FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Promotionsarbeit in Informatik

Adviser for Energy Consumption Management: Green Energy Conservation

Hayk Shoukourian

TECHNISCHE UNIVERSITÄT MÜNCHEN

Fakultät für Informatik
Lehrstuhl für Rechnertechnik und Rechnerorganisation

Adviser for Energy Consumption Management: Green Energy Conservation

Hayk Shoukourian

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Thomas Huckle
Prüfer der Dissertation:	
	1. Univ.-Prof. Dr. Arndt Bode
	2. Univ.-Prof. Dr. Dieter Kranzlmüller *Ludwig-Maximilians-Universität München*

Die Dissertation wurde am 11.06.2015 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 23.07.2015 angenommen.

Acknowledgments

This dissertation has been kept on course and been led to a successful completion with the tremendous support and encouragement of numerous people, who I wish to thank.

First and foremost I would like to express my sincere gratitude to my supervisor Professor Arndt Bode, for his systematic and thorough guidance, throughout which he engaged me in new ideas, stimulated my professional growth, and, at the same time, gave me an intellectual freedom in the research, simultaneously requesting a high quality of work in all my aspirations. Once again, I would like to thank him for his valuable support, comments, and remarks, as well as for giving me the great opportunity to be a part of the research and development done at Leibniz Supercomputing Centre (LRZ).

I would also like to offer my profound appreciations to my second supervisor, Professor Dieter Kranzlmüller, for his kind agreement and time for reviewing this dissertation.

Furthermore, I would like to express my thankfulness to all my friends and colleagues at LRZ for their cheering, support and all the great moments spent together throughout these years. In particular, to Torsten, who was my M.Sc. thesis adviser, and latter a colleague and friend - a person, who never stopped advising me and is always ready to share his multi-year professional experience. Many thanks Torsten, for all the interesting discussions, joint brainstorming, and comments. To Jeanette, who despite her occupation, was always there to proofread my several research papers (including this work) as well as willing to assist the advancement of my scientific article writing skills. To Axel, for all his helpful suggestions and valuable remarks. To Daniele, Matteo, Michael, Jimmy, Siew Hoon (Cerlane), Volker, Alex, Anupam, and many others for making my time here at LRZ a lot more fun.

Last but not the least, I would like to express my eternal gratitude to my family - to my lovely wife for her unconditional support, loving care, and for being always there, cheering me on in all my pursuits. To my dearest parents for their continuous encouragement and unwavering spiritual support over so many years, without which none of this would have been possible.

Hayk Shoukourian

Munich, Germany
August 2015

Abstract

The power consumption of High Performance Computing (HPC) systems, which are the key technology for many modern computation-intensive applications, is rapidly increasing in parallel with their performance improvements. These increased power expenditures together with the rising electricity costs, lead to the necessity for prioritization of energy efficiency requirements when designing and expanding modern HPC data centers. It is equally important to ensure the sustainable, reliable, and environmental-friendly functioning of the data centers during the normal operational modes as well as in emergency situations. Power and energy capping are two of emerging techniques aimed towards controlling and efficient budgeting of power and energy consumption within the data center.

This dissertation examines the prerequisites and proposes a framework, referred to as Energy Consumption Management Adviser (ECMA), to enhance the energy-efficient application scheduling and backfilling decisions as well as to support for power and energy capping. This framework encompasses a unified energy measurement and evaluation toolset, which is an integrated solution aimed at monitoring, collection, and correlation of power consumption relevant data from all the components of the target supercomputing site, stretching from the environmental aspects and data center building infrastructure over the deployed HPC systems and reuse technologies, to the system software stack and executed large-scale applications. The main aim of the developed toolset is not to replicate any of the existing data center monitoring tools, and rather to act as a hub among several monitoring tools for provision of an integrated view on energy and power flows of the complete HPC data center without which the energy efficiency improvements would be non-trivial and incomplete.

The developed ECMA framework features models that allow for a beforehand estimation of execution time, average power and aggregated energy consumption profiles for given HPC applications. These models are automatically calibrated for the increase of the prediction accuracy during the lifetime of the target HPC system, using the collected and evaluated data obtained from the presented monitoring solution. Based on these models, a Configuration Adviser (CA) plug-in for resource management and scheduling systems is presented. This plug-in uses an efficient algorithm to determine an energy-wise optimal resource configuration for a given application. This algorithm: *(a)* ensures that the application will be executed with the energy optimal resource configuration fulfilling the specified application execution time and average power consumption constraints; and *(b)* considers the existing power diversity among the compute nodes of a given homogeneous HPC system. The presented CA plug-in is in complement to the energy/power saving techniques present in current resource management and scheduling systems and can be used in conjunction with any other energy/power reduction efforts.

The currently foreseen application range of the ECMA framework includes, but is not limited to, the usage: *(i)* by data center operators to completely assess the current energy-efficiency status and the success of the applied optimization solutions as of the residing individual HPC systems as well as of the integral data center; *(ii)* by HPC users/clients for estimating the potential execution time, power, and energy costs of their applications as well as for the dynamic control over their energy budgets; *(iii)* for the support of energy and power capping techniques; and *(iv)* for further enhancements of existing energy-aware resource management and scheduling solutions.

Zusammenfassung

Der Stromverbrauch von Hochleistungs-Rechensystemen, die eine Schlüsseltechnologie für viele moderne rechenintensive Anwendungen darstellen, steigt parallel zu ihrer Leistungsverbesserungen. Der erhöhte Verbrauch geht mit steigenden Energiekosten einher. Somit ist die Steigerung der Energieeffizienz ein wichtiges Planungskriterium für den Entwurf und Ausbau moderner Hochleistungsrechenzentren. Ebenso wichtig ist es, die nachhaltige, zuverlässige, und umweltfreundliche Arbeitsweise der Datenzentren sowohl im normalen Betriebsmodus als auch in Notfallsituationen zu gewährleisten. "Power Capping" und "Energy Capping" sind zwei aufkommende Techniken für effiziente Steuerung und Planung von Leistungs- und Energieverbrauch im Rechenzentrum.

Diese Dissertation untersucht die notwendigen Voraussetzungen um die energieeffiziente Anwendungsplanung zu verbessern sowie "Power Capping" und "Energy Capping" zu unterstützen, und schlägt ein Modell dazu vor, das als "Energy Consumption Management Adviser" (ECMA) bezeichnet wird. Dieses Modell beinhaltet eine Lösung für die einheitliche Energiemessung und Auswertung aller Komponenten des Rechenzentrums, angefangen von Umweltaspekten und Gebäudeinfrastruktur über Informationen des eingesetzten Hochleistungs-Rechensystems und die Wiederverwendung-Technologien, bis hin zur Betriebssoftware und den ausgeführten Anwendungen. Das Hauptziel des entwickelten Werkzeugs ist nicht ein bestehendes Rechenzentrumsüberwachungs-Werkzeug nachzuahmen, sondern eine Verbindung zwischen mehreren Überwachungswerkzeugen herzustellen, um eine integrierte Sicht auf Leistungs- und Energieflüsse des gesamten Hochleistungsrechenzentrums zu haben. Ohne diese Sicht ist die Verbesserung der Energieeffizienz nicht trivial und möglicherweise unvollständig.

Das entwickelte ECMA Modell umfasst Teil-Modelle, die die Vorhersage von Ausführungszeit, Durchschnittsleistung und Energieverbrauch für eine gegebene parallele Anwendungen ermöglichen. Diese Modelle werden automatisch während der Lebensdauer des Hochleistungs-Rechensystems kalibriert um die Vorhersagegenauigkeit zu steigern. Basierend auf diesen Modellen, wird ein "Configuration Adviser (CA)" Plug-In für das Ressourcenmanagement und Planungssysteme (Scheduler) vorgestellt. Dieses Plug-In verwendet einen effizienten Algorithmus, um eine energetisch optimale Konfiguration der Anwendung zu ermitteln. Dieser Algorithmus: *(a)* stellt sicher, dass die Anwendung mit der energetisch optimalen Ressourcenkonfiguration ausgeführt wird, welche mögliche Ausführungszeit- und durchschnittliche Stromverbrauch-Beschränkungen erfüllt; und *(b)* berücksichtigt die bestehenden Stromverbrauchsunterschiede der Rechenknoten eines bestimmten homogenen Hochleistungs-Rechensystems. Das vorgestellte CA Plug-In ergänzt die derzeitigen Leistung- und Energiespartechniken in Ressourcenmanagement Systemen und kann in Verbindung mit anderen Leistungs- und Energiespartechniken verwendet werden.

Die Mögliche Anwendungsgebiete des ECMA sind vielfältig. Derzeit ist eine Verwendung in folgenden Bereichen geplant: *(i)* für die Betreiber von Rechenzentren, um die Energieeffizienz des vollständigen Rechenzentrums, sowie den Erfolg der anwendungsbezogenen Optimierungslösungen zu bewerten; *(ii)* für Nutzer/Kunden zur Abschätzung der potenziellen Ausführungszeit, Leistungs- und Energiekosten ihrer Anwendungen sowie für die dynamische Kontrolle über ihre Energiebudgets; *(iii)* für die Unterstützung von

“Power Capping” und “Energy Capping” Techniken; und *(iv)* für mögliche weitreichende Verbesserungen bestehender Ressourcenmanagements- und Planungs-Systeme.

Contents

List of Figures

List of Tables

Part I.

Energy Efficient Management in High Performance Computing

"He who seeks for methods without having a definite problem in mind seeks in the most part in vain"
DAVID HILBERT

1. Power and Energy Capping in HPC Data Centers

1.1. Motivation and Problem Statement

High Performance Computing (HPC) systems are getting more and more computationally powerful making power consumption a vital issue for many modern data centers. Some of the current HPC systems consume more than 17 MW of electrical power[1] [1] making the energy consumption a dominating factor for the Total Cost of Ownership (TCO) of the target HPC system [2], defined as the entire costs spent on using and acquiring assets [3]. Apart from the high power bills, the power consumption of this magnitude could already cause the entire data center power delivery and cooling infrastructures to breach the safety limits as well as affect the environmental sustainability, by producing high carbon footprint. All these issues make a well-defined and efficient energy management process a necessity for providing a sustainable functioning of the deployed Information Technology (IT) systems, keeping operational costs in budget, and reducing possible environmental impacts.

Power capping is one of the emerging management techniques used for limiting the amount of power a system can consume when executing various applications, thus aiming to keep the system usage within a given power band and prevent possible power overloads [4]. In general, power capping covers a wide range of use cases - it can be used for:

- **ensuring the sustainable operation of the current, future, or extended high-end systems within the current data center infrastructure capacity limits**
  The average power consumption of the SuperMUC supercomputer (described in Section 2.5) in the normal operation modes is less than 2.3 MW [5], whereas under High Performance Linpack (HPL) benchmark [6] the power consumption of the SuperMUC reaches 3.4 MW [1]. For example, with the 20 MW power consumption goal for the future Exascale systems[2] [7] this would translate to around 30 MW peak power consumption for Exascale systems under HPL benchmark, which could be already an infrastructure capacity limit requiring additional resources (e.g. backups through diesel generators, etc.) for some data centers. Power capping could be a possible solution for a high-end performance provision within the data center current infrastructure capacity limits for the normal operational modes of the target system.

[1]Which is already a sufficient amount of power for sustaining a small city.

[2]Systems representing the next thirtyfold increase in computing capabilities beyond currently existing Multi-Petascale systems.

- **operating the system with renewable energy sources**
  In order to support for a renewable energy source based (e.g. hydroelectrical energy, solar energy, wind energy, etc.) operation of the target system, the dynamic of the available power must be taken into account, since in the considered case the available portion of the power will change with the time. Power capping could help to ensure that the currently available power capacity is never violated.

- **controlling the peak power consumption**
  The operation at the peak power consumption of the target large-scale system affects the:
  - capital expenses
    Due to the out-of-band requirements in the power distribution and cooling infrastructures for providing the required power and cooling down the heat generated from that power.
  - operational expenses
    Since in addition to the per 1 kWh energy cost, the majority of the utility contracts for data centers include a penalty fee for violating the pre-defined power band boundaries[3] [8].

  Power capping mechanism could further ensure that no violations of the power band boundaries are recorded.

- **re-charge of UPS's within the capacity limits**
  Modern data centers are enhanced with Uninterruptible Power Supplies (UPS's) for providing a "backup" power in case of power failures. After the power supply is restored, the UPS's consume power for re-charging. This would imply a requirement for **either**: *(i)* an additional power capacity (apart from the one dedicated for operating the system) allocation for re-charging the UPS's, which would most probably be unused after the batteries will be recharged due to the rareness fact of the power failures **or** *(ii)* a temporary system power capacity straining for freeing up a power margin required for re-charging. Power capping can be applied for implementing the latter, more energy-efficient, approach, by ensuring the availability of the required (for re-charging) power margin.

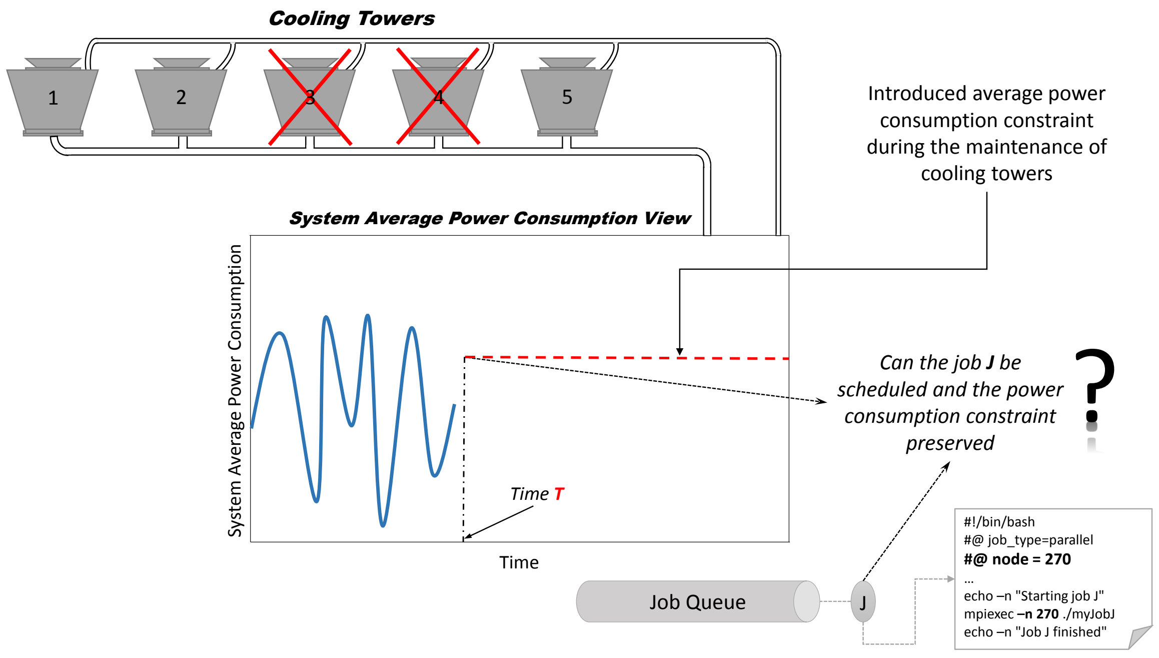
- **cost efficient scheduling**
  Assume the Resource Management and Scheduling System (RMSS) of the target high-end system supports for a priority based scheduling of various applications with different power consumption profiles and the data center infrastructure is designed for providing the power capacity for near peak usage for high priority applications. Assume further that these near peak operating applications are served only for a fraction of a day, leaving the dedicated power capacity underutilized for the rest of the time. During these time fractions, the low priority applications can be executed for increasing the mentioned utilization rate. Power capping can be used for ensuring the usage of only those power resources that are unused by currently executed high-priority applications without intervention to their performance.

[3]This fee can propagate to the entire operational year bill, even if the power band was violated for a single time.

- **controlling the power budget of HPC users**
  Most of the HPC users/clients (e.g. groups from industrial, research and educational institutions, etc.) can be charged according to consumed Central Processing Unit (CPU) hours. With the increased knowledge on power consumption of the large-scale applications and the need for keeping the capital and operational expenses within the existing power budget, data center operators might consider introducing power consumption budgets to the users and/or caps on the average power consumption of their various applications. Power capping could help to ensure that these user-level average power consumption constraints are never violated and that the users operate within their power budgets and are never additionally charged or penalized.

- **ensuring the operational continuity of the target system in case of power outage**
  The limitation in cooling capacities and/or in power deliveries (which could arise, for example, due to the data center infrastructure planned maintenance, unexpected failures of cooling towers, etc.), would force data center operators to introduce a system power cap that must be followed in order to prevent power overloads and allow for further system utilization.

Figure 1.1 illustrates this last use case scenario. The blue solid curve shows the average power consumption behavior of a given HPC system cooled with the use of five cooling towers depicted on the top of the image. Assume that at time T two of the data center's cooling towers are taken into maintenance, introducing a temporary average power consumption constraint for the complete system.


Cooling Towers
1
2
3
4
5
Introduced average power consumption constraint during the maintenance of cooling towers
System Average Power Consumption View
System Average Power Consumption
Can the job J be scheduled and the power consumption constraint preserved
?
Time T
Time
Job Queue
J

```
#!/bin/bash
#@ job_type=parallel
#@ node = 270
...
echo –n "Starting job J"
mpiexec –n 270 ./myJobJ
echo –n "Job J finished"
```

Figure 1.1.: Power capping example use case scenario

Assume further that there is a queued job (application) J with a utilization requirement of 270 compute nodes (servers), which needs to be scheduled for execution. The question that needs to be answered here would be: *is the execution of job J with the request of* 270 *compute nodes possible within the introduced average power consumption constraint?* None of the currently available RMSS's can answer this question - while the lack of this knowledge could lead to a possible scheduling of job J, which in its turn could overload the available cooling capacity.

