibel, Adam Wagenhäuser, Maximilian Wagner, and Wolfgang Schröder-Preikschat
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

ABSTRACT

The ongoing evolution of the power grid towards a highly dynamic supply system poses challenges as renewables induce new grid characteristics. The volatility of electricity sources leads to a fluctuating electricity price, which even becomes negative when excess supply occurs. Operators of high-performance–computing (HPC) clusters therefore can consider the highly dynamic variations of electricity prices to provide an energy-efficient and economic operation.

This paper presents Albatross, a runtime system for heterogeneous HPC clusters. To ensure an energy-efficient and economic processing of HPC workloads, our system exploits heterogeneity at the hardware level and considers dynamic electricity prices. We have implemented Albatross and evaluate it on a heterogeneous HPC cluster in our lab to show how the power demand of the cluster decreases when electricity prices are high (i.e., excess demand at the grid). When electricity prices are low or negative (i.e., excess supply to the grid), Albatross purposefully increases the workload and, thus, power demand of the HPC cluster—to make profit.

CCS CONCEPTS

• **Software and its engineering** → **Operating systems**; • **Computer systems organization** → **Distributed architectures**; **Heterogeneous (hybrid) systems**; *System on a chip*; *Embedded systems*; • **Hardware** → **Power and energy**; • **Computing methodologies** → *Graphics processors*;

KEYWORDS

runtime and operating systems, power and price awareness, electricity prices, high-performance computing, heterogeneity, energy demand, power management, power capping, embedded systems

ACM Reference Format:
Timo Hönig, Christopher Eibel, Adam Wagenhäuser, Maximilian Wagner, and Wolfgang Schröder-Preikschat. 2018. How to Make Profit: Exploiting Fluctuating Electricity Prices with Albatross, A Runtime System for Heterogeneous HPC Clusters. In *ROSS'18: 8th International Workshop on Runtime and Operating Systems for Supercomputers, June 12, 2018, Tempe, AZ, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3217189.3217193

ROSS'18, June 12, 2018, Tempe, AZ, USA

ACM ISBN 978-1-4503-5864-4/18/06…$15.00
https://doi.org/10.1145/3217189.3217193

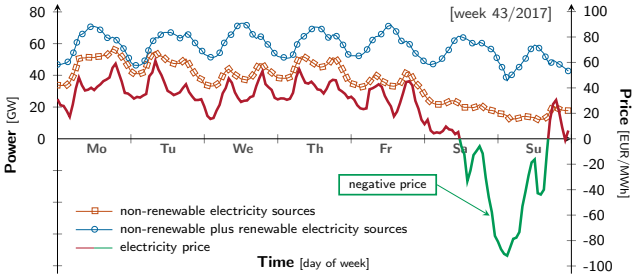


Figure 1: Electricity prices fluctuate strongly and depend on the availability of renewables. The prices even become negative when excess supply meets under demand [4].

1 INTRODUCTION

The evolution of the power grid towards a highly dynamic supply and demand system poses challenges [1, 2] to its operators and subscribers. The dependence on renewable electricity (e.g., wind, solar, and water) induces new grid characteristics: the volatility of electricity sources leads to an interplay of excess supply and demand, which results in fluctuating electricity prices. Thus, the operation of high-performance–computing (HPC) clusters can work with the fluctuating electricity price to ensure cost effectiveness [3].

Operators of HPC clusters and supercomputer systems can encounter highly dynamic electricity prices (cf. Figure 1) due to the increasing amount of renewable electricity that enters the power grid [5] and the yet missing adaptation of new power-storage technologies (e.g., power-to-gas [6], power-to-liquid [7]). On the one hand, electricity prices rise when excess demand of the grid occurs or an under supply is detected. On the other hand, the prices are dropping when an under demand or excess supply occurs. Based on these constraints, electricity prices are calculated and, therefore, subject to high fluctuation. In extreme cases, prices even become negative [8]—grid subscribers are getting paid for consuming power.

A common cause of negative electricity prices is an excess of renewable electricity being supplied to the grid. For example, electricity prices dropped to -91.87 EUR (-113.34 USD) per megawatt hour in Germany during week 43 of 2017 (cf. Figure 1). Currently, negative electricity prices regularly occur in countries with strong commitment to renewables [8] and, as a future perspective, 80 % of the energy demand in the US is feasible to be met by renewables [9].

To large electricity customers (i.e., operators of supercomputers and HPC clusters), negative electricity prices are both a challenge and an opportunity. From an economic standpoint, energy-efficient operations can be unwanted during times when electricity prices are

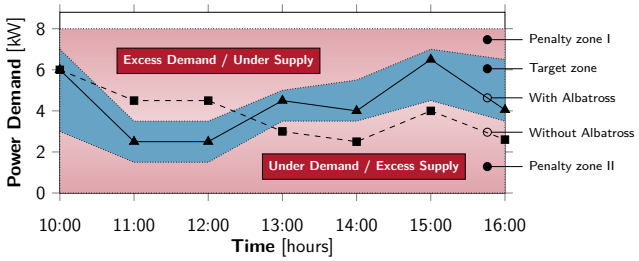


Figure 2: Power grids charge financial penalties according to the dynamic demand and supply during operation.

negative. Thus, next generation supercomputers and HPC systems need to consider operation modes that lower the power demand when electricity prices are high, and operation modes that increase the power demand when electricity prices become low or negative. To achieve this, workload shifting (i.e., by means of scheduling [10]), however, is not the only option to control the power demand of the overall system. Instead, the increasing number of heterogeneous processors [11] of supercomputer systems and HPC clusters opens new ways for lowering and (purposefully) increasing the power demand of the overall system dynamically at runtime.

Existing algorithms and HPC applications must be ported [12] when new hardware architectures are introduced to improve performance aspects. Over the last years, however, energy efficiency of supercomputers and HPC clusters has emerged as an additional first-class design criterion [13]. Thus, porting efforts now also focus on improving the energy efficiency of HPC applications when new hardware architectures are introduced. The porting of HPC applications to an increasing number of platforms motivates the use of heterogeneous HPC clusters [14] that consist of individual systems with distinct performance and energy-efficiency properties.

This paper presents Albatross[1], a runtime system for heterogeneous HPC clusters. Our proposed system dynamically adjusts the power demand of the HPC cluster in order to adapt to the current state of the grid. For this purpose, Albatross considers heterogeneity at different levels, for example, at architecture level (i.e., x86-64, ARM) and component level (i.e., CPU, GPU), to match with requirements of individual workloads and in coordination with the current grid state (i.e., electricity prices). We have implemented Albatross and evaluate it on an HPC cluster in our lab. The evaluation shows how the power demand of the cluster decreases when electricity prices are high (i.e., excess demand at the grid). When electricity prices are low or negative (i.e., excess supply to the grid) Albatross intentionally increases the workload, which results in a higher power demand of the HPC cluster—to make profit.

The contribution of this paper is threefold. First, we present the system design of Albatross, a power- and price-aware runtime system for HPC clusters. Our system exploits different scopes of heterogeneity to decrease the power demand of the overall system when electricity prices are high, and increase the power demand of the system once electricity prices are low or even negative. Second, we discuss insights from implementing Albatross for an HPC cluster in our lab and share evaluation results for the system with various workloads. Third, we compare Albatross with a commonly used HPC workload manager that provides a similar feature set.

The paper is structured as follows. Section 2 motivates the need for power-aware HPC runtime systems that go beyond the current state of the art, and discusses current challenges in the area. The system design of Albatross addresses outstanding challenges and is discussed in Section 3 together with an implementation of our system. We evaluate the implementation in Section 4 with various HPC workloads and compare it to another HPC workload manager. Section 5 discusses practicability, adaptability, and combinability aspects of Albatross, Section 6 summarizes related work, and Section 7 concludes the paper.

2 MOTIVATION AND PROBLEM STATEMENT

Supercomputing systems and HPC clusters are currently undergoing fundamental structural changes. On the one hand, the structural changes are affecting the heterogeneity of systems and, on the other hand, renewables are currently revolutionizing the electrical grid.

Internally, today's HPC systems and supercomputers strongly rely on heterogeneous compute nodes [14] that are part of the individual system architectures. For example, different processor architectures (i.e., CISC, RISC, and application-specific integrated circuits) ensure that workloads, which are heterogeneous at the software level, can be processed in an efficient manner. Central processing units are additionally supplemented with high-performance co-processors (i.e., GPUs) that further increase the heterogeneity at the hardware level. With their unmatched power demand, it is important for HPC and supercomputer systems to adapt requirements of different workloads to the heterogeneous hardware components. As such, today's systems commonly optimize for performance (i.e., increase loop throughput [15]) under consideration of power constraints [16, 17], which eventually encourages sustainability aspects—also in supercomputing environments [18].

At the same time, the electrical grid that powers supercomputing systems and high-performance–computing clusters is changing dramatically, especially in countries that spend large efforts on exploiting renewables to the greatest extent possible [19]. With the integration of renewable energies, the amount of available electricity is subject to strong fluctuations and depends on uncontrollable factors (i.e., weather conditions). The structural changes of the electrical grid therefore have a direct influence on the operation of large-scale computing systems, in particular when the economic operation of such systems is considered.

To adapt to the changes of the power grid, it is no longer sufficient to build systems that provide a maximum amount of processing power while keeping power demand low [20]. Today, however, it is necessary to operate HPC clusters and supercomputer systems with a high degree of flexibility that ensures that electricity—which is generated by renewables—is used whenever it is available. As soon as the supply of renewables is high, grid operators must ensure the stability of the grid by increasing the subscribers' energy demand. To do so, grid operators dynamically lower electricity prices, which motivates subscribers to increase their demand of electricity. Prices even become negative [4, 8] when excess supply of renewables hits low demand of electricity.

[1] Albatrosses are considered as masters of efficient flight, and feed themselves from a variety of different, heterogeneous foods, such as squid, small fish, and crustaceans.

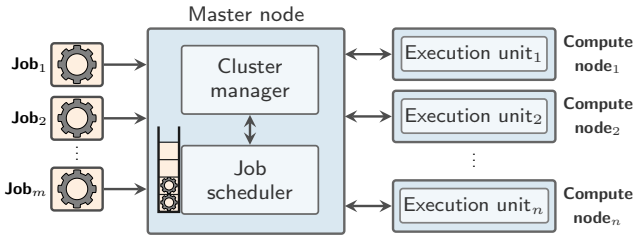


Figure 3: Overview of the targeted HPC systems, consisting of one master node and multiple compute nodes.

Therefore, it is necessary to redesign HPC clusters and supercomputer systems to adapt to the new external constraints in order to operate the systems economically. To achieve this, it is common to work with upper and lower bounds on the necessary power demand that must be met by a grid subscriber [21]. Figure 2 visualizes the target zone of power demand by a subscriber (blue area) that is dynamically adjusted over time. The target zone is limited by two penalty zones (red areas): the first penalty zone is activated when excess demand is detected at the grid. Thus, subscribers are motivated to reduce their power demand to increase the grid stability. When the electrical grid experiences excess supply of renewables, subscribers need to raise the power demand to escape the second penalty zone and implicitly protect the grid stability. This mechanism is equivalent to a minimum purchase quantity [21].

Adapting HPC clusters and supercomputers to the new grid characteristics requires a re-engineering of the systems' runtime environment: the operating system and the workload managers. With Albatross, we propose a runtime system that is capable of exploiting heterogeneity aspects at different points of the system design. Our system manages workloads according to the current penalty zones, which are provided by electrical-grid operators. The distribution of workloads is matched with available compute nodes and the current state of the grid in order to find a balanced operation point where required performance levels meet an economically efficient operation of the overall system.

Albatross leverages the heterogeneity of hardware components in two ways. On the one hand, Albatross reduces the power demand of the overall system and implements a *low-power operation mode* that is based on exploiting the heterogeneity characteristics of individual system components at the hardware level. For example, workloads are processed by different compute nodes that provide different execution units with varying power-efficiency characteristics. With the low-power mode Albatross reacts to excess demand at the power grid (i.e., electricity prices are high). On the other hand, Albatross deliberately increases the power demand of the overall system and accordingly implements a *high-power operation mode*, too. Our system uses the second operation mode when excess supply to the power grid occurs (i.e., due to large quantities of renewables) and electricity prices are low (i.e., potentially negative). To implement the two operation modes, Albatross uses data on power-demand characteristics of individual workloads and compute nodes, and combines this information with the current electricity price at runtime to dynamically allocate workloads to specific hardware components.

3 SYSTEM DESIGN

This section details the system design of Albatross. We first give an overview of design considerations with respect to the targeted HPC systems and describe the system model of Albatross. Next, we describe Albatross's system modules that interact with each other to jointly exploit the heterogeneous cluster components for implementing the low- and high-power operation modes. Finally, we present and discuss our implementation of Albatross.

3.1 System Model and Overview

We designed Albatross to be a system extension for existing workload managers that are used in traditional HPC systems.

3.1.1 HPC Workload Management. Figure 3 depicts a general HPC system and its components. The HPC system contains a coordinating *master node* that distributes pending work (*jobs*) among several *compute node*s, each having at least one *execution unit* (e.g., CPU). The *cluster manager* on the master node keeps track of all machines in the HPC cluster, whereas the *job scheduler* takes control of 1) receiving the jobs to process, 2) coordinating as well as enqueuing them, and 3) distributing them to individual compute nodes. The latter adheres to job constraints (e.g., resource demand, such as minimum number of CPUs or minimum amount of memory).