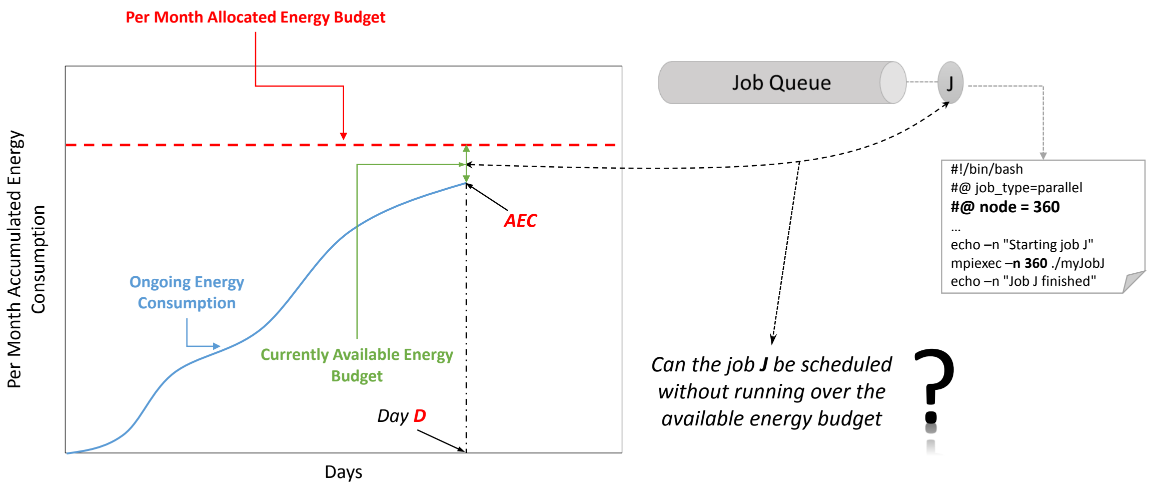
The majority of current techniques that implement power capping (e.g. [9, 10, 11]) involve Dynamic Voltage and Frequency Scaling (DVFS) [12, 13], which is a power management technique for a dynamic adjustment of the supply voltage and operating frequency of the underlying processor for satisfying certain power consumption constraints. DVFS is also used when an increase of compute performance is required - in this case the supply voltage as well as the operating frequency of the processor are increased. A number of modern processors such as Intel's Sandy Bridge[4] are performing the power-performance control through the DVFS. One of the major blocks of the Sandy Bridge's power management architecture is the Package Control Unit (PCU) [14]. The PCU is an integrated patchable microcontroller used for power/thermal management of the processor. Its firmware permanently collects power and thermal data and performs various power/performance optimizations by communicating with the external voltage regulator and the embedded controller responsible for system-wide power management. The PCU of the Intel's Sandy Bridge processor, in contrast to the PCU of the former generation of Nehalem processors, features a more complex model allowing to predict the package's active power consumption by collecting various architectural events from each core, processor graphics, and I/O. The PCU of Intel's Sandy Bridge performs energy budgeting by tracking periods where the CPU is idle or consuming less than the determined Thermal Design Power (TDP) value (which indicates the average maximum power consumption value a processor can consume without overheating [12]) and accumulates the unused amount of energy. It then uses this accumulated amount of energy to consume more power than the set steady TDP limit for short periods ($\sim 30-60$ seconds [14]) in order to achieve some performance gains, and then rolls back to a stable power state. Whenever the Operating System (OS) detects a performance requirement, it issues a corresponding performance state (P-state) [15] request. Higher P-states imply slower processor speeds and lower power consumption [16], i.e. a processor in the $P3$ state runs more slowly and consumes less power than a processor running in the $P2$ state[5]. The power management algorithms of Sandy Bridge processors dynamically track and assess (via power-management agents [14]) the current allowance of various system physical constraints, such as silicon capabilities, thermomechanical capabilities, current power-delivery/battery capabilities, etc. If these checks show that there still exists available power and thermal margin, the PCU of the processor increases the voltage and frequency to a point, that is lower and or equal the OS requested state and still meets the all system related physical constraints [14]. This feature is especially useful for applications showing highly dynamic power usage profile by providing short performance boosts when needed.

On the other hand, the DVFS technique, when applied for power conservation, will in most cases, increase the runtime of the application [13] (because of possible decrease in

[4]The processors used in the SuperMUC supercomputer deployed at LRZ and described in Section 2.5.

[5]The number of the supported P-states is processor-specific and varies with different types.

supply voltage and operating frequency), thus increasing the integral of power consumption over time (i.e. energy). **Energy capping** is another management technique that limits the amount of energy a system can consume when executing applications for a given time period. In other words, energy capping limits the integral amount of power consumption over time and, in contrast to power capping, it does not limit the amount of power the system can consume at a given point in time. Figure 1.2 illustrates an example use case scenario for energy capping.


Per Month Allocated Energy Budget
Per Month Accumulated Energy Consumption
AEC
Ongoing Energy Consumption
Currently Available Energy Budget
Day D
Days
Job Queue
J
Can the job J be scheduled without running over the available energy budget
?

```
#!/bin/bash
#@ job_type=parallel
#@ node = 360
…
echo –n "Starting job J"
mpiexec –n 360 ./myJobJ
echo –n "Job J finished"
```

Figure 1.2.: Energy capping example use case scenario

The dashed red line shows the introduced per month allocated energy budget that can be consumed on a monthly basis (this can be for the whole system or on a per user/customer basis) whereas the blue solid curve shows the ongoing energy consumption. Assume on a day D the system/user has already an Accumulated Energy Consumption (AEC) of a given amount (Figure 1.2). Assume further, that there is a pending job J, with 360 compute node count request. In order to understand whether the job J can still be scheduled for execution within the available energy budget, the RMSS of the target system has to have the information on the potential energy consumption of the job J with 360 compute nodes. Also this knowledge cannot be beforehand provided by any of currently existing RMSS's.

Though power and energy capping (as of described for above use case scenarios) solve different problems, ***they both require the same knowledge of, potentially unknown, power and energy consumption related data for the large-scale applications to be executed.*** Without the access to this knowledge, the implementation of these techniques will be incomplete.

This dissertation will present:

- a monitoring toolset that will be used for collecting, measuring, estimating, and correlating data regarding the energy/power flows in the data center. This toolset will also be used for verification of the optimization methods aimed at energy/power consumption reduction as well as for calculating certain HPC data center efficiency relevant Key Performance Indicators (KPIs);

- a metric for calculating the aggregated energy consumption of large-scale applications;
- models (and their corresponding implementations) allowing for prediction of the execution time, power and energy consumptions of large-scale applications for different to-be-utilized resource configurations;
- the logistics for implementing power and energy capping; and
- a framework for automatic selection of energy-wise optimal resource configuration for large-scale applications under specified execution time and average power consumption constraints

thus covering the mentioned prerequisite for implementing efficient energy and power capping. Section 1.5 provides the detailed outline of this dissertation.

1.2. Framework for Energy Efficiency Management: Requirements, Pillars, and Key Performance Indicators

A general basis for building frameworks aimed towards energy efficiency management of HPC systems was suggested in [17]. Figure 1.3 illustrates the main four aspects, referred to as pillars [17], that play a major role in overall data center energy efficiency improvements.

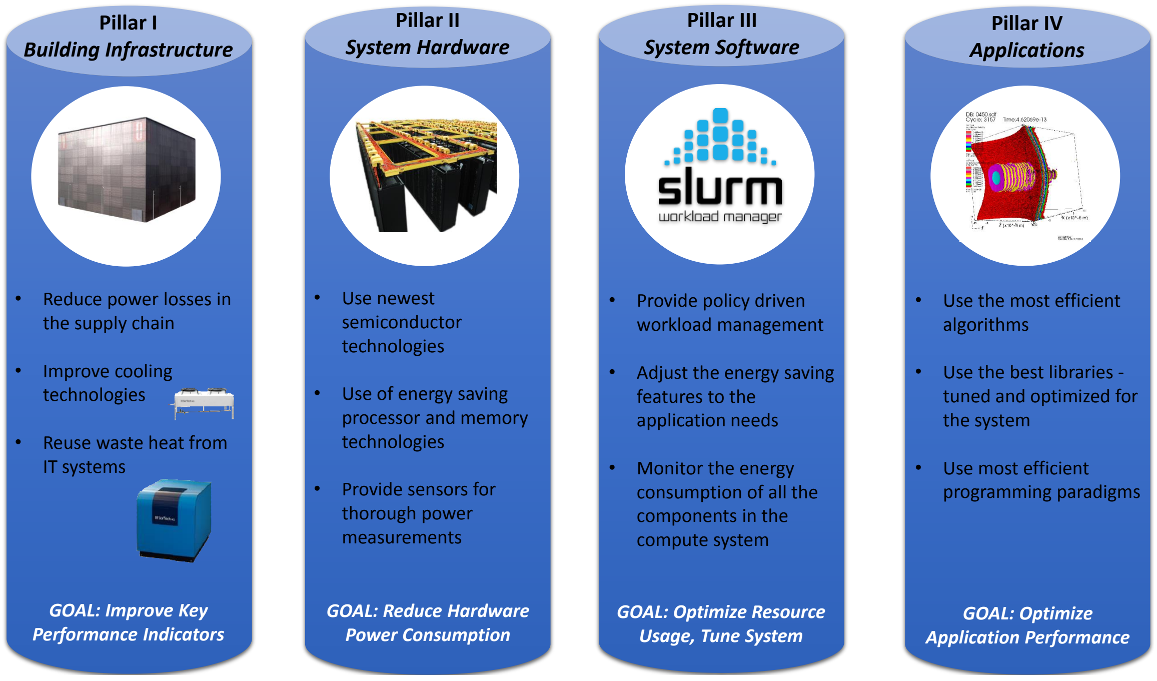

Pillar I
Building Infrastructure
• Reduce power losses in the supply chain
• Improve cooling technologies
• Reuse waste heat from IT systems
GOAL: Improve Key Performance Indicators
Pillar II
System Hardware
• Use newest semiconductor technologies
• Use of energy saving processor and memory technologies
• Provide sensors for thorough power measurements
GOAL: Reduce Hardware Power Consumption
Pillar III
System Software
slurm
workload manager
• Provide policy driven workload management
• Adjust the energy saving features to the application needs
• Monitor the energy consumption of all the components in the compute system
GOAL: Optimize Resource Usage, Tune System
Pillar IV
Applications
• Use the most efficient algorithms
• Use the best libraries - tuned and optimized for the system
• Use most efficient programming paradigms
GOAL: Optimize Application Performance

Figure 1.3.: The 4 pillar framework

The identified four pillars are:

I **Building Infrastructure**
Representing the complete non-IT infrastructure (including reuse technologies) required for operating a data center. This pillar is intended to support improvements in power, cooling, and heat reuse infrastructures. These improvements can then be verified using various KPIs (defined by the standardization bodies as American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE) [18] and The Green Grid Association [19]) such as Power Usage Effectiveness (PUE), Water Usage Effectiveness (WUE), Carbon Usage Effectiveness (CUE), Energy Reuse Effectiveness (ERE), etc. The currently existing building infrastructure control and management tools [20, 21] used in data centers do not account for the dynamic operational behavior of the deployed HPC systems. Therefore the building infrastructures are typically optimized only for the operation at maximum power draws [17] of deployed supercomputing systems, and do not consider the variability of their power/heat dissipation on the installed cooling infrastructures, which in general might lead to inefficiencies.

II **System Hardware**
Representing the complete IT infrastructure (including networks and storage systems) that is installed in a data center. The goal of this pillar is to assist the reduction of the power consumption drawn by system hardware. While, the data center operator has a little or no influence on the system, since it is constrained by the vendor product availability, more and more hardware vendors see the energy efficiency of their products as an important feature. For example, the DVFS (as described in Section 1.1) is used by the PCU of the Intel's Sandy Bridge processors to automatically perform the power/performance control [14] for the running application.

III **System Software**
Representing the complete system level software stack required for operating the system hardware. The goal of this pillar is to support efficient management of the system resources and to minimize the amortized costs. This can be done through the RMSS of the target high-end system in conjunction with the data center specified policies (e.g. system power consumption constraint, available energy budget, etc.), the existing energy saving features of the target hardware platform (e.g. DVFS, Dynamic Power Management (DPM) [22], etc.), and application specifics (e.g. compute bound, memory bound, etc.). It is equally important to provide a lightweight support to the target RMSS's for beforehand estimation of the application energy/power consumption related data in order to assist energy-efficient resource management and scheduling decisions. Currently, none of the existing RMSS's possesses that feature.

IV **Applications**
Representing the various applications of users. The main goal of this pillar is to support the optimization processes of the application performance (also with regard to the target hardware platform specifics). As it is shown in [23], the improvement of the application performance will in most cases lead to savings in the energy consumption. This can be achieved through the usage of target platform specific efficient algo-

rithms and libraries, programming paradigms, etc. However, as will be seen in Subsection 4.2.1, the "artificially" achieved performance gains only through the adjustments of some target hardware platform parameters, will not always lead to energy savings.

This framework clearly shows that the data from all the four pillars needs to be collected and correlated in order to evaluate the current data center energy-efficiency, to quantify the overall energy/power costs, to further explore the cross-pillar interdependencies, and to assess the outcome of various (cross-pillar) energy-reduction aimed optimization solutions. This information will also allow to measure the amount of energy/power consumed by specific large-scale applications. The latter knowledge in its turn will allow for energy-driven charging policies as an alternative to currently existing CPU-hour based charging policies. Within this context, it is equally important to provide tools that will allow various HPC users to estimate beforehand the possible costs when running different applications.

Figure 1.4 illustrates a use case, showing that the mentioned integral view of data center energy/power flows is equally important for energy and power capping problem domains. Assume a user, who has a currently available energy budget of E kWh, wants to obtain the results from an application X in T minutes using n compute nodes of the target HPC system. Assume further, that there is a cap (limit) on the average power draw that a system can consume.

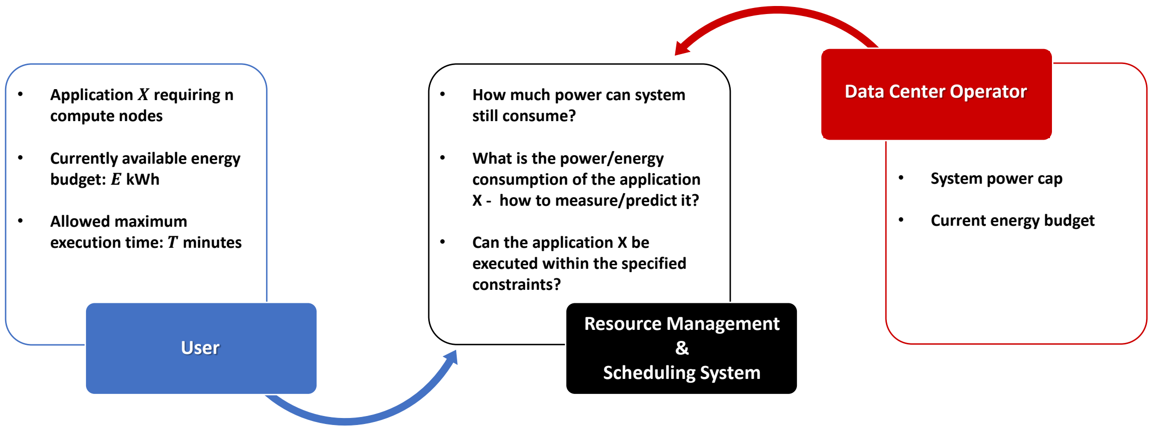

Application X requiring n compute nodes
Currently available energy budget: E kWh
Allowed maximum execution time: T minutes
User
How much power can system still consume?
What is the power/energy consumption of the application X - how to measure/predict it?
Can the application X be executed within the specified constraints?
Resource Management & Scheduling System
Data Center Operator
System power cap
Current energy budget

Figure 1.4.: Overview of the power and energy capping problems

In order to understand whether it is possible to schedule the application within the specified constraints the information on currently available power budget and at least on two of the following metrics is required (these two metrics can then be used to derive the third) by the RMSS of the target system:

- **Time-to-Solution (TtS)**
  Indicating the execution time of a given application
- **Energy-to-Solution (EtS)**
  Indicating the aggregated energy consumption of a given application

- **Average Power Consumption (APC)**
  Indicating the mean power draw of a given application

These three metrics are formally defined as follows. The EtS *energy* consumed by the application over some $[t_{start}; t_{end}]$ time interval (meaning that TtS $= t_{end} - t_{start}$), is defined as [24]:

$$\text{EtS} = \int_{t_{start}}^{t_{end}} P(t)dt \tag{1.1}$$

where $P(t)$ is the application *instantaneous power* consumption. The APC *average power* over this integral is given by the following equation:

$$\text{APC} = \frac{\text{EtS}}{\text{TtS}} = \frac{1}{\text{TtS}} \int_{t_{start}}^{t_{end}} P(t)dt \tag{1.2}$$

Section 2.6 presents the detailed description on EtS calculation for a given HPC application. Currently, none of the existing RMSS can beforehand provide the information on these three TtS, APC, and EtS metrics for an application with a given compute node count request.

This dissertation will present an ECMA framework that covers all these requirements and answers to all the questions presented in Figure 1.4. Moreover, ECMA framework, for a given application, will advise on an energy-wise optimal **Resource Configuration (RC)** (*defined as a triple **(a,b,c)** where **a** indicates the number of compute resources, **b** indicates the maximum CPU frequency of the compute resources, and **c** indicates the set of **a** compute resources*) that will adhere to the predefined TtS and APC constraints.

1.3. Background and State of the Art

Several institutions and societies are persistently addressing the issues regarding the efficient energy management of HPC data centers. This section will in detail describe the related studies aimed towards energy and/or power capping support - will present the state of the art frameworks and tools.

1.3.1. Frameworks Aimed Towards Power Capping

Modern Intel CPUs provide a hardware based power capping feature [25, 14] which as stated by Rountree et al. [26] trades power variation for performance variation. At the same time, different HPC applications have different performance, implying that the purely hardware based power caps will make the efficient utilization of large scale HPC systems difficult since in most of the scenarios the hardware limit will "waste" the available system power. For example, if only $1/2$ of a system is utilized, the power cap for some of the compute resources could be higher allowing for a better performance of certain applications. Hence, the power cap must be enforced at the system level and not identically for each compute resource, and has to allow for a dynamic control. Software defined power capping might be a better approach here since it would allow for dynamic adjustments of system power consumption based on the current resource usage and real application consumption. In addition, some of the applications might not use the available power, giving an opportunity of power cap increase for others. A similar investigation aimed towards power capping support is found in [27]. It proposes an adaptive configuration selection process for power-constrained heterogeneous systems. This approach is validated only for a single heterogeneous processor and there is no validation of the suggested process against multi-node compute systems. Thus this process does not consider the node power variation across the system. Moreover, the proposed approach is aimed at maximizing application performance without considering the tradeoff in aggregated energy consumption, and thus the discussed, in this dissertation, energy-wise optimal resource configuration estimation problem.

Another possible solution for implementing power capping could be the usage of energy aware scheduling. As it is shown in [28] it is possible to find a global maximum operating CPU frequency of all the cores of the compute nodes of a given application that will lead to a minimum EtS consumption value. This approach usually results in a tradeoff between resource power consumption and application execution time and it does not focus on resource power consumption. Additionally, as will be shown in Section 4.3, the suggested (in [28]) prediction method is not able to predict the APC, EtS, and TtS of a given application for unknown resource configurations, and due to that, it cannot be used for power and/or energy capping.

An investigation of Sarood et al. [29] considers a software-based resource management system for a dynamic change of the resource configuration for a running job. The study claims that this leads to an improved application throughput within a given power budget. While useful, this approach relies on the dynamic resource configuration modification feature of the underlying applications which is not supported by the majority of current HPC applications mainly due to the existing limitations in the used communication libraries [30]. The suggested in [29] approach also does not consider the power diversity among the compute nodes (see Section 3.5 and Section 6.3) of a given homogeneous high-end system. This inconsideration leads to possible inefficiencies, since due to the diversity in compute resource requirements of different applications, and the differences in application energy and power consumption profiles, it might be possible to schedule more applications at a given point in time within given application-specific energy, power consumption, and/or execution time constraints via compute node pre-selection methods. Another study by Sarood et al. found in [31] is aimed towards finding an optimal tuple of compute node count, operating CPU, and memory power levels that will minimize the application execution time under a strict power budget. The study proposes an application profile based model, that uses interpolation methods for estimating the application execution time for different compute node counts, CPU levels, and memory levels. As stated by the authors, the presented execution time and average power consumption prediction models require a relatively high number of available application-relevant profile data and, at the same time, the models need to be trained on a selected set of configurations that span the entire range of the available resource configurations. In other words, the suggested in [31] models would fail to predict the application execution time or average power consumption for higher than the profiled compute node counts. Additionally, the study in [31] does not consider the power variation among the homogeneous system compute nodes, which also, as will be shown in Section 6.3, varies at different maximum operating CPU frequencies and is an important consideration point.

Another set of studies aimed towards power capping is found in [9, 10, 11], which employ methods for dynamic, automatic and transparent adaptation of voltage and frequency of the underlying application-utilized compute resources for power reductions. Two important topics are not considered there: *(i)* the maximum allowed operating frequency of the application compute nodes and *(ii)* the specific compute node set, which together, will satisfy the predefined execution time and average power consumption constraints while yielding in minimal energy consumption value. The proposed, in Chapter 6, CA plug-in complements the methods of [9, 10, 11] by efficiently providing advice on the mentioned topics.

1.3.2. Prediction Tools That Could Have Been Used For Power and Energy Capping

A method for application performance prediction is described by Ipek et al. in [32]. The authors introduce an adaptive model for predicting the TtS of parallel applications with respect to the input problem size of the application for a fixed compute node count. Although, it could be argued that it is possible to derive the energy consumption of an application using the corresponding knowledge of the TtS value and the vendor provided maximum TDP value [12] of a system compute node, this method will not be applicable for the use cases of power and energy capping, as of identified in this dissertation, since it does not provide the knowledge on TtS behavior with respect to different compute node counts.

The problem of cross platform energy consumption estimation of individual applications is investigated by Tsafack Chetsa et al. in [33]. The study suggests a model for predicting the energy consumption of a given application during the application's execution phase. This model is not applicable for implementing power and/or energy capping techniques since it does not provide the knowledge on application power/energy consumption data in advance, which is a basic RMSS requirement for correct application scheduling within the specified power and/or energy consumption constraints.

Another set of approaches focused on predicting the energy consumption of applications is suggested by Hager et al. in [34] and by Brochard et al. in [35]. The presented methods use analytical models for estimating the power consumption of a given application with respect to a given CPU frequency. Both models require knowledge of either the application (e.g. scaling properties) and/or the platform characteristics for different CPU frequencies. They both are not yet extended/validated for multi-node compute systems and are analytic predictive models, often requiring additional manual tunings [32] for target high-end system specifics and peculiarities, which in their turn restrict the usage of these methods.

Another method aimed at controlling power consumption, and referred to as "Pack & Cap", is presented in [36]. This method adaptively manages the number of cores and CPU frequency depending on the given application characteristics, in order to meet the user-defined power constraints. The described "Pack & Cap" technique is validated on a single, quad core server node, and, as authors mention, it is not yet extended/validated for large scale computing systems, implying its inapplicability for the HPC domain. In addition, the method requires a large volume of application performance data to conduct power/energy capping, which could not be available in the real-world scenarios. Moreover, "Pack & Cap" does not predict the power/energy consumption of applications with respect to different compute node counts. Lastly, it is important to mention that the suggested method is specifically targeted for virtual machines, implying possible difficulties when adapting it for HPC systems.

The next set of works focused on application power/energy consumption estimation, given the in-depth characteristics of the underlying application, is described by Olschanowsky et al. in [37] and by Song et al. in [38]. Olschanowsky et al. present an energy consumption prediction model requiring application tracing (information on floating point operation count, memory operation count, etc.) and information on the energy profile of the target high-end system (e.g. average energy cost per fundamental operation, etc.), obtained through the use of several special benchmarks. The suggested in [37] model could be used for a cross platform application energy consumption prediction, if the required energy profile data (e.g. achievable memory bandwidths for each level of the memory hierarchy) of the target system is available. At the same time this method involves application code instrumentation and attempts to split the application into "basic blocks" [37]. This would require a lot of effort, especially when dealing with several hundred different large-scale applications, which is typically the case for modern HPC data centers. The quasi-analytical model suggested by Song et al., combines the application analytic description (achieved through extensive application analysis) with the compute platform parameters (such as per-core power consumption of a computation unit and power consumption during inter-processor communication) obtained via experimental benchmarks [38]. The validation of the model was shown using a single benchmark and the suggested in [38] method requires a thorough analysis of the given application, which could be impractical in real-world scenarios, when myriad of applications with different characteristics are queued for execution.

To summarize, none of the above discussed methods predicts the application execution time, power, and energy consumption values with respect to the number of compute nodes, and thus none of them can be applied for implementing power and energy capping techniques for the use cases discussed in this dissertation.

As to the available power and energy consumption monitoring solutions, then Section 2.3 provides in detail discussion on the existing Data Center Infrastructure Management (DCIM) tools.

1.4. Contributions of This Work

The main contribution of this dissertation is justification and development of **Energy Consumption Management Adviser (ECMA)** framework comprising of the following components:

- **a unified energy and power monitoring toolset**
  The toolset is aimed towards collection and correlation of energy and power consumption relevant data from all the aspects of a given data center, stretching from the building infrastructure over the deployed HPC systems and heat reuse technologies to the system software stack and actual (in-execution or to-be-in-execution state) large-scale applications. The toolset will allow for dynamic and static assessment of various (customized) KPIs and metrics that will help in identification of the present inefficiencies, verifications of the applied optimizations, as well as allow for further

tailored data analysis. The toolset can further assist to the agnostic decision making (done by the data center operational energy and power control point) based on the provided wholistic view on the current energy/power flows of the complete supercomputing site.

- **a model and an implementation allowing for beforehand prediction of application TtS, APC, and EtS metrics with respect to a given resource configuration**
  The model is application independent but provides an application specific results. It does not require any knowledge on application internals and target HPC system. The only required knowledge is per application available monitored history data on TtS, APC, and EtS profiles. Due to that, the model covers all the above mentioned prerequisites for implementing energy and power capping techniques. The model, as well as the monitoring toolset, can be further used for increasing the user awareness regarding the potential power/energy consumptions of applications when run with different resource configurations.

- **a mechanism for detecting the dissimilar power consumption behavior of homogeneous HPC system compute nodes and its possible use cases in real-world scenarios**
  It is shown that the power consumption profile of the compute nodes of homogeneous HPC systems varies under the same workload. Moreover, it is shown that this distribution of the average power consumption of the compute nodes changes with the operating maximum CPU frequency of the compute node cores. It is further demonstrated that simple pre-selection methods of compute nodes (based on the observed power variation among system compute nodes) can lead to significant energy and power savings - in the considered case of 512 compute nodes, at least 4.7%.

- **a RMSS plug-in for energy-wise optimal resource configuration selection**
  An efficient algorithm for selecting operating frequency and also the set of available compute nodes which would minimize the aggregated energy consumption (i.e. EtS) of a given application preserving the predefined application execution time and average system power consumption constraints is presented - the algorithm takes advantage of the existing average power consumption variability among the compute nodes of the target homogeneous system. Based on this algorithm, a plug-in is proposed which acts as an energy-optimal resource configuration adviser for a given RMSS. The presented plug-in is in complement to the existing RMSS energy/power saving techniques and can be used in conjunction with any other energy/power reduction efforts.

Figure 1.5 illustrates the specification of the suggested ECMA framework. The ***Energy/Power Monitoring Solution*** should ensure the collection of energy/power consumption relevant data. Based on the collected data, the monitoring toolkit dynamically evaluates all of the interest metrics and KPIs, including the TtS, APC, and EtS profiles of executed applications. The latter data is then used by the predictor for determining the application TtS, APC, and EtS behavior for a given RC. Once a new measured data on application TtS, APC, and EtS profiles is available, the predictor is calibrated for the increase of the prediction accuracy.

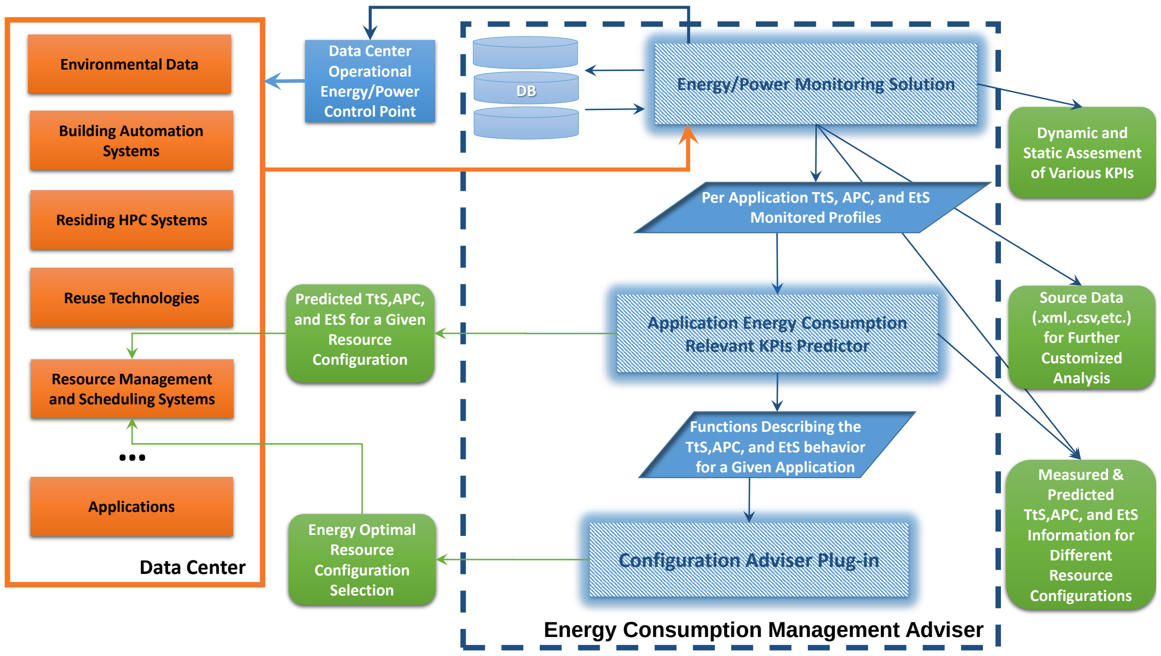

Environmental Data
Building Automation Systems
Residing HPC Systems
Reuse Technologies
Resource Management and Scheduling Systems
•••
Applications
Data Center
Data Center Operational Energy/Power Control Point
DB
Energy/Power Monitoring Solution
Dynamic and Static Assesment of Various KPIs
Per Application TtS, APC, and EtS Monitored Profiles
Predicted TtS,APC, and EtS for a Given Resource Configuration
Application Energy Consumption Relevant KPIs Predictor
Source Data (.xml,.csv,etc.) for Further Customized Analysis
Functions Describing the TtS,APC, and EtS behavior for a Given Application
Measured & Predicted TtS,APC, and EtS Information for Different Resource Configurations
Energy Optimal Resource Configuration Selection
Configuration Adviser Plug-in
Energy Consumption Management Adviser

Figure 1.5.: Infrastructure Specification of the Energy Consumption Management Adviser

The functions describing the application TtS, APC, and EtS (determined by the predictor) are then used by the CA RMSS plug-in for estimating the energy-wise optimal RC which will adhere to the specified application maximum TtS and APC constraints. This estimated RC is then used by the target RMSS for dispatching the application execution.

1.5. Outline of This Work

The reminder of this dissertation is organized in three parts. Having discussed the requirements for energy efficiency management and the current state of the art in Part I, Part II outlines the need for an integrated (wholistic) evaluation of the HPC data centers, without which it would be non-trivial to verify and improve the existing and future power/energy-reduction aimed solutions. This required integrated view includes all the aspects of the HPC data center, from environmental information over the site infrastructure and the residing IT systems to resource management systems and large-scale HPC applications.

Chapter 2 introduces the Power Data Aggregation Monitor (PowerDAM) tool, used for collecting, measuring, estimating, and correlating energy/power consumption relevant data from the mentioned aspects of the HPC data center.

Part III discusses the prerequisites that have to be met in order to implement the energy and/or power capping. It is organized in the following three chapters:

Chapter 3 examines the scaling behaviors of the parallel applications, discusses the corresponding studies. The chapter then introduces an adaptive model, referred to as

Adaptive Energy and Power Consumption Predictor (AEPCP) model, for predicting the energy, power, and execution time of HPC applications for a given number of compute resources. This chapter also outlines the model as a necessary milestone towards the energy and/or power capping implementation.

Chapter 4 extends the AEPCP process to enable the development of analytical models for estimating application execution time, power, and energy consumptions as functions of the number of compute nodes and maximum operating CPU frequency. Based on these analytical models, an adaptive model, referred to as Lightweight Adaptive Consumption Predictor (LACP), is presented that implements the extended prediction process.

Chapter 5 describes the LACP model usage for quantifying energy-power-time tradeoffs for real world HPC applications, which is currently not possible with the usage of any resource management and scheduling systems.

Part IV presents a unified framework for software defined power capping. This part is logically structured as follows:

Chapter 6 introduces an efficient algorithm and a corresponding scheduler plug-in allowing for software based power capping within the user specified application execution time restrictions. This chapter compiles all the achieved milestones into a framework, refereed to as ECMA and describes its reference flow. The chapter outlines the application range of the developed ECMA framework in the real-world situations for enhancing the existing energy efficient resource management, scheduling, and backfilling solutions in modern HPC data centers.

Chapter 7 draws the summary of the main accomplished contributions, delineates the future development directions, and concludes the dissertation.

Part II.

Evaluating The Energy Efficiency of a HPC Data Center

"If someone separated the art of counting and measuring and weighing from all the other arts, what was left of each (of the others) would be, so to speak, insignificant."

PLATO

2. Monitoring the Energy and Power Consumptions in HPC Data Centers☆

2.1. Preface

Having set the approaches aimed towards power and energy consumption reduction as important milestones towards economically well-balanced, sustainable, and environmentally friendly operation of a data center, this chapter will look into the requirements for monitoring the power and energy consumption flows within the HPC data center and for evaluating various power/energy consumption optimization aimed techniques.

There are several available monitoring and evaluation tools that can be used for various individual components of a data center such as building automation systems', IT systems', applications' performance monitoring tools, etc. One of the main characteristics that unites these highly diverse tools, with each having its own fortes, is their scope-based applicability and their isolation from each other, which in its turn disallows the integral assessment of the actual data center energy-efficiency and forbids the complete verification of applied optimization solutions. This chapter will present a first step towards a unified energy efficiency evaluation toolset, referred to as Power Data Aggregation Monitor (PowerDAM), which enables these lacking functionalities by acting as a hub among various data center specific monitoring tools and thus covering a wide range of power and energy consumption analysis capabilities ranging *from* the complete data center view *over* the deployed IT systems and waste heat reuse technologies *to* the system software stack and large-scale user applications. The subsequent section will show that without this integrated view, any improvements aimed towards energy consumption reductions would be incomplete.

2.2. Need for an Integrated View

Current building infrastructure management systems, present in modern data centers, control and monitor only infrastructure related components [20, 21]. These infrastructure control and management systems are completely isolated from the behavior of the target supercomputing systems residing in the data center and do not provide a control-side coupling between building infrastructure and HPC system management. Therefore the building infrastructure is typically optimized for the operation at maximum power draws of deployed IT systems.

Figure 2.1 shows the complete facility power consumption profile of the Leibniz Supercomputing Centre (LRZ) data center (Complete Facility Power Profile), the latest HPC

☆This chapter is partly based on the following previous work of the author: *Hayk Shoukourian et al. Monitoring Power Data: A first step towards a unified energy efficiency evaluation toolset for HPC data centers. Journal of Environmental Modelling & Software, Elsevier, Vol* 56*, pages* 13 − 26, 2013.

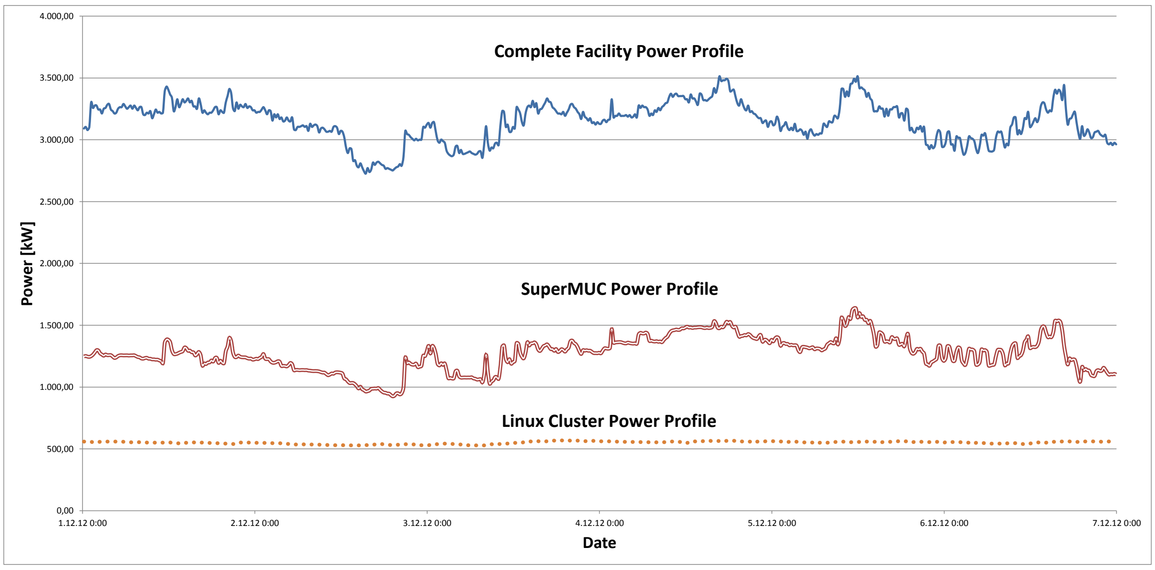


Figure 2.1.: Power consumption profiles of LRZ for a period of 8 days

system (SuperMUC[1] Power Profile), and the Linux cluster (Linux Cluster Power Profile) for a period of 8 days. Despite the almost constant power consumption behavior of the Linux cluster[2], which is similar to the older generation of HPC systems deployed at LRZ, SuperMUC shows a variation of up to 800 kW and dominates the overall power profile of LRZ.