3.1.2 System Model. Figure 4 shows a high-level overview of Albatross's system model and its modules that are designed to be integrated into existing HPC systems. That is, with Albatross, we propose to extend HPC architectures as follows: The master node's functionality is supplemented with a *resource governor*, which is responsible for dynamic run-time decisions. It interacts with the job scheduler to retrieve job-execution information to be aware of the currently available free compute nodes. The resource governor receives incoming work orders and maintains a job queue, where pending jobs and their parameters are stored. This includes quality-of-service (QoS) information that reflect the performance requirements of the individual jobs. Furthermore, the resource governor receives electricity data (i.e., electricity prices, target- and penalty-zone limits), which it incorporates into its decision-making process. Compute nodes with similar heterogeneity properties are grouped into *resource groups*. For its job-allocation decisions, Albatross takes the following scopes of heterogeneity into account:

(1) System scope: varying design and architecture of the system (e.g., ARM, PowerPC, x86-64)
(2) Component scope: different execution units in a single system (e.g., CPU, integrated GPU, graphics card)
(3) Configuration scope: power-management features at the hardware level (e.g., C-states, DVFS, RAPL power capping)

Each compute node contains an additional *probe*, which on one side returns power-demand data to the resource governor, and on the other side implements power-management features (e.g., power capping) that are enforced by the resource governor. The resource governor runs a continuous feedback loop that matches incoming data (i.e., energy prices, job-execution information, power demand) with available heterogeneous compute-node resources to decide where and when to execute the individual jobs. With this approach, Albatross exploits the heterogeneity and diversity of multiple hardware components to control the cluster's power demand in

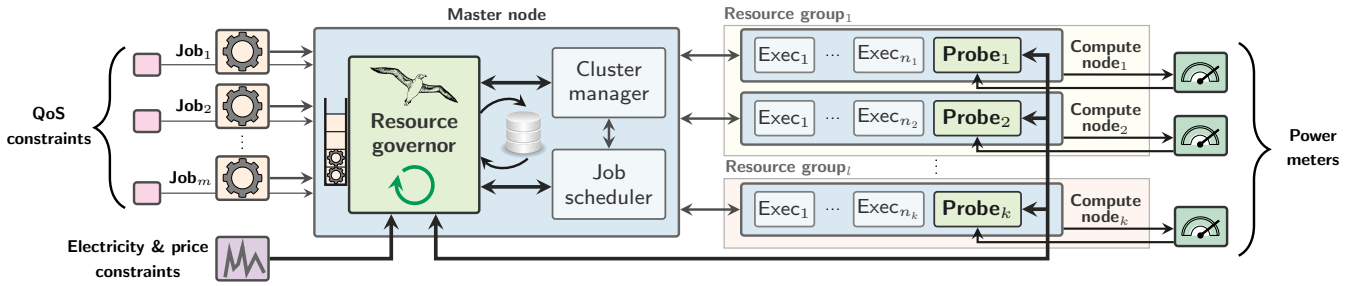


Figure 4: Overview of Albatross's integration into an HPC system. Albatross introduces 1) a controlling resource governor at the master node, 2) sensing as well as acting probes at each compute node, and 3) heterogeneity-aware resource groups.

dependence of the current electricity price. When electricity prices are high, Albatross enforces its low-power operation mode, which primarily aims for the best energy efficiency that is achievable in adherence with current performance constraints. When electricity prices are low (or even negative), Albatross employs its high-power operation mode, which loosens the reins and operates the cluster with higher power demand and performance.

In the subsequent sections, we present the individual modules of the Albatross runtime system. We first describe data that are the working basis for the resource governor (cf. Section 3.2), before we explain how this data are used to make dynamic job-allocation decisions (cf. Section 3.3) which are power and price aware.

3.2 Performance, Power, and Price Awareness

Albatross incorporates user-provided quality-of-service aspects (e.g., job-termination deadlines), which the system brings into accordance with controlling the cluster's power demand to reduce the electricity costs. First, Albatross makes itself aware of the composition of the cluster, all currently running jobs, and the performance requirements of each single job to adhere to certain termination deadlines (cf. Section 3.2.1). Second, Albatross gathers concrete power-demand values for all jobs to make assumptions about the energy efficiency (cf. Section 3.2.2). Third, Albatross incorporates the price of electricity, which dynamically changes at runtime, into its decision process (cf. Section 3.2.3).

3.2.1 Performance and Cluster Awareness. To retrieve cluster information and performance data, Albatross's resource governor uses the following interface, whose implementation is adapted to the specific underlying workload manager:

```
interface Albatross_Cluster_Control {
  /* Get cluster configuration */
  NodeList get_nodes();

  /* Get all running jobs and their allocation */
  JobList get_jobs();

  /* Submit job with attached job-resource info */
  JobResult submit_job(Job j);
}
```

The function `get_nodes()` retrieves a list of all available compute nodes in the heterogeneous HPC cluster. This function makes the resource governor aware of the number of compute nodes per resource group as well as the exact type of available hardware components (i.e., execution units) and their individual features. The information is either retrieved from the cluster manager or by directly accessing the compute nodes (e.g., using tools such as `lspci` or `lscpu`). The resource governor offers a `submit_job()` method, which is called instead of the original version of the workload manager when a job is submitted to the HPC cluster. A `Job` container comprises the binaries to be executed, the job parameters, and metadata (e.g., most suitable target platforms, QoS and power constraints). To quantify performance, the resource governor retrieves the complete execution time of an individual job by obtaining the value from `JobResult`, which is implemented as a future [22] and may block until the job has finished its execution.

3.2.2 Power Monitoring and Control. Power awareness is one of the key aspects of Albatross to keep the cluster's power demand in the target zone and avoid the penalty zones (cf. Figure 2). For this purpose, the resource governor uses the probes to access all power-metering devices and features that are available in the cluster. Albatross supports external measuring devices and CPU-integrated features, for example, Intel's running average power limit (RAPL) [23], which grants access to power-demand data of domains such as CPU, internal GPU, and DRAM. Each compute node contains a probe module to 1) retrieve power values and 2) control the node's power-management features. The probe module is used by the resource governor via the following probe interface:

```
interface Albatross_Power_Control {
  /* Triggers power measuring for an exec. unit */
  void start_measurement(Node n, ExecUnit u);
  PowerValues stop_measurement(Node n, ExecUnit u);

  /* Sets the value of a power-management feature */
  void set_pm(Node n, ExecUnit u, PowerConfig c);
}
```

The resource governor uses the probe methods to trigger the beginning (`start_measurement()`) and end (`stop_measurement()`) of a power measurement on a specific compute node for the job it is interested in. The call returns a list of power values, whose granularity depends on the type of measuring methodology (e.g., external measuring device, CPU-integrated RAPL). The probe's implementation is independent of the concrete underlying workload manager. This is also true for the `set_pm()` method, which adjusts the power-management feature of the execution unit of a

specific compute node. `PowerConfig` encapsulates an identifier for the power-management feature and the specific value to set. For example, a power cap for an Intel CPU is set by encapsulating a corresponding request, and is eventually enforced at the hardware level via RAPL. The resource governor is aware of the heterogeneity of available hardware and thus power-management features—since it gathered the corresponding information on the cluster topology.