A maximum power consumption of 3.4 MW [1] was observed on SuperMUC during the HPL [6] benchmark. This further means that it is possible to have around 2.6 MW variation (Figure 2.1) in the power draw of SuperMUC during the normal operation modes. One consequence of the shown highly dynamic power profile is a strong fluctuation of the data center infrastructure efficiency. Only by collecting and correlating the infrastructure data with HPC system data is it possible to detect and resolve the inefficiencies present in the installation. This process will provide an integrated view across multiple sources and structures of the supercomputing site and will allow a better understanding of the complete data center power profile. This understanding in its turn will lead to better optimization techniques which are necessary for improving the overall energy efficiency and thus reducing the data center TCO. According to a study on principles of energy efficiency in HPC [39], despite the presence of necessary individual components, there is a lack of a unified energy measuring and evaluation toolset which is capable of monitoring and analyzing the energy consumption of a supercomputing site in an integrated (wholistic) way, combining the residing HPC systems with data from the cooling, building infrastructure, and the large-scale applications. PowerDAM, presented in Section 2.4, is an example of such a unified energy measuring and evaluating toolset.

[1]Briefly described in Section 2.5.

[2]LRZ Linux Cluster consists of several segments with different types of compute nodes, interconnect and sizes of shared memory [5].

2.3. Related Works

A set of related works on energy measuring tools is found in [40] and [41]. The first study presents a framework for energy efficient management of large scale distributed systems. The presented approach does not consider the infrastructure relevant data. In the second study, a measurement evaluation tool is presented as a combination of software components and additional external hardware components (e.g. digital meters). This implies that the software can only be used with the custom hardware. For that reason, this approach cannot be applied to highly integrated large-scale systems since it would require a complete instrumentation of an entire supercomputer which are already starting to have more than 10000 compute nodes. That is the main reason that more and more vendors aim to provide the measuring infrastructure inside the actual system.

Another set of related studies found in [42, 43] are specifically aimed at estimating the application energy and power consumption (without accounting for data center infrastructure) using performance counter based models (in case no watt meters are present in system installation). While the developed PowerDAM toolset can also be used for monitoring hardware performance counters, the main disadvantages of this approach are: *(i)* the measurements do not reflect the complete power consumption of a compute node (but only CPU and possibly memory); *(ii)* the measurements have to be done in-band, i.e. during the application execution, thus possibly causing some performance degradations; and *(iii)* power/energy estimation models have to be constructed for each hardware generation. Another approach aimed at software-based application energy/power estimation methods is found in [44]. It proposes estimation models for clusters for which it is possible to identify a relatively constant set of applications. The models were constructed for individual applications using performance counters. While the suggested approach could be used as an alternative to Power Distribution Units (PDUs) (in case the later ones are not available due to their high purchasing costs), it is worth noting that in the real-world scenarios the applicability of these models would be non-trivial, since it would require *(i)* an additional application identification phase, and *(ii)* additional analysis of each, out of reference, application and corresponding model tunings, which could be impractical, when several applications with different characteristics are queued for execution. Figure 2.2 shows that this is the very case for the LRZ HPC application stack.

Another set of related tools in the area of operational management are Rackwise [45] and openDCIM [46]. These tools are examples of DCIM tools which were developed to monitor and analyze the physical assets and resources within a data center. However, they do not consider the system software side and do not provide complete information on energy consumption of all necessary components that are present in the compute system. For example, without the information on power and/or energy consumption of running applications the complete exploitation of the energy saving features of the target platform would be relatively complicated. This information will allow for the further understanding and tuning of the application internally (through the change of algorithms, memory access patterns, etc.) as well as externally via hardware adjustments (e.g. static/dynamic voltage and frequency scaling, etc.).

It can be further argued that PowerDAM could have been implemented on top of these DCIM tools as an extension (plug-in) in order to supplement the lacking functionalities. In this case, the usage of PowerDAM would have been limited to the data centers that use a

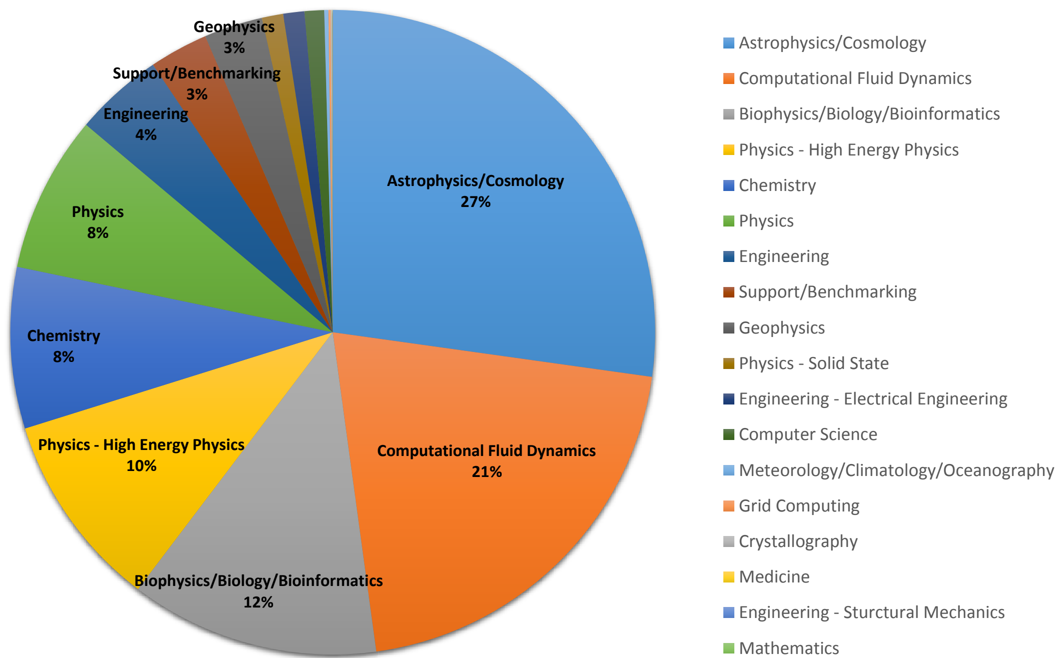


Figure 2.2.: Core hour usage of different HPC applications at LRZ

certain DCIM contradicting its purpose. The main aim of PowerDAM is not to mimic any of the existing data center monitoring tools, but to act as a hub among several monitoring tools (e.g. infrastructure, IT systems hardware/software, applications, etc.)[3], allowing for data collection and correlation from all HPC data center energy consumption-relevant components, and thus providing an integrated evaluation of energy costs for a supercomputing site. Without this data the improvement of the data center energy efficiency would not be trivial and would also be incomplete.

When summarizing, it can be stated that within the problem domain of power/energy efficiency the above mentioned approaches are scope based, as they pose a specific question and develop their solutions for answering that specific question. Whereas, PowerDAM is a generic tool, which gathers point-of-interest data from different sources in order to answer any questions that may arise within the problem domain with little to any further development.

2.4. Power Data Aggregation Monitor (PowerDAM)

Power Data Aggregation Monitor (PowerDAM) [47, 48, 49] is a unified energy measuring and evaluating toolset for HPC data centers. It is aimed towards collecting and correlating energy/power consumption-relevant data from different aspects of the HPC data center (as identified in Figure 1.3) thus covering a wide range of power consumption analysis capabilities from:

[3]Each of these aspects can have different monitoring and controlling systems - LRZ is an example of such installation, where each of the data center aspects (i.e. building infrastructure, HPC systems, etc.) has different monitoring and management tools.

- **the complete data center view** over
  Ability to asses the current status of data center KPIs [19], such as PUE, ERE, WUE, DWPE [50], etc.

- **the individual HPC systems** to
  Ability to report on current system utilization rate, power consumption as well as analyse the system comprising IT components such as compute nodes, networks, etc.

- **the large-scale user applications.**
  Ability to report on application Energy-to-Solution (EtS), Average Power Consumption (APC), etc.

Figure 2.3 shows the overview of PowerDAM. It uses an agent-based data communication model for actual sensor data retrieval from the monitored entities/systems. These agents reside on the monitored entity side and are configured to have a permission for accessing and pushing the requested sensor data over the network. PowerDAM in its turn maintains plug-ins for communicating with these system side agents. This approach makes PowerDAM loosely coupled with the monitored entities and thus allows for relatively easy extension of the monitored entity set.

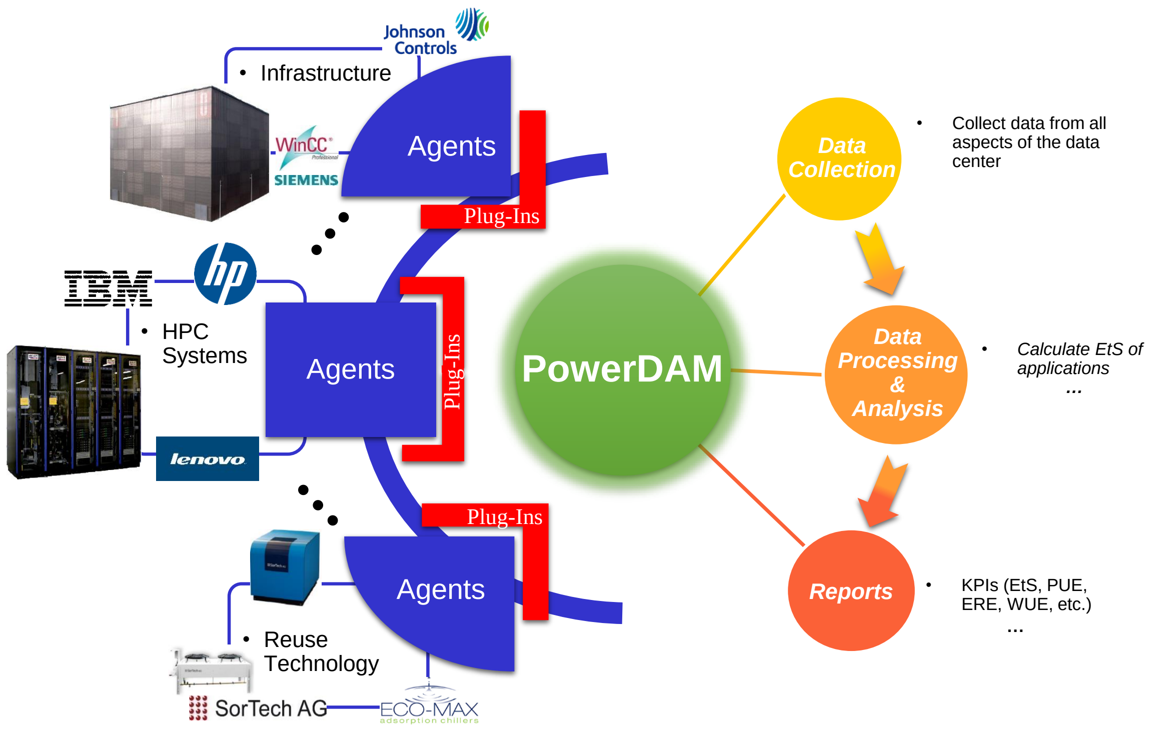

Johnson Controls
• Infrastructure
WinCC
SIEMENS
Agents
Plug-Ins
IBM
hp
• HPC Systems
Agents
Plug-Ins
lenovo
Plug-Ins
Agents
• Reuse Technology
SorTech AG
ECO-MAX adsorption chillers
PowerDAM
Data Collection
• Collect data from all aspects of the data center
Data Processing & Analysis
• Calculate EtS of applications ...
Reports
• KPIs (EtS, PUE, ERE, WUE, etc.) ...

Figure 2.3.: PowerDAM overview

The main design goals of PowerDAM were:

- **independence of monitored entities**;
  Different data centers are comprised from different HPC systems manufactured by various

hardware vendors (e.g. IBM, HP, Cray, etc.), use different reuse technologies from distinct producers (e.g. SorTech AG, ECO-MAX, etc.), and use different vendor-specific building automation systems (e.g. Siemens, Johnson Controls, Mitsubishi Electric, etc.). Thus the way sensor data (e.g. temperature, power, etc.) is accessed can vary from entity to entity. PowerDAM is not tied to a specific entity and, therefore vendor.

- **easy extensibility of monitored entity set**;
  Because of variety of components present in the data center, which could be potentially a point of monitoring interest, an easy integration of new entities is required.
- **ability to access the energy/power consumption of the applications**; and
  A simple and easy to understand metric is necessary in order to increase the user awareness of application energy/power consumption. EtS and APC are examples of two such simple metrics.
- **ability to correlate the collected data and to access the status of various metrics and KPIs**
  The collected data should be easily correlated in order to better understand the interactions between different components of the data center for identifying the improvement areas, as well as for assessing and verifying the applied optimization solutions.

2.4.1. Framework

PowerDAM is developed in Python [51] and provides a plug-in framework for defining the monitored entities. PowerDAM provides two plug-in interfaces for each monitored entities: one for sensor data collection and one for collecting application relevant data from Resource Management and Scheduling System (RMSS) of the target entity. The implementations of these interfaces define how the agent based data collections are managed. A monitored entity is free to define only one of these interfaces. The purpose of this optionality is to allow PowerDAM to monitor entities that do not represent a HPC system (e.g. building automation system of the data center, power management system, etc.) and thus do not account for RMSS data, or the HPC systems that are only of interest from RMSS data perspective and do not account for sensor data.

For agent based communication PowerDAM has well-defined Application Programming Interfaces (APIs). For instance, the API syntax for the sensor data collection is presented in 2.1:

$$RootResourceIdentifier(.ResourceIdentifier)^{*}\_\_SensorType=Value;Timestamp \tag{2.1}$$

where

- *RootResourceIdentifier* represents the monitored entity
- *ResourceIdentifier* represents the monitored component of the entity (e.g. in case of a HPC system as a monitored entity, both a rack and a compute node can be considered as a *Resource*). The * indicates that the expression enclosed in parentheses (i.e. .ResourceIdentifier) can be repeated (appended to the former one) zero or more times

- *SensorType* represents the type of the *Resource* sensor (e.g. power, load, temperature, etc.). For example, load (or utilization rate) of a given compute node illustrates the averaged workload of physical processors residing in the compute node
- *Value* represents the actual sensor measurement
- *Timestamp* is the time and date at which the sensor measurement was obtained
- **.**, **__**, **=**, and **;** are the delimiters between the above listed identifiers and are part of the syntax. Therefore, they can't be used in any of the identifiers (e.g. *example1*__*example2* is a not valid identifier)

Due to this API syntax, PowerDAM is capable of monitoring any entity which can be represented in a hierarchical tree structure. Figure 2.4 illustrates an example of such a hierarchical structure, where each vertex represents an entity/resource and the edge indicates the relation between parent and child entities/resources.

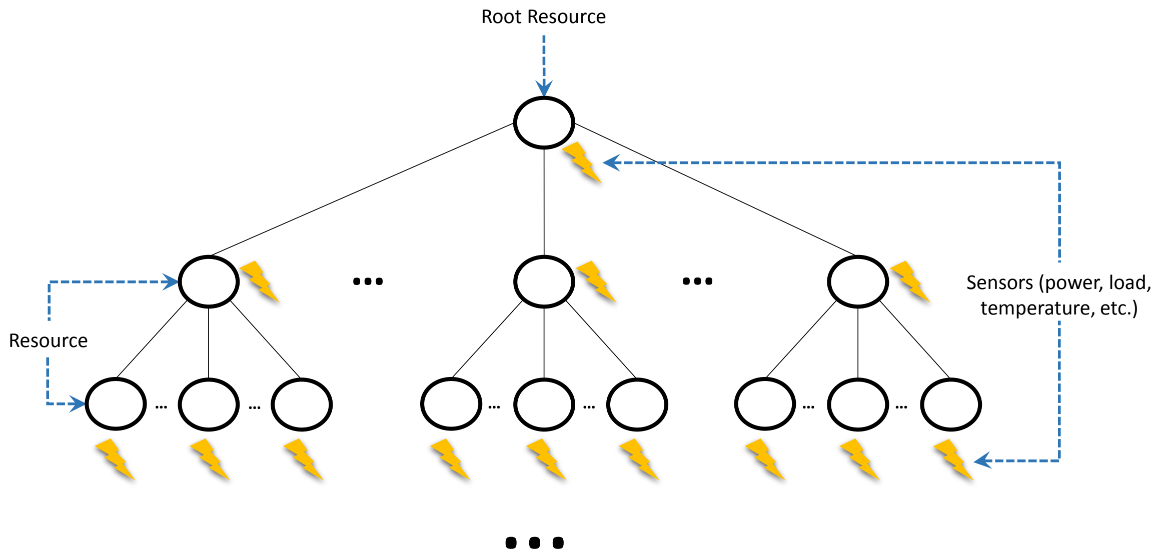


Figure 2.4.: Hierarchical tree structure

As can be seen, each resource is allowed to have a set of different sensors attached to it. The mentioned 2.1 PowerDAM API syntax allows to easily and correctly describe the set of sensors (and their respective values at a given time stamp) of a corresponding entity/resource and to preserve their hierarchical relationship. For example, the statement:

$$SystemA.RackB.NodeC\_\_power = 223;2015-03-17\ 14:31:27$$

would indicate that the power sensor of compute node *NodeC*, residing in the *rackB* rack of *SystemA* system has a value of 223 W at the time stamp value of 2015−03−17 14 : 31 : 27.

The main aim of the sensor API syntax presented in 2.1, is to support the data collection of low-level hardware sensor readings not supporting object orientation, serialization, etc. The currently developed version of PowerDAM (briefly described in Section 2.8) will in parallel support for sensor object serialization, which thus will allow for further definition-abstractions of the collected sensors (e.g. addition of physical location identifier, measurement read-out frequency, etc.). There is no restriction on the agent implementation language.

PowerDAM monitors *physical sensors* meantime allowing for custom *virtual sensor* definitions, which can represent different functional compositions of several physical/virtual sensors of different monitored entities. This option is especially useful in the case of a sensor requirement representing the functional composition of several physical sensors deployed on different entities/systems. The specification of such custom sensors cannot be done on the monitored entity-specific agent sides, since each entity-agent has only the knowledge on the sensors with which that specific entity is endowed. The definitions of various virtual sensors are carried out within PowerDAM using a dedicated and separate plug-in framework.

Since physical sensors are subject to intermittent failures or noise it is possible that some of the obtained measurement data will be invalid. PowerDAM uses linear interpolation in order to minimize the error introduced by the invalid measurements. For this reason, it maintains measurement boundaries for each monitored sensor type (specified by data center operators/PowerDAM administrators) which define the range of valid measurement values. It marks the approximated measurements in the database[4], and later, if any exists, notifies users on the replacement of source data.

2.4.2. Workflow

Assume PowerDAM is monitoring n systems (entities). These n systems are further differentiated into two categories - the ones that do provide sensor data (i.e. do have a corresponding PowerDAM sensor plug-in - assume k of n), and the ones that do provide scheduler data (i.e. do have a corresponding PowerDAM scheduler plug-in - assume q of n).

Figure 2.5 illustrates the PowerDAM workflow, which is as follows:

1. The EtS is calculated and the PowerDAM database is correspondingly updated, in parallel, for all executed applications of all currently monitored q HPC systems. Section 2.6 discusses the calculation details.

2. The current sensor data from all the k monitored systems is, in parallel, obtained and the database is correspondingly updated. Once the measurements are obtained, they are checked against the specified measurement validity boundaries (Subsection 2.4.1), and approximated, if needed.

3. The current scheduler data from all the q monitored systems is, in parallel, obtained and the database is correspondingly updated.

4. Once the physical sensor data from all the k monitored systems are obtained, the actual data for virtual sensors is calculated and the database is correspondingly updated.

5. After a configurable amount of a waiting time, this routine is repeated[5].

[4]Also saves the tracked invalid measurement values.

[5]All the $1-3$ steps are executed in parallel.

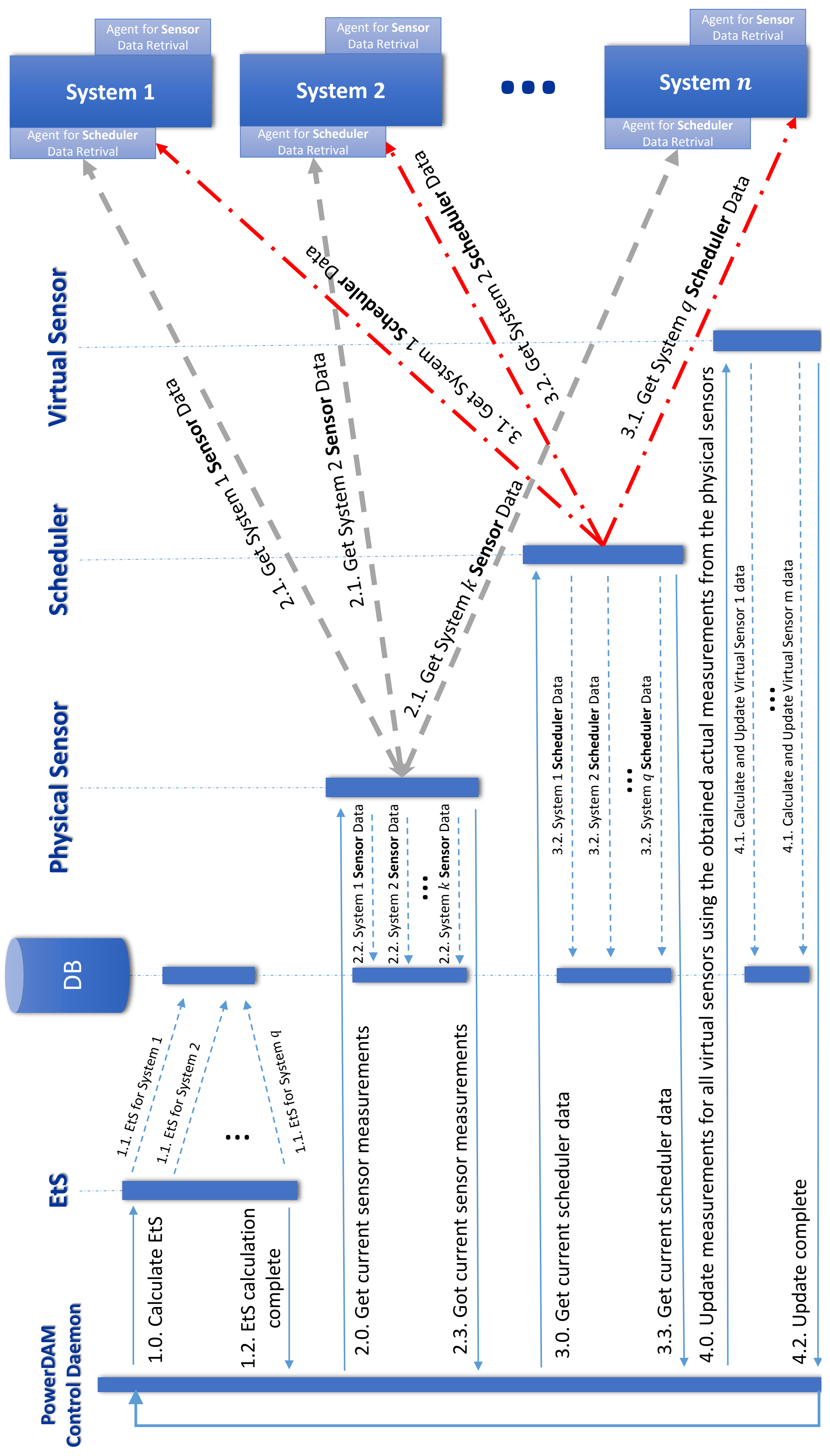

System 1
Agent for Sensor Data Retrival
Agent for Scheduler Data Retrival
System 2
Agent for Sensor Data Retrival
Agent for Scheduler Data Retrival
System n
Agent for Sensor Data Retrival
Agent for Scheduler Data Retrival
Virtual Sensor
Scheduler
Physical Sensor
DB
EtS
PowerDAM Control Daemon
1.0. Calculate EtS
1.1. EtS for System 1
1.1. EtS for System 2
1.1. EtS for System q
1.2. EtS calculation complete
2.0. Get current sensor measurements
2.1. Get System 1 Sensor Data
2.1. Get System 2 Sensor Data
2.1. Get System k Sensor Data
2.2. System 1 Sensor Data
2.2. System 2 Sensor Data
2.2. System k Sensor Data
2.3. Got current sensor measurements
3.0. Get current scheduler data
3.1. Get System 1 Scheduler Data
3.1. Get System 2 Scheduler Data
3.1. Get System q Scheduler Data
3.2. System 1 Scheduler Data
3.2. System 2 Scheduler Data
3.2. System q Scheduler Data
3.3. Get current scheduler data
4.0. Update measurements for all virtual sensors using the obtained actual measurements from the physical sensors
4.1. Calculate and Update Virtual Sensor 1 data
4.1. Calculate and Update Virtual Sensor m data
4.2. Update complete

Figure 2.5.: PowerDAM workflow

2.5. PowerDAM usage at Leibniz Supercomputing Centre (LRZ)

PowerDAM is currently monitoring three different HPC systems (all three with different hardware architectures) which are deployed at LRZ.

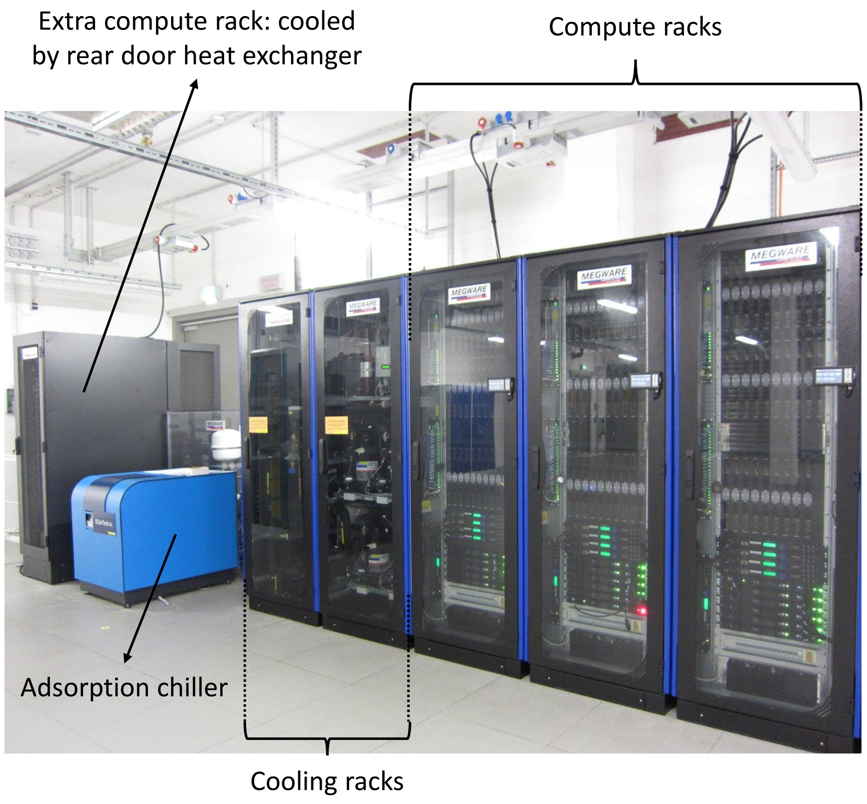


Figure 2.6.: The CoolMUC Linux cluster

The first system is CoolMUC, also referred to as Massively Parallel Processing (MPP) Linux cluster system, shown in Figure 2.6, which is a direct warm water cooled AMD processor based Linux cluster built by MEGWARE [52]. It is equipped with 178 compute nodes (8 nodes of which are dedicated to interactive sessions) with 2 sockets per node and 8 cores per socket. Each of the compute nodes has two AMD Opteron 6128 HE (Magny-Cours) 8 core processors which have three levels of cache: 128 KB $L1$ (64 KB data and 64 KB instruction cache) and 512 KB $L2$ cache per core and 12 MB $L3$ cache shared among all cores. The default operating frequency of the CPUs is 2.0 GHz. The total aggregated operating memory of the cluster is 2.8 TBytes with up to 16 GBytes per node. The main interconnect network is InfiniBand QDR using a fat tree topology. CoolMUC is connected to a SorTech [53] adsorption chiller allowing the exploration of further possibilities of waste heat reuse of the system (Figure 2.6). It allows power monitoring for nodes, internal network equipment, and internal cooling hardware. CoolMUC has closed racks and is, therefore, room neutral - no need for Computer Room Air Conditioning (CRAC) units. All heat is removed solely via the water cooling loop of LRZ [5]. CoolMUC uses Simple Linux

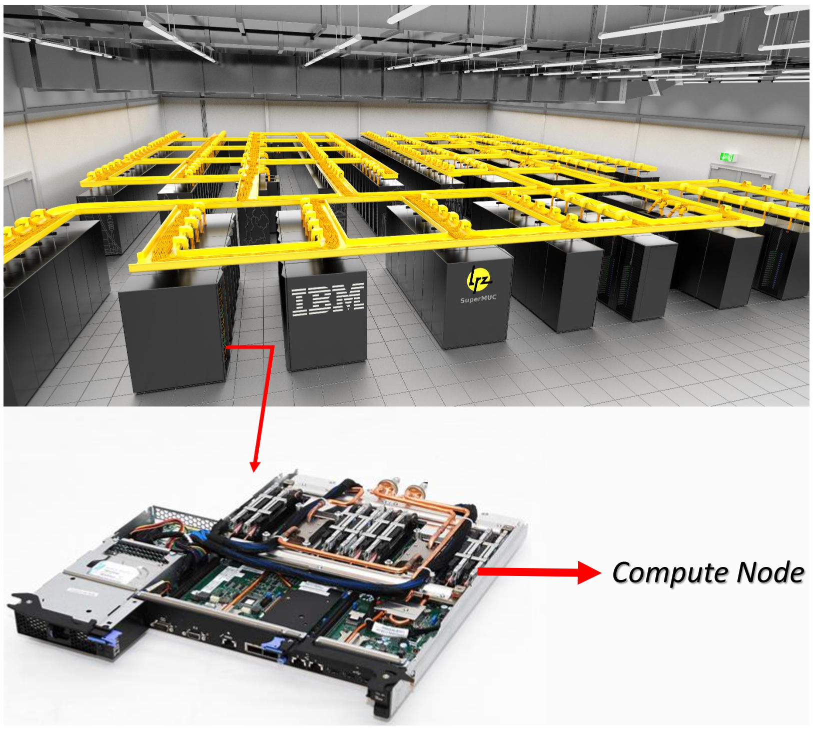


Figure 2.7.: The SuperMUC supercomputer

Utility For Resource Management (SLURM) [54] as a RMSS.

The second system is SuperMUC, shown in Figure 2.7, which is the 14^{th} fastest supercomputer in the world according to the TOP500 [1] November 2014 rankings. It is a Gauss Center for Supercomputing (GCS) [55] infrastructure system and one of the Partnership for Advanced Computing in Europe (PRACE) [56] Tier0 systems and is built by IBM [57]. SuperMUC consists of 18 thin node islands and 1 fat node island, which is referred to as SuperMIG, and also used as a migration system. All compute nodes within an individual island are connected via a fully non-blocking Infiniband network - FDR10 for the thin nodes and QDR for the fat nodes. The high speed interconnect between the islands enables a bi-directional bi-section bandwidth ratio of $4:1$ (intra-island / inter-island). The 18 thin islands in total contain $147,456$ processor cores in 9216 compute nodes. Each node is equipped with 2×8 processors. Each processor is of Sandy Bridge-EP Xeon $E5-2680$ $8C$ type, having a maximum allowed operating frequency of 2.7 GHz (maximum turbo frequency is 3.5 GHz), and a TDP [12] of 130 Watt. SuperMUC uses IBM LoadLeveler [58] as a RMSS. The peak performance of the system is 3.2 PetaFLOPS ($= 3.2 \cdot 10^{15}$ Floating Point Operations Per Second (FLOPS)). It's active components (e.g. processors, memory, etc.) are directly cooled with an inlet water temperature of up to $40°$ Celsius [5]. Currently, only the SuperMUC RMSS data is collected by PowerDAM.

The third system is SuperMIG, shown in Figure 2.8, which is the migration system for the SuperMUC supercomputer. It contains 8200 cores in 205 compute nodes. Each node is equipped with 4×10-core processors. Each processor is of Westmere-EX Xeon $E7-4870$

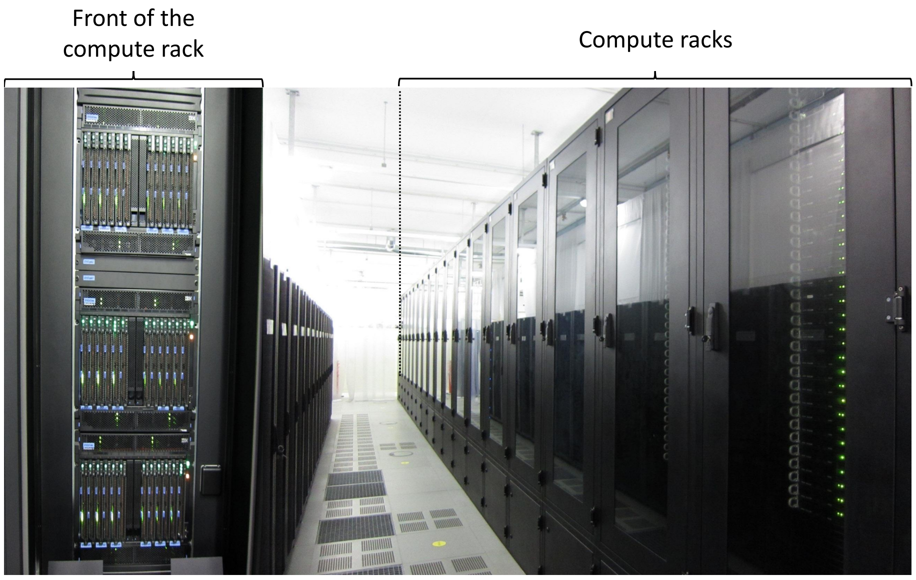


Figure 2.8.: The SuperMIG migration system

$10C$ type, having a maximum allowed operating frequency of 2.4 GHz (maximum turbo frequency is 2.8 GHz), and a TDP of 130 Watt. The peak performance is 78 TeraFLOPS $(= 78 \cdot 10^{12}$ FLOPS) [5].

Sensor data, like measurements from rack-based PDUs, are obtained using the system monitoring tools. RMSS specific data is obtained via SLURM (for CoolMUC) and LoadLeveler (for SuperMIG and SuperMUC) specific commands.

CoolMUC has a dedicated per-node PDU outlet, which allows for calculating the EtS of a given application (as described in Section 2.6) independently from other running applications.

On SuperMUC, each of the 18 islands is comprised of seven compute racks, where each rack has 2 PDUs. A single PDU has 6 outlets and each outlet can power from 4 to 8 compute nodes, which makes it impossible to accurately calculate the power consumption of individual applications. Figure 2.9 illustrates this issue. Consider the following scenario - an application A is utilizing 6 compute nodes (a_1 to a_6) which are connected to the *outlet 2* of a PDU. Meantime, an application B is utilizing 4 nodes, two of which (a_7 and a_8) are powered through the same *outlet* 2, whereas the rest two (b_1 and b_2) are powered through the *outlet 3*. Since the only available measurements are the ones that can be read from the PDU outlets, this makes it impossible to correctly identify the power consumption share of application A and application B by only using the power values obtained from outlets 2 and 3. For this reason, PowerDAM further relies on the paddle card data for obtaining per node power measurements. The LoadLeveler, the RMSS of SuperMUC (and SuperMIG), uses this data to calculate the EtS of an application. PowerDAM uses directly this calculated data for the SuperMUC and SuperMIG cases, since the same power measurement setup is present for the SuperMIG migration system. Section 3.6 discusses in detail the accuracy of these EtS measurements.

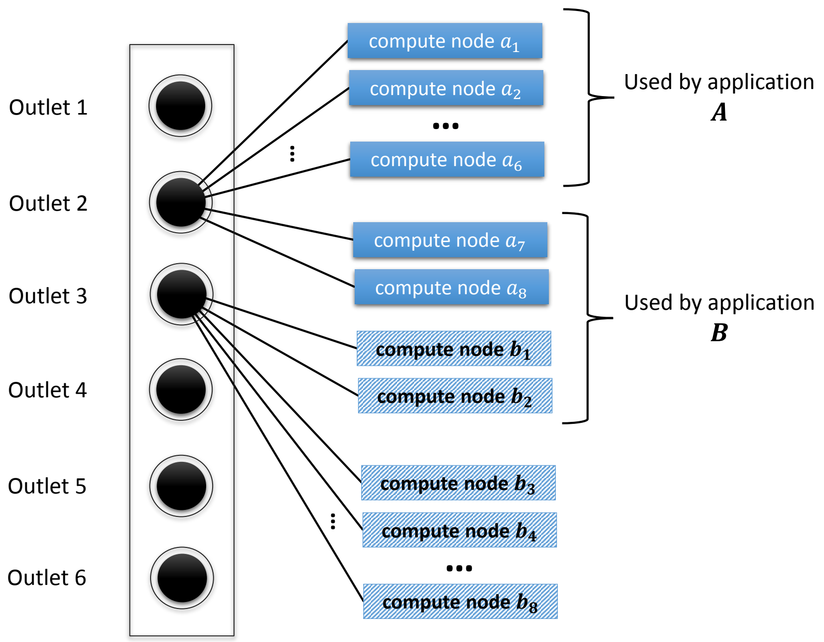


Figure 2.9.: Relation between SuperMUC PDU outlets and the compute nodes

As for the CoolMUC case, the monitored PDUs from which the sensor data is obtained in the current system setup deliver averaged (over 60 seconds) power measurements in one minute intervals. Thus there is a high probability that the time stamps which correspond to the PowerDAM monitoring steps do not match with exact starting and ending time stamps of the executed applications. This results in a maximum error of 2 minutes (1 minute for start time detachment and 1 minute for end time detachment) in the measurements, which further means that in order to have a measurement error of less than 5% in the current setup, the application has to run a period of at least 40 minutes. Since most of the HPC applications today run for several hours, this error becomes negligible.