3.2.3 Price and Electricity Constraints. The information on electricity prices and constraints (i.e., target and penalty zones) is deposited at the side of the grid operator. Thus, the resource governor receives this data from the grid operator, for example by using a web-service API [4]. The data are populated to a local database which the governor uses for dynamic price-aware job scheduling. The data entries contain specific timespans for the lower and upper power bounds as well as the prices for the target and penalty zones.

3.3 Dynamic Runtime Resource Allocation

The resource governor runs a continuous control and feedback loop which uses the previously described interfaces to retrieve values for power, execution time, current electricity constraints (power bounds, electricity prices), and available nodes in the cluster. The runtime data of the individual workloads are queried from the cluster during operation and are stored as execution traces in a database. The execution traces serve as a reference point for future runtime decisions by the resource governor. Whenever a new job is submitted, the resource governor first queries the database for an already existing execution-trace entry and, if present, uses this information for the pending resource-allocation decision.

The resource governor takes control of managing the jobs before actually submitting them to the job scheduler for further processing (cf. Figure 4). By inspecting the jobs' individual QoS constraints (i.e., processing deadlines) in combination with the currently running jobs on all nodes (i.e., cluster load), the resource governor decides which jobs are ready to be scheduled on which compute node. As not all jobs are immediately scheduled for execution, the resource governor maintains a separate queue for jobs that can be deferred. With the available runtime information, the resource governor of Albatross dynamically adjusts the power demand of the cluster to stay in the requested target zone. Depending on the current state of the power grid, Albatross pursues either the low- or the high-power operation mode. The low-power operation mode is achieved by dynamically deferring workloads and executing non-deferrable workloads on the most energy-efficient hardware platform that is currently available. The high-power operation mode, in contrast, is applied when energy prices decrease and may even become negative. In such situations, Albatross uses its ability to dynamically allocate as many resources as possible in order to increase the power demand of the cluster. The following section presents a corresponding implementation of Albatross.

3.4 Implementation

As a baseline for the implementation of our Albatross prototype we use the HPC workload manager SLURM [24], since it offers most of the base functionality that is required by Albatross and it is open source. The prototype implementation embraces the original system architecture which consists of three standalone software components: the control daemon `slurmctld`, the execution daemon `slurmd`, and several binaries (e.g., `srun`, `scontrol`, `sinfo`) that function as interfaces between the user and the daemons. In addition, a highly modular plug-in infrastructure is offered, which enables purposeful modifications of specific aspects while maintaining the base functionality to the greatest extent possible. In order to achieve a power- and price-aware management of the HPC cluster, the Albatross prototype implements three key features: 1) constraint-aware job-to-node–assignment strategies, 2) power-drain limitation by execution-environment adaptation, and 3) power-drain control by selective deferral of jobs. By adding these features to an existing cluster-management software such as SLURM, we can control the cluster's power drain to avoid penalty zones and achieve a better economic efficiency. The individual features of Albatross are implemented as follows.

Two of Albatross's features are implemented in a custom plug-in. This plug-in is based on the default `sched/builtin` plug-in, since it offers access to required data (i.e., job and node information) and has the best position in SLURM's existing internal algorithmic hierarchy. Within the plug-in, we implement the constraint-aware job-to-node–assignment strategy, which considers constraints (i.e., QoS, penalty zones, and electricity prices) and execution traces (cf. Section 3.3) during assignment decisions. The execution traces provide data such as average power drain and total energy demand. In addition to the assignment strategy, the plug-in also contains the master node's part of the power-limitation feature. Within the plug-in, information on the cluster's current power drain and zone constraints is combined with power-management features (i.e., power capping with RAPL) to achieve a power drain that violates electricity constraints as little as possible. Each applicable execution unit's power drain is limited separately. This is dynamically controlled byAlbatross's probe modules (cf. Section 3.1.2).

To further control the HPC cluster's power drain at runtime, we implemented an additional front-end component where jobs are either deferred or immediately forwarded to the job scheduler based on the cluster's current operation mode (i.e., low- and high-power operation modes, cf. Section 2). The selective deferral and forwarding of jobs help to avoid penalty zones in general, and increase the cluster's power drain while Albatross runs its high-power operation mode in particular.

Albatross's additional features reduce the costs of operating an HPC cluster by controlling the cluster's power drain. The implementation maintains as much of SLURM's default behavior as possible, which makes our prototype system a convenient drop-in replacement for HPC deployments that already use SLURM, but lack power and price awareness, yet.

4 EVALUATION

In this section, we evaluate Albatross on a heterogeneous HPC cluster in our lab. We first describe the evaluation setup that consists of the evaluated soft- and hardware as well as a power-measuring infrastructure (cf. Section 4.1). We explore the system, configuration, and component heterogeneity scopes in Section 4.2 and 4.3 and show how Albatross exploits heterogeneity aspects to stay within given power-target zones and how our prototype considers fluctuant electricity prices (cf. Section 4.4) dynamically at runtime.

4.1 Evaluation Setup

Figure 5 shows the evaluation setup, consisting of heterogeneous hardware components and the power-measuring infrastructure.

Cluster Setup. In alignment with the heterogeneity scopes that are considered by Albatross, we use a diverse set of heterogeneous hardware devices (cf. Table 1). Figure 5 (Ⓐ–Ⓔ) further shows the hardware setup and its arrangement in our lab. All devices are connected via switched 1 Gbps Ethernet.

Power Measuring. For the different hardware components we employ various power-measurement devices. To measure the power demand of individual compute nodes and the total power demand of the cluster, we use the Microchip MCP39F511 device (① in Figure 5). The MCP39F511 has a sampling rate of 400 Hz and a measuring error of ≤ 0.1 % [25]. To measure the power demand of graphics cards, we use a PCIe-lane–extending riser card (②) in combination with a current clamp that has a signal bandwidth of 1 MHz (③), and a STEMlab Red Pitaya board [26] (④). The board retrieves current values from the current clamp and multiplies them with the supply voltage to determine the power demand.

Evaluation Workloads. The evaluated workloads consist of different benchmarks from the NAS Parallel Benchmarks (NPB) suite [27]. We use two different versions of the benchmark: an OpenMP version to showcase system- and configuration-scope heterogeneity and an OpenCL version to showcase component-scope heterogeneity. We focus on the five kernel benchmarks (i.e., `CG`, `EP`, `FT`, `IS`, `MG`) with benchmark class A for standard test problems.

4.2 System- and Configuration-Scope Heterogeneity Effects

In the first experiment, we show the effects of choosing a certain system (i.e., compute node) for the execution of a job and investigate the effect of different configurations (i.e., with and without power cap). We vary the workload using the OpenMP version of NPB and discuss both energy-demand and execution-time values.

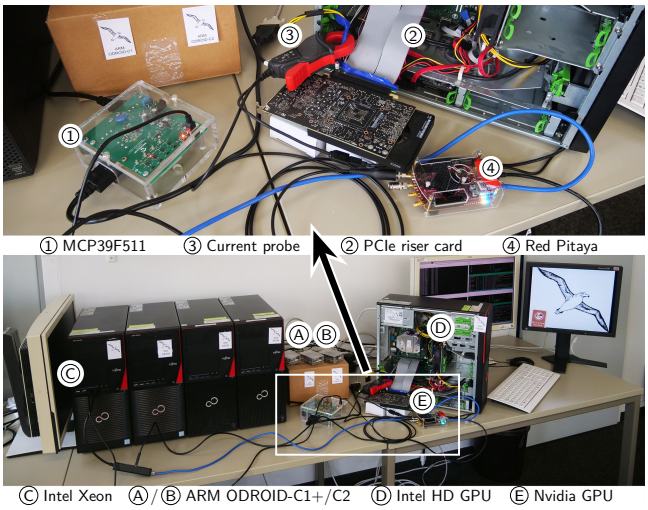


Figure 5: The evaluation setup consists of heterogeneous compute nodes and different power-measuring devices.

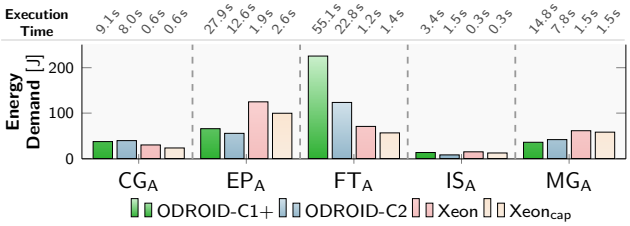


Figure 6: The energy demand and execution times of individual benchmarks vary for heterogeneous compute nodes.

4.2.1 Energy-Demand Results. For different NPB benchmarks and problem size A, Figure 6 shows energy-demand and execution-time values for the two ARM-based systems (ODROID-C1+ and ODROID-C2) and the Intel-based platform (Xeon). The fourth value (Xeon_{cap}) is a result of running the benchmarks with a power cap on the Xeon platform. We chose a power cap of 20 W as this configuration shows the best trade-off between energy demand and execution time across benchmarks.

From the energy-demand results in Figure 6, we make the following two observations: First, the values indicate that the platforms with the lowest and highest energy demand vary with the benchmark. For example, the ODROID-C1+ has the highest energy demand for the `FT` benchmark: it is 4x higher than the lowest measured energy demand (Xeon_{cap}). However, the same system (i.e., ODROID-C1+) has the lowest energy demand for the `MG` benchmark: it is 0.6x lower than using the Xeon without power cap—the highest observed energy demand. Second, applying the power cap on the Xeon platform reduces the energy demand for all benchmarks. The reduction varies between 7 % (`MG`) and 35 % (`FT`). This shows the importance of exploiting configuration-scope heterogeneity.

4.2.2 Execution-Time Results. The evaluation results shown in Figure 6 further reveal that the ARM-based systems (i.e., ODROID-C1+ and ODROID-C2) need significantly longer to process the benchmark workloads compared to the Intel-based platforms (i.e., Xeon and Xeon_{cap}). Even in the best case, the execution times on the ARM-based systems require 4.7x (`IS`) longer than on the Intel-based systems, and in the worst case, 44.7x (`FT`) longer.

The evaluation results of our first experiment highlight the importance of using Albatross, which considers both energy demand and execution times. It is necessary to not only consider the pure energy-demand values for the dynamic resource-allocation process, as this may violate QoS constraints (i.e., performance, deadlines).

Device	Processor details	Mem.
Ⓐ Hardkernel ODROID-C1+	Amlogic Cortex-A5 (4x 1.5 GHz)	1 GiB
Ⓑ Hardkernel ODROID-C2	Amlogic Cortex-A53 (4x 1.5 GHz)	1 GiB
Ⓒ Fujitsu CELSIUS W550	Intel Xeon E3-1275 v5 (8x 3.6 GHz)	16 GiB
Ⓓ Intel HD Graphics P530	24 pipelines (1.15 GHz)	shared
Ⓔ Nvidia Quadro P2000	1,024 CUDA cores (1.375 GHz)	5 GiB

Table 1: Overview of the evaluated devices with system-scope (A–C) and component-scope (D–E) heterogeneity.

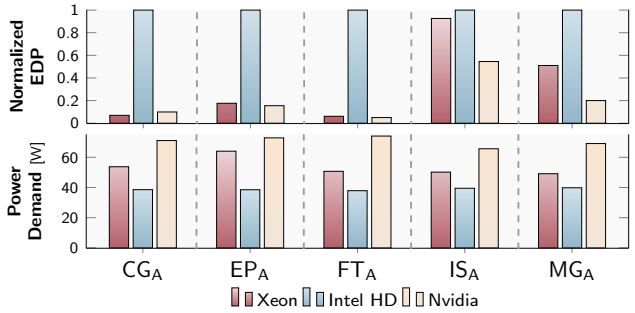
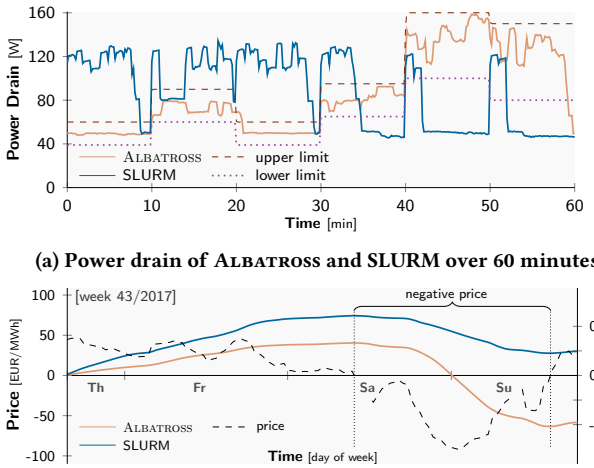


Figure 7: Comparison of the energy–delay product and average power-demand values when using different execution units with modules of the NAS Parallel Benchmarks.