2.6. PowerDAM and EtS

Energy-to-Solution (EtS) [59] is an important metric for PowerDAM. This value indicates the aggregated energy consumption of compute nodes used by the application/job and partial sub-system components (system networking, system cooling, and infrastructure) which were utilized during the run of that application. The infrastructure can be either system specific or encompass a value for multiple systems.

The EtS for a given finished job J on system S is calculated iteratively as:

$$EtS(J,S) = \sum_{i=startIteration}^{endIteration} \Delta t_i \cdot P_i(J,S) \tag{2.2}$$

where

- i is the iteration index
- $startIteration = \min\{i \mid J^{startTime} \leq timestamp_i \leq J^{endTime}\}$
- $endIteration = \max\{i \mid J^{startTime} \leq timestamp_i \leq J^{endTime}\}$
- $J^{startTime}$ and $J^{endTime}$ are respectively start and end times of job J
- $\Delta t_i = timestamp_i - timestamp_{i-1}$
- $timestamp_i$ is the time and date of i^{th} iteration
- P_i is defined below (2.3)

Since system cooling and networking power can only be measured for the entire system, a way has to be found to attribute a fraction of total power of such sub-systems to individual nodes (Equation 2.3). The required cooling for a component can be seen as directly related to its consumed power. Therefore, a job's share of system cooling power is directly related to the proportion of its consumed power and the overall power consumed by all nodes at a given timestamp. For calculating a job's share of the overall networking power consumption we assume that the moment the node is active it's networking port is active as well. For the power consumption of the current InfiniBand network fabric the communication pattern seems to have no significant impact. This behavior was observed on SuperMUC, where the power consumption of the InfiniBand switches was relatively constant under different communication patterns. Thus, a job's networking share is a fraction of the overall system networking consumption defined by the ratio of utilized nodes and the overall number of active nodes in the system.

$$P_i(J,S) = P_i^J + \frac{P_i^J \cdot P_i^{S_{cooling}}}{P_i^S} + \frac{P_i^{S_{networking}} \cdot N_i^J}{N_i^S} \tag{2.3}$$

where

- P_i^J is the power sum of all compute nodes which were utilized by job J at the i^{th} iteration of monitoring
- P_i^S is the power sum of all active system nodes at the i^{th} iteration of monitoring
- $P_i^{S_{cooling}}$ is the cooling power value of entire system S at the i^{th} iteration of monitoring
- $P_i^{S_{networking}}$ is the networking power value of entire system S at the i^{th} iteration of monitoring
- N_i^S is the number of system active nodes at the i^{th} iteration of monitoring
- N_i^J is the number of nodes utilized by job J at the i^{th} iteration of monitoring

2.6.1. EtS on CoolMUC, SuperMIG, and SuperMUC

This subsection shows the usage of PowerDAM for determining the most-energy efficient HPC system (out of the three described in Section 2.5) for running a scientific application HYDRO [60], which is an application-benchmark extracted from the real-world astrophysical code RAMSES [61]. HYDRO is a computational fluid dynamics $2D$ code, which uses the finite volume method, with a second order Godunov scheme [62] and a Riemann solver [63] at each interface on a $2D$ mesh, for solving the compressible Euler equations of hydrodynamics. The input problem size was set to use 320 tasks. 20 compute nodes were used on CoolMUC and SuperMUC (since on each system a compute node has 16 cores) and 8 nodes were utilized on SuperMIG (since each compute node on SuperMIG has 40 cores).

```
      timestamp, sensor, value, unit
      2015-03-23 03:26:33, mpp1_networking_power, 5256.0 W
      2015-03-23 03:26:33, mpp1_cooling_power, 8055.7 W
      2015-03-23 03:26:33, mpp1.lxa159_power, 138.0 W
      2015-03-23 03:26:33, mpp1.lxa159_load, 58.167708125 %
      2015-03-23 03:26:33, mpp1.lxa167_power, 213.0 W
      2015-03-23 03:26:33, mpp1.lxa167_load, 62.239754375 %
      2015-03-23 03:26:33, mpp1.lxa9_power, 236.0 W
      2015-03-23 03:26:33, mpp1.lxa9_load, 62.883333125 %
(a)
                              ...

      2015-03-23 04:48:28, mpp1.lxa78_power, 233.0 W
      2015-03-23 04:48:28, mpp1.lxa78_load, 100.0 %
      2015-03-23 04:48:28, mpp1.lxa110_power, 231.0 W
      2015-03-23 04:48:28, mpp1.lxa110_load, 96.82738125 %
      2015-03-23 04:48:28, job.mpp1_cooling_power, 168.597716462 W
      2015-03-23 04:48:28, job.mpp1_networking_power, 574.863387978 W
      ==================================================================
       WARNING: Following sensor measurements were invalid and have been approximated
                (sensor name; timestamp; approximated value; original value)
      mpp1.lxa147_load ; 2015-03-23 03:28:09 ; 100.0 % ; 103.006355625 %
(b)   mpp1.lxa166_load ; 2015-03-23 03:28:09 ; 100.0 % ; 106.166666875 %
      mpp1.lxa32_load ; 2015-03-23 03:28:09 ; 100.0 % ; 101.05 %

                                      ...
      ==================================================================
      EtS is: 6.28719 kWh
      Computation:   73 %
(c)   Networking:    10 %
      Cooling:       15 %
```

Figure 2.10.: Detailed EtS report of HYDRO on CoolMUC

The consumed energy of an application executed on a given system is calculated[6] by PowerDAM using the collected sensor values. The EtS of an application can be queried through the usage of the following command:

```
./PowerDAM_UI --option=ets --system=SystemName [--user=userID] --job=jobID [--trace]
```

The latter *trace* option, when specified, prints in addition to the EtS value the detailed sensor trace of all the compute nodes which were utilized by the given job and all sensor measurements that were approximated. Figure 2.10 illustrates the trace enabled EtS reporting option for HYDRO when executed on CoolMUC using 10 compute nodes.

The first *part (a)* of the report presents the sensor measurements for all utilized components. The first two entries show system level measurements as system networking power (presented as *mpp1_networking_power*) and system cooling power (presented as *mpp1_cooling_power*) in the order of *timestamp, sensor name, value,* and *unit*. The subsequent entries show the power and utilization rates of all compute nodes utilized by the application. Finally, it shows the application's share of system networking and system cooling power presented as *job.mpp1_networking_power* and *job.mpp1_cooling_power*. This output format is then repeated for the complete runtime of the application.

The second *part (b)* of the report lists all replacements of invalid measurement data. In this case, an invalid utilization rate measurement of 103% for *lxa147* compute node was recorded. This invalid source data was normalized to 100%.

The final *part (c)* of the report informs on EtS and also on the consumption percentages of the computation, networking, and cooling. HYDRO ran 83.63 minutes on CoolMUC consuming around 6.29 kWh energy, 73% of which was spent on computation, 10% was spent on networking, and 15% on cooling.

Table 2.1 shows the execution results for HYDRO when run on SuperMIG and SuperMUC using the same input problem size. As can be seen, despite the 3 minute overhead, the execution of HYDRO on SuperMIG consumes on average less power and is more energy-efficient than the corresponding execution on SuperMUC.

System	Number of Utilized Compute Nodes	Measured EtS Value (kWh)	Measured APC Value (W)	Measured TtS Value (min)
SuperMIG	8	1.88	2443.74	46.3
SuperMUC	20	2.32	3199.04	43.56

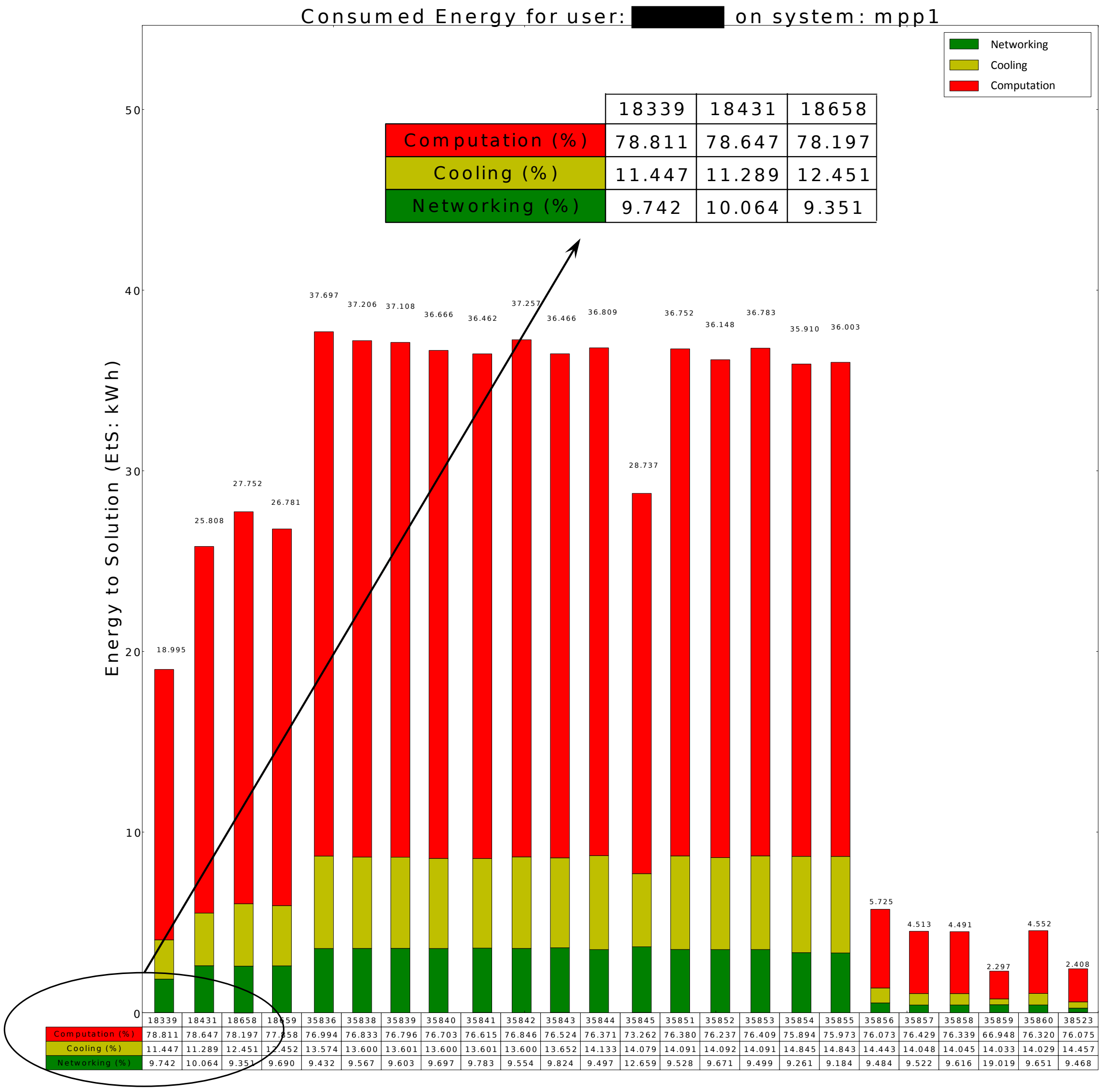
Table 2.1.: HYDRO execution results on SuperMIG and SuperMUC systems

2.7. Some PowerDAM Reporting Features

PowerDAM provides a separate plug-in interface for defining customized data analysis reports. This section will present some of the currently existing ones.

Figure 2.11 illustrates the detailed EtS report showing the consumption percentages of computation and sub-system components of all submitted and executed applications for

[6] And collected in cases of SuperMUC and SuperMIG as described in Section 2.5.



	18339	18431	18658
Computation (%)	78.811	78.647	78.197
Cooling (%)	11.447	11.289	12.451
Networking (%)	9.742	10.064	9.351

	18339	18431	18658	18659	35836	35838	35839	35840	35841	35842	35843
Computation (%)	78.811	78.647	78.197	77.858	76.994	76.833	76.796	76.703	76.615	76.846	76.524
Cooling (%)	11.447	11.289	12.451	12.452	13.574	13.600	13.601	13.600	13.601	13.600	13.652
Networking (%)	9.742	10.064	9.351	9.690	9.432	9.567	9.603	9.697	9.783	9.554	9.824

	35844	35845	35851	35852	35853	35854	35855	35856	35857	35858	35859	35860	38523
Computation (%)	76.371	73.262	76.380	76.237	76.409	75.894	75.973	76.073	76.429	76.339	66.948	76.320	76.075
Cooling (%)	14.133	14.079	14.091	14.092	14.091	14.845	14.843	14.443	14.048	14.045	14.033	14.029	14.457
Networking (%)	9.497	12.659	9.528	9.671	9.499	9.261	9.184	9.484	9.522	9.616	19.019	9.651	9.468

Figure 2.11.: PowerDAM detailed EtS report for a user on a given HPC system

a given user on a given (in this case the MPP CoolMUC) system. The y-axis represents the EtS values of the user submitted applications in kWh. The x-axis represents a table where the first row presents the RMSS assigned application ids, and the second, third, and fourth rows present the computation, cooling, and networking consumption percentages respectively.

Figure 2.12 illustrates the power, load (utilization), and temperature graphs for an application having a RMSS assigned 42664 id. The graph in the upper left corner shows the accumulated power of all used compute nodes. The one in the upper right corner shows the normalized load rates of the compute nodes. The lower left and the lower right graphs show the averaged temperature behavior of each CPU of the compute nodes[7].

[7]On CoolMUC MPP Linux cluster, each compute node has 2 × 8-core AMD CPUs.

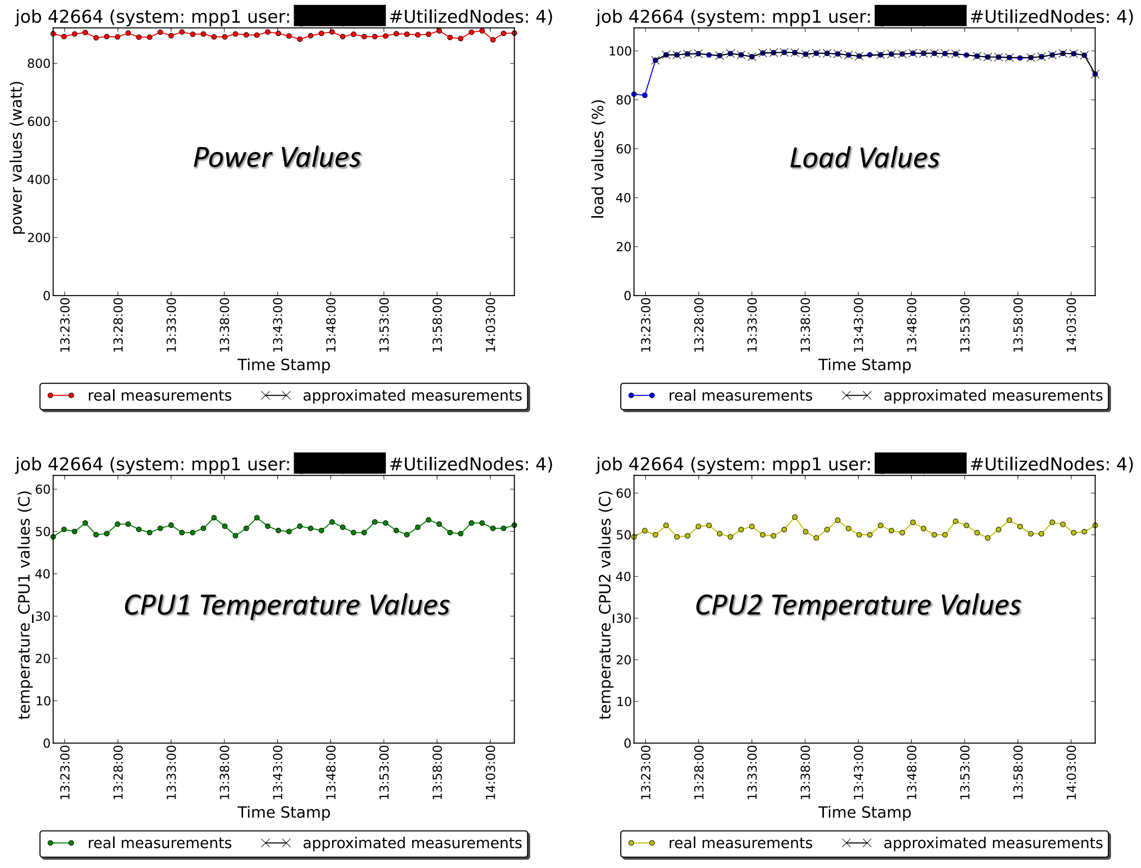


Figure 2.12.: Power, Load, and CPU Temperature graphs for an Executed Application

The cyan colored measurement points in the load graph (upper right corner of Figure 2.12) illustrate the PowerDAM approximated measurements for all compute nodes which were utilized by the application 42664.

Figure 2.13 illustrates the correlation between power and load of the compute nodes which were utilized by a given application. The top line illustrates the application power draw, and the bottom line the load rate. This correlation can be used, for example, to classify the compute nodes into several zones (groups) according to the performance per watt indicator of the compute nodes. This classification could then be used by the RMSS of the target high-end system for an energy efficient selection of compute resources. A possible example of an efficient selection policy can be the scheduling of computation-intensive applications to zones having higher performance/watt ratio and reservation of zones with lower performance/watt ratio for data-intensive applications. Another would be to schedule the applications on the most efficient zones first and on the least efficient last.

The classification maps of the dynamic change of compute node sensor data through a color mapping is another analysis option provided by PowerDAM. The two presented classification maps beneath can represent any sensor type which is supported by the target compute nodes (e.g. power, load, CPU temperatures, etc.). These maps update automat-

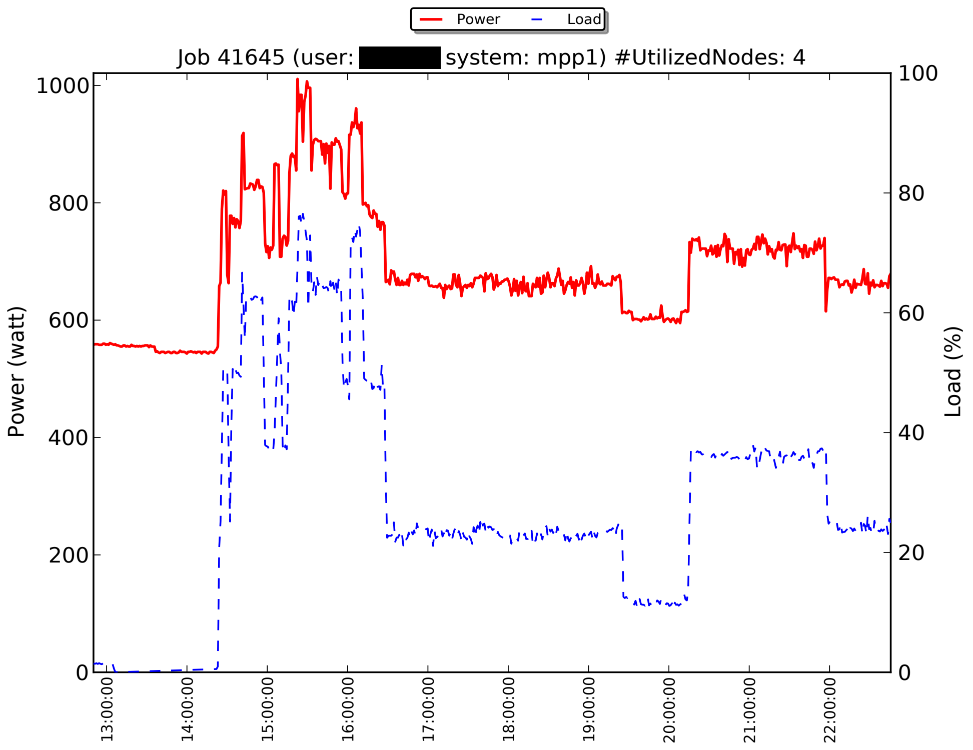


Figure 2.13.: Correlation between power and load for a given application

ically after a specified amount of time. Figure 2.14 presents the example classification of load sensor data for the CoolMUC compute nodes.

For this example, three classification categories were defined with different ranges. The color green in Figure 2.14 (rectangles labeled with star)[8] corresponds to the desired load rate of compute nodes and was mapped to illustrate the $95\% - 100\%$ load range. The color white corresponds to the normal load rate of compute nodes and was mapped to illustrate the 0% and the $90\% - 94\%$ load range. The color red (rectangles without star markers) corresponds to less favorable load rate of compute nodes and was mapped to illustrate the $1\% - 89\%$ load range.

Figure 2.15 and Figure 2.16 present another classification map example that is currently provided by PowerDAM. The columns in these figures illustrate the compute racks of the CoolMUC system, and each cell represents a compute node. Figure 2.15 illustrates the average CPU temperature heat map of all the compute nodes of the CoolMUC prototype. Each cell in Figure 2.15 contains the compute node name (e.g. lxa83), the power value (e.g. 232 W), and the utilization/load rate (e.g. 99.8%). Similarly, Figure 2.16 illustrates the power heat map of the CoolMUC compute nodes. Other reporting options of PowerDAM include the Coefficient of Performance (COP) analysis of the adsorption chiller, system power consumption for a given time frame (e.g. day, month, and year), system PUE, system ERE, etc.

[8]The star markers are being used only for illustrative purposes and are not part of PowerDAM compute node classification map.

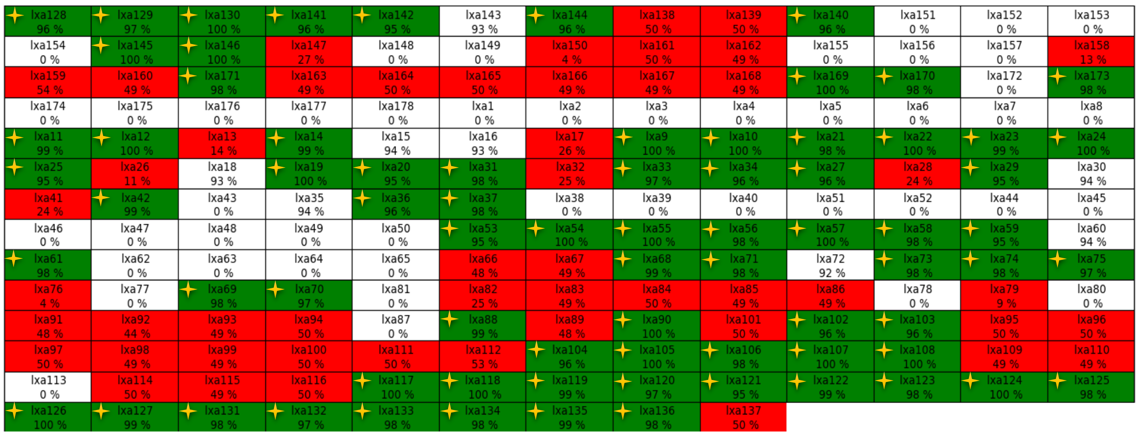


Figure 2.14.: PowerDAM classification map example of load sensor data for compute nodes of CoolMUC Linux cluster

2.8. PowerDAM Current Development Directions

The PowerDAM workflow described in Subsection 2.4.2 has two potential bottlenecks: the processing time of the received data and the waiting time, aroused due to the data pulling requests. In order to eliminate the waiting time and improve the overall scalability, PowerDAM is currently being developed to support for:

- **push network communication service**
  Where the data will get automatically 'pushed' into PowerDAM by the monitored entity.

(also known as publish/subscribe communication model [64]) in addition to the described in Subsection 2.4.2:

- **pull network communication service**
  Where the data collection request is initiated by PowerDAM.

In this way, PowerDAM is allowing for asynchronous data insertions for different sensors having potentially different read-out frequencies.

Figure 2.17 shows the architecture of the in development version of PowerDAM publish/subscribe communication model. In this messaging pattern, the data sender parties, referred to as *publishers* do not send the messages/data directly to the receiving parties, referred to as *subscribers*. Similarly, the *subscribers* receive only those messages that they had subscribed to without knowledge of the exact publishers. This makes data providers loosely coupled with data receivers, allowing each to operate independently from the other.

The mentioned decoupling is achieved through an intermediary physical component, referred to as a *broker*, that handles the communication between sending and receiving parties. *Publishers* post messages to the *broker* which then performs the filtering, i.e. the process of selection and processing of the posted messages to the registered subscriptions. The open source *Mosquitto* [65] Message Queue Telemetry Transport (MQTT) TCP/IP-based

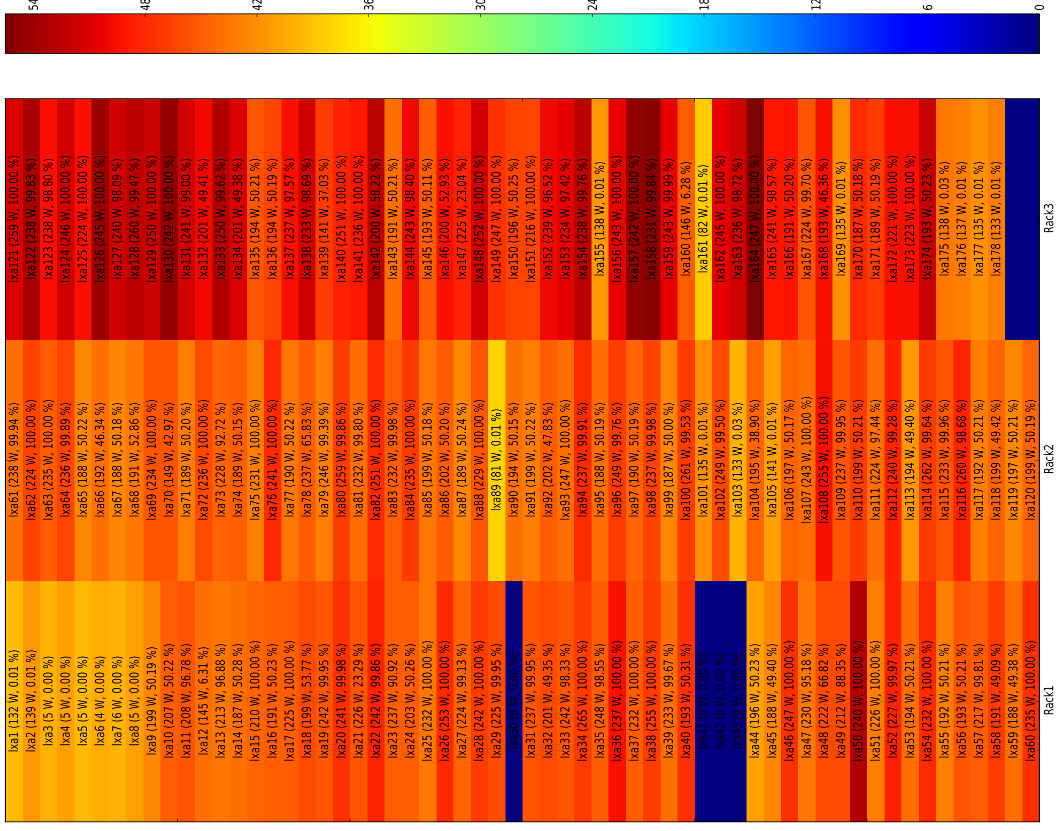


Figure 2.15.: PowerDAM temperature heat map for CoolMUC Linux cluster

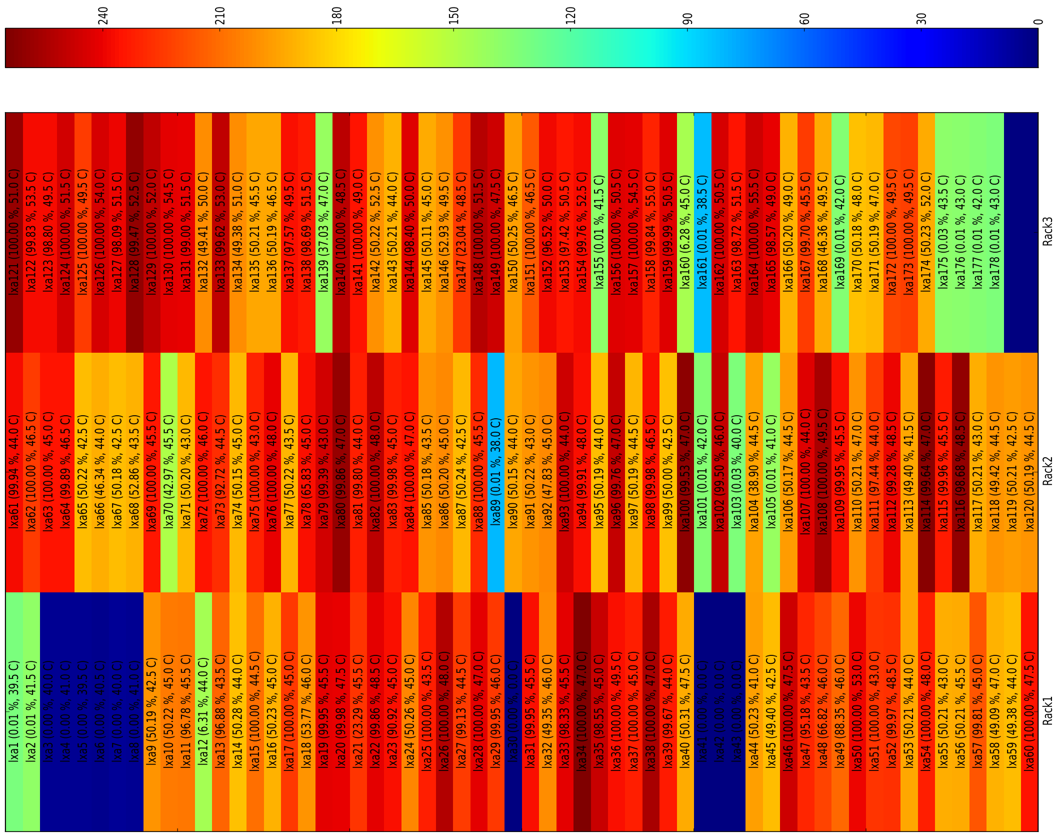


Figure 2.16.: PowerDAM power heat map for CoolMUC Linux cluster

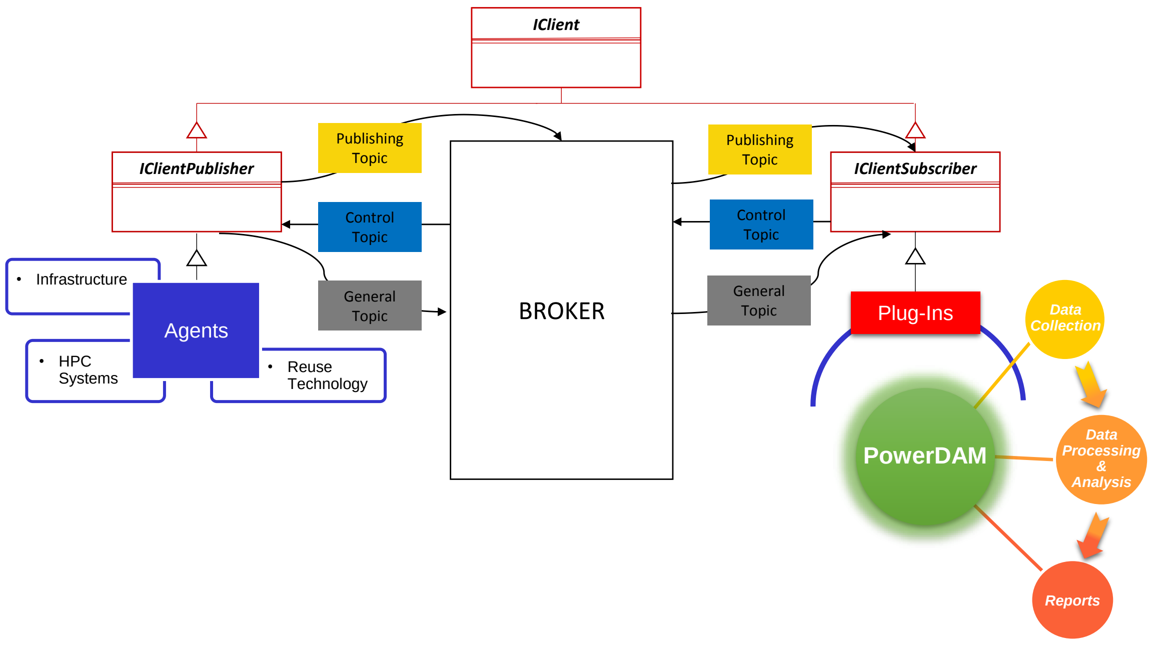


Figure 2.17.: PowerDAM and the publish/subscribe communication protocol

protocol is used as a data broker. Mosquitto is a publish/subscribe, simple and lightweight communication model designed for resource-constrained devices and low bandwidth, high latency, which also allows for its effective usage in embedded systems. These makes it a good fit for simple notification scenarios as the updates of power, temperature, humidity, etc. sensor values.

The *topic-based* filtering is used for distributing the data to the subscribers, i.e. the exchange of the information is accomplished through a set of predefined topics, representing many-to-many (and one-to-many in case of MQTT) distinct *logical* channels. The *subscribers* will receive only the messages which were published to the topics to which they had subscribed. The all subscribers of the *same topic* will receive the *same data*. The main advantages of the topic-based filtering approach are: *(i)* the enforced platform interoperability achieved through the usage of strings as keys to divide the event space (thus more lightweight than its content-based counterpart), which also makes the routing simple through multi-cast group to peers that match the subscription topics, and *(ii)* the usage of hierarchies to orchestrate topics (typically denoted with URL-like notation) and wildcards which allow for organizing the topics according to their containment relationships [66].

Three topics are maintained for this case:

- **Publishing Topic** - used for publishing/sending the actual, in-band, monitored data (the data flows from agents to PowerDAM);
- **Control Topic** - used for communicating with publishers in cases of out-of-band requests, such as data resubmission request, failure notifications, etc. (the data flows from PowerDAM to agents); and
- **General Topic** - used for answering out-of-band requests originated from PowerDAM

via the control topic (the data flows from agents to PowerDAM).

As can be seen, in the presented case, both the agents and PowerDAM act as a publisher and meantime as a subscriber but on different topics. The main drawback of the described publish/subscribe communication model is the side-effect of one of its main advantages: the loose coupling between publishers and subscribers, which adds an additional complexity for further development, maintenance, and testing.

2.9. PowerDAM Usage By Other Data Centers

The early version of PowerDAM was released to PRACE Work Package (WP) 11 partners for a primary evaluation of the software package. Based on the obtained feedbacks the software was further improved.

Two examples of PowerDAM installation and usage for different supercomputing systems are described in the subsections beneath. Overall, they demonstrate the PowerDAM applicability range in the energy efficiency analysis of various HPC prototype systems as well as in assessment of the energy consumption for various large-scale applications using diverse programming paradigms.

2.9.1. Usage at PSNC

PowerDAM was installed as a part of the software monitoring stack on the hybrid CPU/GPU prototype system at Poznan Supercomputing and Networking Center (PSNC) [67]. The main aim of the installation was the provision of the application energy consumption profiles to the system users.

CPU/GPU is a single rack cluster built by Iceotope [68]. The cluster has 40 compute nodes equipped with dual Xeon Sandy Bridge 2620 with a nominal frequency of 2.0 GHz and 2.3 GHz at turbo mode. Each compute node has 32 GB Random-Access Memory (RAM), dual 1 Gbit Ethernet links, and InfiniBand QDR. One compute node is connected to a dedicated 10 TB Solid-State Drive (SSD) matrix exposed to the nodes using Lustre storage architecture. Six compute nodes are connected to slave modules that host 2 AMD S9000 Graphics Processing Units (GPUs) each. The prototype uses SLURM as a RMSS, and thus it was possible to reuse the already developed (for CoolMUC) SLURM-for-PowerDAM plug-in and the corresponding agent for obtaining the application relevant information.

At PSNC, PowerDAM was deployed to monitor the per compute node available information - CPU utilization rate, CPU temperatures, and the power consumption (direct current). Additionally, data regarding the networking and cooling loops (both in rack-pumps and external) were collected to allow for an accurate EtS calculation of the executed applications.

2.9.2. Usage at CINECA

PowerDAM was also installed on EURORA prototype system which is deployed at CINECA [69]. EURORA is a single rack cluster equipped with 64 compute nodes, each with two Intel Xeon Sandy Bridge eight-core processors. The internal network is composed of 1 Altera Stratix V series Field Programmable Gate Array (FPGA) per node, and

IB QDR and 3D Torus interconnects. It has a per node 16 GByte DDR3 1.6 GHz RAM and 160 GByte SSD disc space. EURORA is a warm water cooled system and was the most energy efficient HPC system, with an overall of 3, 208.83 MegaFLOPS/W performance, according to June 2013 Green500 rankings [70]. The system uses PBS Professional (PBSPro), developed by Altair [71], as a RMSS which additionally required a PBS-for-PowerDAM plug-in development. Currently, PowerDAM is a part of the EURORA system software stack and is used for collecting thermal, power, and energy consumption related data for various projects.