(a) Power drain of Albatross and SLURM over 60 minutes.

(b) Accumulated electricity costs for Albatross and SLURM.

Figure 8: The power drain of the cluster varies and depends on the use of Albatross and SLURM (top graph). A comparison of the resulting, accumulated electricity costs shows the advantage of Albatross over SLURM (bottom graph).

4.3 Component-Scope Heterogeneity Effects

In our second experiment, we use the OpenCL version of NPB to show the effects of choosing different components (i.e., execution units): we compare the execution of benchmarks on a CPU (Xeon), an integrated GPU (Intel HD), and a graphics card (Nvidia).

Albatross considers the trade-off between energy demand and execution time for processing workloads and, thus, uses the energy–delay product (EDP) [20]. The EDP is defined as follows: EDP $= E \cdot t$, where E is a job's energy demand and t its execution time.

Figure 7 shows the corresponding EDP values (top graph) and average power-demand values (bottom graph). The EDP values are normalized to those of the integrated GPU (Intel HD). In all cases, the execution on Intel HD results in the highest EDP value, and all but one execution on the graphics card (Nvidia) results in the lowest. With a factor of 20.0x, the highest difference in EDP is observed for `FT` between Nvidia and Intel HD. The execution of `IS` results in the lowest difference—a factor of 1.8x between the Nvidia graphics card and the Intel HD internal GPU.

Since Albatross is in charge of controlling the cluster's power drain, we also evaluate the average power-demand values. On the one hand, Intel HD yields the highest EDP values, but also the lowest average power-demand values across all of our benchmarks. On the other hand, the Nvidia graphics card yields the lowest EDP values across almost all experiments, but also the highest average power-demand values across all of them. Even though Intel HD provides comparatively poor EDP results, it is still useful to balance the cluster's power drain and avoid penalty zones (cf. Figure 2).

The results from Section 4.2 and 4.3 highlight that there is no single system, component, or configuration that is first choice in all cases. Depending on available hardware devices and the workload to be executed, Albatross dynamically chooses the most suitable devices and configurations in a best-effort manner, in accordance with current power and electricity constraints.

4.4 Combining Power and Price Awareness

In the final experiment, we evaluate Albatross regarding power and price awareness. We use a versatile workload that is a large composition of all previously used benchmark types (i.e., OpenMP and OpenCL versions of NPB). The workload runs for 60 minutes during which each benchmark type receives the same amount of time to execute. We use SLURM (i.e., as baseline) and Albatross in sequence to execute the workload on the same HPC cluster. The resulting execution traces are then projected onto a longer period of time (i.e., four days) to calculate reasonable electricity costs.

Figure 8a shows the power drain of the HPC cluster during the experiment. The penalty zones (i.e., upper and lower limit) are dynamically adjusted every 10 minutes, based on the price fluctuation shown in Figure 8b. SLURM executes submitted jobs as fast as possible, ignoring the penalty zones. This leads to a frequent violation of the limits, resulting in extensive penalties. To the contrary, Albatross satisfies the limits and executes the workloads without spending significant amounts of time outside the target zone. In numbers, SLURM spends 83.1 % of the execution time within the penalty zone, whereas Albatross reduces this to merely 2.5 %. However, the completion time of individual jobs increases when using Albatross due to their deferred execution.

Next, we project the execution traces of this experiment onto a four-day time frame and merge our data with electricity prices as obtained from a power exchange [4]. Correspondingly, Figure 8b shows the accumulated electricity costs for running the heterogeneous HPC cluster over time. SLURM is unaware of the current electricity price and, thus, can not adapt the operation of the HPC cluster. Albatross, however, is aware of the current price and exploits the heterogeneity of the HPC cluster during operation. This leads to a reduction of the accumulated operation costs, and, in the concrete example, to a profit of 0.10 EUR (0.12 USD), whereas SLURM's operation leads to costs of 0.05 EUR (0.06 USD).

The numbers are small for the HPC cluster that operates in our lab as it has a low average power drain. On large systems, however, Albatross makes a considerable difference. For example, projecting our results onto the current Top-500 #1 supercomputer, Sunway TaihuLight [18], which has a power drain of 15.37 MW, would yield a profit of 16.7 thousand EUR (20.6 thousand USD) in just four days.

5 DISCUSSION

This section discusses issues regarding the practicability, adaptability, and combinability of Albatross.

Practicability. The workload utilization impacts the effectiveness of Albatross, for example, on HPC clusters and supercomputers with a sustained utilization of 100 %. Such systems leave only little room for dynamic resource allocation. Thus, job-to-node–assignment strategies of Albatross are limited as jobs can not be assigned to the most suited compute node or execution unit. Price awareness of Albatross is especially useful for large-scale systems that qualify for receiving negative prices. However, small customers also benefit as low prices are passed on, for example, by cheaper prices for HPC-as-a-Service [28]. To further increase profits, Albatross can be used for cryptocurrency-mining workloads. However, we strongly advocate climate-friendly goals to be pursued in such cases.

Adaptability. Albatross is compatible with the design of today's workload managers. We have inspected the features of actively maintained workload managers that are listed in Table 2. Our analysis has revealed that any of the listed workload managers can be extended to provide the core features of Albatross (i.e., heterogeneity, power, and price awareness). Unavailable features (i.e., red crosses in Table 2) need to be added to the individual workload manager to provide the basic functionality that is required by Albatross. Thus, we base our prototype on SLURM as it provides the majority of the necessary features.

Combinability. Computing resources that are available on spot markets [29] offer a cost-effective use of dynamically allocated computing resources. The consideration of using such compute resources from spot markets lowers costs significantly while the performance impact is insignificant [30]. Albatross gives the unique opportunity to combine dynamic pricing of different resource types (i.e., energy resources and compute resources).

6 RELATED WORK

Maximizing performance under a power limit is a common challenge for workload managers [31–36]. PTune [37] distributes power budgets among jobs and takes performance variants into account. Moreover, it determines the number of processors to use for each job. READEX [38] auto-tunes HPC applications to increase performance and energy efficiency. In contrast, Albatross does not require any program-code modifications; yet, such modifications would not be contradictory to its system design and can be seamlessly integrated. Wallace et al. [39] guide scheduling decisions towards higher power efficiency and show—at the example of a representative supercomputer system—that most jobs are repetitive and predictable. This further encourages the way Albatross traces jobs regarding power and performance for its decision process.

Helal et al. [40] and Goel et al. [41] propose to use performance and power models; however, establishing and using such models is cumbersome and error-prone. Therefore, Albatross conducts online measurements to work with values that best reflect the real world. Still, as done in previous work [38], we plan to extend Albatross with a machine-learning approach that uses profiling data to better satisfy power and electricity constraints when faced with unknown jobs. Electricity awareness has been addressed by Aikema et al. [10] and Yang et al. [3]. None of these works exploit heterogeneity to better adhere to given power-target zones.

Heterogeneity in HPC systems is affected at multiple levels. First, the software that runs on HPC systems becomes increasingly heterogeneous in the future. For example, Baer et al. [42] started to run Apache Spark applications on an HPC cluster. Such applications have different workload patterns, which are more dynamic, requiring adequate response at runtime, as is the case with Albatross. Second, hardware components become even more heterogeneous. For example, works such as the one from Fornaciari et al. [43] incorporate, besides desktop and HPC systems, embedded devices into a multi-layer resource manager. However, the presented results indicate an early stage of development and, in contrast to Albatross, the resource manager lacks the ability to exploit multiple heterogeneous hardware resources at the same time. Commercial cluster managers such as Univa Grid Engine [44] and Moab [45] are already prepared for heterogeneity support, however, in contrast to Albatross, they do not take power or energy demand into consideration. Wang et al. [46] and Tang et al. [11] show that with other GPU and CPU models an even more diverse behavior in terms of energy and power is achievable. Thus, further increasing the amount of component-scope heterogeneity with different GPUs and CPUs is especially beneficial for Albatross when adhering to dynamic pricing and power-target zones.

Feature	Moab	Tivoli	Univa	SLURM	**A'ross**
Job pinning	✗	✗	✓	✓	✓
CPU allocation	✓	✓	✗	✓	✓
Quality of service	✓	✗	✓	✓	✓
Generic resources	✓	✗	✓	✓	✓
Cluster status	✓	✓	✓	✓	✓
Heterogeneity aware	✓	✗	✓	✗	✓
Power/price aware	✗	✗	✗	✗	✓

Table 2: Feature comparison of Albatross with several other state-of-the-art workload managers.

7 CONCLUSION

This paper presents Albatross, a runtime system that takes on exploiting negative electricity prices when managing workloads on heterogeneous HPC clusters. Our system uses heterogeneity at the hardware level to dynamically control and adjust the power demand according to current workloads, QoS requirements, and external constraints (i.e., electricity price). When the amount of renewables is high and prices become negative, Albatross makes profit by adapting workloads. In practice, the achieved profit further increases when the system is used for cryptocurrency mining. However, we advocate climate-friendly goals to be pursued.

ACKNOWLEDGMENTS

We thank Heiko Janker for his help towards a precise and accurate power-measuring infrastructure and we highly appreciate the insightful feedback of the anonymous reviewers. This work was partially supported by the German Research Council (DFG) under grant no. SCHR 603/13-1 ("PAX"), grant no. SFB/TR 89 ("InvasIC"), and grant no. DI 2097/1-2 ("REFIT").

REFERENCES

[1] Hassan Farhangi. The path of the smart grid. *IEEE Power and Energy Magazine*, 8(1):18–28, January 2010.

[2] Julio Romero Aguero, Erik Takayesu, Damir Novosel, and Ralph Masiello. Modernizing the grid: challenges and opportunities for a sustainable future. *IEEE Power and Energy Magazine*, 15(3):74–83, May 2017.

[3] Xu Yang, Zhou Zhou, Sean Wallace, Zhiling Lan, Wei Tang, Susan Coghlan, and Michael E Papka. Integrating dynamic pricing of electricity into energy aware scheduling for HPC systems. In *Proceedings of the 2013 International Conference on High Performance Computing, Networking, Storage and Analysis (SC'13)*, pages 1–11. IEEE, 2013.

[4] Fraunhofer ISE. Electricity production and spot prices in Germany. URL: www.energy-charts.de/price.htm.

[5] Harry Wirth and Karin Schneider. Recent facts about photovoltaics in Germany. Technical report, Fraunhofer Institute for Solar Energy Systems, 2015.

[6] Mareike Jentsch, Tobias Trost, and Michael Sterner. Optimal use of power-to-gas energy storage systems in an 85% renewable energy scenario. *Energy Procedia*, 46:254–261, December 2014.

[7] Alberto Varone and Michele Ferrari. Power-to-liquid and power-to-gas: an option for the German Energiewende. *Renewable and Sustainable Energy Reviews*, 45:207–218, May 2015.

[8] Stanley Reed. Power prices go negative in Germany, a positive for consumers. *The New York Times*, 167:B3, December 2017.

[9] Matthew R Shaner, Steven J Davis, Nathan S Lewis, and Ken Caldeira. Geophysical constraints on the reliability of solar and wind power in the United States. *Energy and Environmental Science*, 11:914–925, 2018.

[10] David Aikema, Cameron Kiddle, and Rob Simmonds. Energy-cost-aware scheduling of HPC workloads. In *Proceedings of the 2011 International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM '11)*, pages 1–7. IEEE, 2011.

[11] Kun Tang, Devesh Tiwari, Saurabh Gupta, Sudharshan S Vazhkudai, and Xubin He. Effective running of end-to-end HPC workflows on emerging heterogeneous architectures. In *Proceedings of the 2017 International Conference on Cluster Computing (CLUSTER '17)*, pages 344–348. IEEE, 2017.

[12] James Jeffers and James Reinders. *Intel Xeon Phi Coprocessor High Performance Programming*. Morgan Kaufmann, 2013.

[13] Barry Rountree, David K Lownenthal, Bronis R De Supinski, Martin Schulz, Vincent W Freeh, and Tyler Bletsch. Adagio: making DVS practical for complex HPC applications. In *Proceedings of the 2009 International Conference on Supercomputing (ICS '09)*, pages 460–469. ACM, 2009.

[14] Mohamed Zahran. Heterogeneous computing: here to stay. *Commuications of the ACM*, 60(3):42–45, February 2017.

[15] Prasanna Balaprakash, Ananta Tiwari, and Stefan M Wild. Multi objective optimization of HPC kernels for performance, power, and energy. In *Proceedings of the 2013 International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS '13)*, pages 239–260. Springer, 2013.

[16] H Peter Hofstee. Power efficient processor architecture and the cell processor. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture (HPCA '05)*, pages 258–262. IEEE, 2005.

[17] Maja Etinski, Julita Corbalan, Jesus Labarta, and Mateo Valero. Optimizing job performance under a given power constraint in HPC centers. In *Proceedings of the 2010 International Green Computing Conference (IGCC '10)*, pages 257–267. IEEE, 2010.

[18] Wu-chun Feng and Kirk Cameron. The Green500 list: encouraging sustainable supercomputing. *IEEE Computer*, 40(12), December 2007.

[19] Afshin Izadian, Nathaniel Girrens, and Pardis Khayyer. Renewable energy policies: a brief review of the latest US and EU policies. *IEEE Industrial Electronics Magazine*, 7(3):21–34, September 2013.

[20] Mark Horowitz, Thomas Indermaur, and Ricardo Gonzalez. Low-power digital design. In *Proceedings of the 1994 Symposium on Low Power Electronics*, pages 8–11. IEEE, 1994.

[21] Hayk Shoukourian, Torsten Wilde, Axel Auweter, and Arndt Bode. Power variation aware configuration adviser for scalable HPC schedulers. In *Proceedings of the 2015 International Conference on High Performance Computing & Simulation (HPCS '15)*, pages 71–79. IEEE, 2015.

[22] Henry C Baker Jr and Carl Hewitt. The incremental garbage collection of processes. *ACM SIGPLAN Notices*, 12(8):55–59, August 1977.

[23] Intel Corporation. Intel 64 and IA-32 architectures software developer's manual: system programming guide, part 2, 2016.

[24] Andy B Yoo, Morris A Jette, and Mark Grondona. SLURM: simple Linux utility for resource management. In *Proceedings on the 2003 Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '03)*, pages 44–60. Springer, 2003.

[25] Microchip MCP39F511. URL: www.microchip.com/wwwproducts/en/MCP39F511.

[26] StemLabs Red Pitaya. URL: www.redpitaya.com.

[27] NAS Parallel Benchmarks. URL: www.nas.nasa.gov/publications/npb.html.

[28] High Performance Computing on Amazon AWS. URL: aws.amazon.com/hpc.

[29] Qi Zhang, Quanyan Zhu, and Raouf Boutaba. Dynamic resource allocation for spot markets in cloud computing environments. In *Proceedings of the 2011 International Conference on Utility and Cloud Computing (UCC '11)*, pages 178–185. IEEE, 2011.

[30] Supreeth Subramanya, Tian Guo, Prateek Sharma, David Irwin, and Prashant Shenoy. SpotOn: a batch computing service for the spot market. In *Proceedings of the 2015 Symposium on Cloud Computing (SoCC '15)*, pages 329–341. ACM, 2015.

[31] Huazhe Zhang and Henry Hoffmann. Maximizing performance under a power cap: a comparison of hardware, software, and hybrid techniques. In *Proceedings of the 2016 International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*, pages 545–559. ACM, 2016.

[32] Osman Sarood, Akhil Langer, Abhishek Gupta, and Laxmikant Kale. Maximizing throughput of overprovisioned HPC data centers under a strict power budget. In *Proceedings of the 2015 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '15)*, pages 807–818. ACM/IEEE, 2015.

[33] Peter E. Bailey, David K. Lowenthal, Vignesh Ravi, Barry Rountree, Martin Schulz, and Bronis R. de Supinski. Adaptive configuration selection for power-constrained heterogeneous systems. In *Proceedings of the 2014 International Conference on Parallel Processing (ICPP '14)*, pages 371–380. IEEE, 2014.

[34] Allan Porterfield, Rob Fowler, Sridutt Bhalachandra, Barry Rountree, Diptorup Deb, and Rob Lewis. Application runtime variability and power optimization for exascale computers. In *Proceedings of the 2015 International Workshop on Runtime and Operating Systems for Supercomputers (ROSS '15)*, pages 1–8. ACM, 2015.

[35] Ziming Zhang, Michael Lang, Scott Pakin, and Song Fu. Trapped capacity: scheduling under a power cap to maximize machine-room throughput. In *Proceedings of the 2014 Energy Efficient Supercomputing Workshop (E2SC '14)*, pages 41–50. IEEE, 2014.

[36] Deva Bodas, Justin Song, Murali Rajappa, and Andy Hoffman. Simple power-aware scheduler to limit power consumption by HPC system within a budget. In *Proceedings of the 2014 International Workshop on Energy Efficient Supercomputing (E2SC '14)*, pages 21–30. IEEE, 2014.

[37] Neha Gholkar, Frank Mueller, and Barry Rountree. Power tuning HPC jobs on power-constrained systems. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation (PACT '16)*, pages 179–191. ACM, 2016.

[38] Per Gunnar Kjeldsberg, Andreas Gocht, Michael Gerndt, Riha Lubomir, Joseph Schuchart, and Umbreen Sabir Mian. READEX: linking two ends of the computing continuum to improve energy-efficiency in dynamic applications. In *Proceedings of the 2017 International Conference on Design, Automation, and Test in Europe (DATE '17)*, pages 109–114. EDAA, 2017.

[39] Sean Wallace, Xu Yang, Venkatram Vishwanath, William E. Allcock, Susan Coghlan, Michael E. Papka, and Zhiling Lan. A data driven scheduling approach for power management on HPC systems. In *Proceedings of the 2016 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '16)*, pages 656–666. ACM/IEEE, 2016.

[40] Ahmed E. Helal, Wu-chun Feng, Changhee Jung, and Yasser Y. Hanafy. AutoMatch: an automated framework for relative performance estimation and workload distribution on heterogeneous HPC systems. In *Proceedings of the 2017 International Symposium on Workload Characterization (IISWC '17)*, pages 32–42. IEEE, 2017.

[41] Bhavishya Goel, Sally A McKee, Roberto Gioiosa, Karan Singh, Major Bhadauria, and Marco Cesati. Portable, scalable, per-core power estimation for intelligent resource management. In *Proceedings of the 2010 International Conference on Green Computing (IGCC '10)*, pages 135–146. IEEE, 2010.

[42] Troy Baer, Paul Peltz, Junqi Yin, and Edmon Begoli. Integrating Apache Spark into PBS-based HPC environments. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure (XSEDE '15)*, pages 1–7. ACM, 2015.

[43] William Fornaciari, Gianmario Pozzi, Federico Reghenzani, Andrea Marchese, and Mauro Belluschi. Runtime resource management for embedded and HPC systems. In *Proceedings of the 2016 Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and the 2016 Workshop on Design Tools and Architectures For Multicore Embedded Computing Platforms (PARMA-DITAM '16)*, pages 31–36. ACM, 2016.

[44] Univa Grid Engine. URL: www.univa.com/products.

[45] Moab HPC Suite. URL: adaptivecomputing.com/products/hpc-products/moab-hpc-basic-edition/basic-edition-solution-architecture.

[46] Qiang Wang, Pengfei Xu, Yatao Zhang, and Xiaowen Chu. EPPMiner: an extended benchmark suite for energy, power and performance characterization of heterogeneous architecture. In *Proceedings of the 2017 International Conference on Future Energy Systems (e-Energy '17)*, pages 23–33. ACM, 2017.