2.10. PRACE 1IP WP 9 & 2IP WP 11

The development of PowerDAM started within the scope of the WP 9 *"Future Technologies"* and the WP 11 *"Prototyping"* of the PRACE First and Second Implementation Phase (1IP and 2IP) Projects [56].

The WP 9 *"Future Technologies"* project was aimed towards the exploration of state-of-the-art hardware and software technologies, the evaluation of emerging programming paradigms and the assessment of energy efficient solutions for current leadership and Tier-0 class and future multi-Peta/Exascale HPC systems.

The WP 11 *"Prototyping"* project was aimed at prototyping novel scalable and sustainable computing platforms with the potential for energy efficient computing. This WP investigated number of evolutionary technologies including the latest cooling and hardware, interconnect and accelerator co-processing technologies as well as OS and systems management solutions that would most likely be representative for next generation HPC systems.

As can be seen, both projects required a tool allowing for assessment of the current states of KPIs, which range from the level of scientific applications (e.g. EtS, APC, etc.) to the target HPC systems and reuse technologies (e.g. PUE, ERE, WUE, etc.) for identifying the improvement areas and verifying the success of applied optimizations techniques. PowerDAM was developed specifically for addressing these issues.

2.11. The SIMOPEK Project

PowerDAM is currently being developed within the scope of the Simulation and Optimization of Data Center Energy Flows (Simulation und Optimierung des Energiekreislaufs von Rechenzentrums-Klimatisierungsnetzen unter Berücksichtigung von Supercomputer-Betriebsszenarien) (SIMOPEK) project [72]. The SIMOPEK project is aimed towards modeling, simulation, and optimization of the energy flows within HPC data centers. The MYNTS software [73], developed at Frauhofer SCAI, is being extended for simulating and optimizing the energy flow networks of the data center in a wholistic way, by taking into account the dynamic utilization state of the target HPC as well as its technological and heat reuse components (e.g. hot water cooling, adsorption cooling, etc.).

The purpose of PowerDAM within the SIMOPEK project is the provision of the operational data from all the aspects of the data center ranging *from* the site infrastructure relevant information (e.g. cold and warm water cooling loops, electrical circuit, etc.) *over* the target HPC system(s) (e.g. system power consumption, utilization rate, etc.) *to* the user applications (e.g. power/energy consumption of individual large-scale application, etc.). Some of this collected data will then be supplied to the MYNTS software for the corresponding simulations which will assist in finding energy optimal operational solutions. These solutions, in their turn, will then be verified, again, through the usage of PowerDAM.

2.12. Summary

This chapter discussed the first steps required for a unified energy measurement and evaluation toolset for a HPC data center. It presented the developed data collection and analysis tool, PowerDAM, which can be used for the evaluation of energy consumption and energy efficiency of the individual HPC system as well as the complete data center. The following bullet points summarize the main use cases of PowerDAM.

- The data provided by PowerDAM can help certain RMSS in performing energy efficient decisions
  Having the knowledge of the energy/power consumption of individual applications, it will be possible, for example, to schedule only high priority workload (desirable with low energy consumption rate) during the peak hours and defer the 'energy-hungry' low priority workload to off-peak hours when the cost of electrical power is cheaper.

- PowerDAM can assist in classifying the application according to runtime, temperature, power, or energy consumptions as well as correlate these metrics for further analysis
  The mentioned classification/correlation will further motivate the need of application performance tuning and allow for onward exploration of power/energy consumption conservation static/dynamic techniques (e.g. voltage, current, and frequency scaling).

- PowerDAM can assist in determining the best (performance, power and/or energy wise) HPC system from the set of monitored data center supercomputers for a given large-scale parallel application
  As was seen in Section 2.6 the TtS, APC, and EtS profiles of the same application can be different on different HPC systems. The knowledge on the application consumption profiles for different, monitored HPC systems will allow for an efficient selection of a compute system for a particular application.

- The usage of PowerDAM will allow for better interdependency analysis of different data center components and energy efficiency metrics
  For example, the collected data can assist in better understanding of power, energy, and temperature behaviors of a given set of compute resources under same or different workloads/utilization rates.

- PowerDAM can be used for monitoring and assessing the current status of various, energy efficiency relevant, metrics and KPIs
  This is a necessary feature for detecting the present inefficiencies and verifying the affect of the applied improvements.

- Violations of the predefined power, energy, or thermal constraints, which can range from an individual HPC system to the entire data center, can be tracked with the usage of PowerDAM
  This is a necessary feature for controlling and verifying the applied power, energy, and/or thermal capping solutions.

In short, PowerDAM provides the functionalities that are needed to verify and improve the existing and future data center power consumption and energy efficiency models. It allows for an integrated view across multiple sources and structures and affords a better understanding of complete data center power profile.

Part III.

Covering the Prerequisites for Implementing Energy and Power Capping

"The pure and simple truth is rarely pure and never simple"

OSCAR WILDE

3. Predicting the Energy and Power Consumption of Strong and Weak Scaling HPC Applications☆

3.1. Preface

Having built a software toolset allowing for the measurement and analysis of the power and energy consumption of various large-scale parallel applications in Chapter 2, this chapter will focus on the problems of modeling and beforehand estimation of the power and energy consumption of given applications for different numbers of to-be-utilized compute nodes (servers). More specifically, this chapter will present a model, referred to as Adaptive Energy and Power Consumption Predictor (AEPCP), that is capable of predicting the EtS and the APC metrics for any HPC application, demonstrating strong or weak scaling (described below), with respect to the given number of compute nodes. The model is application independent but provides application specific results. It requires unique identifiers for each application and takes the available per-application historical power/energy data (obtained from the energy/power monitoring solution) as an input. The AEPCP model automatically adapts with each additional execution of the application (throughout the lifetime of the target HPC system) improving the associated prediction accuracy. The model does not require any application code instrumentation and does not introduce any application performance degradation. The chapter will conclude by showing the applicability of the AEPCP model in the real-wold power and energy capping scenarios.

3.2. Investigating the Scaling Behaviors of HPC Applications

The scalability of a parallel HPC application indicates the relation between application execution time and the number of application utilized compute resources, e.g. nodes. There are two different types of scalability studies - *strong* [74] and *weak* [75]. The main difference between them is the change of the application *input problem size*, i.e. the amount of required computation.

Scaling is referred to as **strong** when an application input problem size stays fixed independently from the number of compute nodes which are utilized to solve that problem. This implies that an application demonstrating a strong scaling will have a smaller execution time, i.e. will solve the computation faster, as the number of compute nodes increase

☆This chapter is partly based on the following previous work of the author: *Hayk Shoukourian et al. Predicting the Energy and Power Consumption of Strong and Weak Scaling HPC Applications. Supercomputing Frontiers And Innovations, Vol 1, No 2, pages* 20 − 41, *2014.*

[76].

Scaling is referred to as **weak** when the input problem size of the application stays fixed for each utilized compute node. This indicates that the execution time of an application under weak scaling will show a constant behavior since the input problem size increases accordingly with the number of utilized compute nodes.

Figure 3.1 shows the execution-time, i.e. TtS, behavior for strong and weak scaling scenarios [76].

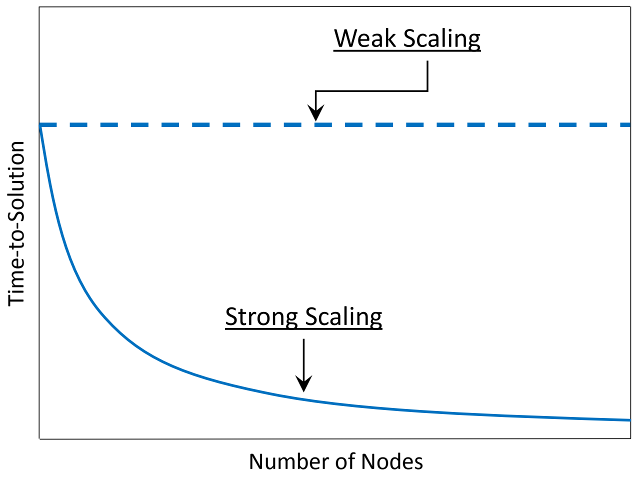


Figure 3.1.: Theoretical TtS curves for strong and weak scaling scenarios

The following denotations and definitions are further used in this chapter:

- $t_s(n)$ - processing time of the application serial part using n nodes;
- $t_p(n)$ - processing time of the application parallel part using n nodes;
- TtS$(1) = t_s(1) + t_p(1)$ - processing time of the application sequential and parallel parts using 1 node;
- TtS$(n) = t_s(1) + t_p(n)$ - processing time of the application sequential and parallel parts using n nodes;
- $p = \frac{t_p(1)}{t_s(1)+t_p(1)}$ - the **non-scaled** fraction of the application **parallel** part [77], i.e. the parallel portion of computation on a **sequential** system $(0 \leq p \leq 1)$. Thus the **non-scaled** fraction of the application **sequential** part will be $(1-p)$;
- $p^* = \frac{t_p(n)}{t_s(1)+t_p(n)}$ - the **scaled** fraction of the application **parallel** part [77], i.e. the parallel portion of computation on a **parallel** system $(0 \leq p^* \leq 1)$. Thus the **scaled** fraction of the application **sequential** part will be $(1-p^*)$;
- k - the fraction of power that is consumed by the compute node in idle state $(0 \leq k \leq 1)$. It is assumed that one compute resource consumes a power of 1 in active state.

3.2.1. Strong Scaling

Strong scaling was first described analytically by Gene Amdahl in 1967 [74]. According to Amdahl's law, the possible speedup that a parallel application can achieve using n compute nodes is:

$$Speedup(n) = \frac{\text{TtS}(1)}{\text{TtS}(n)} = \frac{1}{(1-p) + \frac{p}{n}} \tag{3.1}$$

The total TtS(n) processing time of sequential and parallel parts using n compute nodes, according to Amdahl's law (Equation 3.1), can be derived as:

$$\text{TtS}(n) = \text{TtS}(1) \cdot [(1-p) + \frac{p}{n}] \tag{3.2}$$

A study by Woo and Lee [78], considering Amdahl's law, proposes an analytical model for calculating the APC(n) of a given application when executed on n compute resources. Using the following two observations done in [78]: ***(i)*** the amount of power consumed using n compute resources during the sequential computation phase is: $1 + (n-1) \cdot k$, since only one compute resource is active, while the rest $(n-1)$ are idling; and ***(ii)*** since it takes $(1-p)$ and $\frac{p}{n}$ to execute the sequential and parallel fractions respectively [74], Woo and Lee [78] derive the APC of a given application for given n count of compute nodes as follows:

$$\text{APC}(n) = \frac{(1-p) \cdot [1 + (n-1) \cdot k] + \frac{p}{n} \cdot n}{(1-p) + \frac{p}{n}} = \frac{1 + (n-1) \cdot k \cdot (1-p)}{(1-p) + \frac{p}{n}} \tag{3.3}$$

This further means that when an application demonstrates **ideal scalability** (i.e. when the application parallelizable fraction p equals to 1, and thus the serial fraction $(1-p)$ becomes 0), then APC$(n) = n$ (dashed yellow line in Figure 3.2). While when an application demonstrates **no scalability** (i.e. when the application parallelizable fraction p equals to 0, and thus the serial fraction $(1-p)$ becomes 1), APC$(n) = 1 + (n-1) \cdot k$ (solid yellow line in Figure 3.2).

Knowing that the aggregated energy consumption EtS(n) is the product of application execution time TtS(n) and average power consumption APC(n), the following is obtained from Equation 3.2 and Equation 3.3:

$$\text{EtS}(n) = \text{TtS}(n) \cdot \text{APC}(n) = \text{TtS}(1) \cdot [1 + (n-1) \cdot k \cdot (1-p)] = \text{O}\big(n\big) \tag{3.4}$$

where $\text{O}\big(g(n)\big) = \{f(n) | \; \exists \; c > 0 \; constant \; and \; n_0 \; such \; that \; 0 \leq f(n) \leq cg(n) \; \forall n \geq n_0\}$ [79].

Note, that when an application demonstrates ideal scalability the EtS behavior will be constant (dashed red line in Figure 3.3), while in case of no scalability the EtS behavior will be linear (solid red line in Figure 3.3).

3.2.2. Weak Scaling

Weak scaling was described analytically by John L. Gustafson in 1988 [75]. According to Gustafson's law (also known as Gustafson-Barsis' law [80]), the possible speedup that a parallel application can achieve using n compute nodes is:

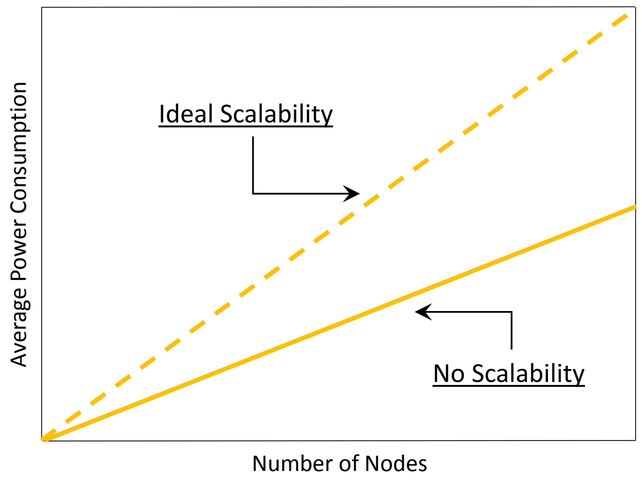


Figure 3.2.: Theoretical APC curves for ideal and no scalability cases for **strong and weak** scaling scenario

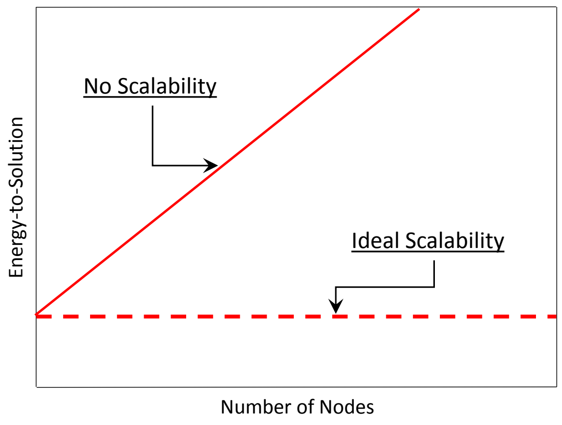


Figure 3.3.: Theoretical EtS curves for ideal and no scalability cases for **strong** scaling scenario

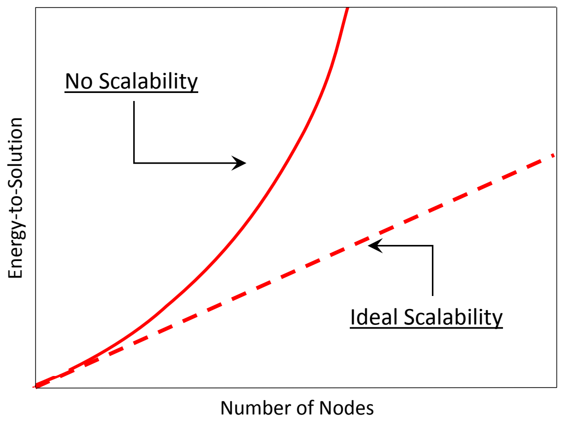


Figure 3.4.: Theoretical EtS curves for ideal and no scalability cases for **weak** scaling scenario

$$Speedup(n) = \frac{\text{TtS}(1)}{\text{TtS}(n)} = 1 + (n-1) \cdot p^* \tag{3.5}$$

Following the same observations proposed by Woo and Lee [78] and noting that it takes $t_s(1)$ to execute the sequential portion of the computation and it takes $t_p(n)$ to execute the parallel portion of the computation, the average power consumption APC(n) with respect to the n number of utilized compute nodes for the case of weak scaling can be written as:

$$\begin{aligned}\text{APC}(n) =& \frac{t_s(1) \cdot [1 + (n-1) \cdot k] + t_p(n) \cdot n}{t_s(1) + t_p(n)} = (1-p^*) \cdot [1 + (n-1) \cdot k] + p^* \cdot n = \\ & 1 + p^* \cdot (n-1) + (1-p^*) \cdot (n-1) \cdot k = \text{O}\big(n\big)\end{aligned} \tag{3.6}$$

This further means that when an application demonstrates **ideal scalability** (i.e. $p^* = 1$) then the average power consumption APC$(n) = 1 + (n - 1) = n = \mathrm{O}\big(n\big)$ showing a linear behavior (dashed yellow line in Figure 3.2). Note, that since the TtS execution time in the case of ideal scalability remains constant as the input problem size increases in parallel with the number of compute nodes, it can be further stated that the EtS behavior of the application with respect to the given n number of compute nodes is of a linear order, since: EtS(n) = APC$(n) \cdot$ TtS$(n) = n \cdot \mathrm{O}\big(1\big) = \mathrm{O}\big(n\big)$. The dashed red line in Figure 3.4 depicts this EtS behavior.

When an application demonstrates **no scalability** (i.e. $p^* = 0$) then the average power consumption APC$(n) = 1 + (n - 1) \cdot k$ (see Equation 3.6). Since the execution time of an application in the case of no scalability increases linearly with the input problem size and the number of compute nodes, it can be further stated that the EtS behavior of the application for this case will have a quadratic behavior, since: EtS(n) = APC$(n) \cdot$ TtS(n) = $[1 + (n - 1) \cdot k] \cdot \mathrm{O}\big(n\big) = \mathrm{O}\big(n^2\big)$. The solid red line in Figure 3.4 depicts this EtS behavior.

As can be implied from the above discussion, the average power consumption of an application, for both strong and weak scaling applications, is the highest when it demonstrates ideal scalability. Thus, an artificial hardware power cap, as described in [26], might keep an application from providing the highest performance and could increase the overall TtS, and subsequently EtS as well.

While it was possible to analytically describe the TtS(n), APC(n), and EtS(n) boundary curves for applications demonstrating strong and weak scaling, as can be observed, the knowledge of an application's non-scaled p (in case of strong scaling) or scaled p^* (in case of weak scaling) fractions (which are application specific information) is necessary in order to estimate the execution time, power, and energy consumption profiles for a given n number of compute nodes. The obtainment of this application specific information could be impractical in real-world scenarios where myriad of different HPC applications are run in a HPC data center.

3.3. Adaptive Energy and Power Consumption Prediction (AEPCP) Model

Figure 3.5 presents the overview of the *AEPCP* prediction process [81], i.e. the structured set of activities required to develop the software (e.g. specification, design, etc.). The *AEPCP* process has two inputs:

- **application identifier** used for uniquely identifying an application, and
- **number of system compute resources** (e.g. CPU, compute nodes, accelerators, etc.), which are planned to be utilized by a given application.

The application identifier is used to query the application-specific history information from the system monitoring tool (step 1). This application-specific history profile data (step 2), together with the number of compute resources, is passed to the predictor (step 3) for corresponding EtS and/or APC predictions. Using this data, the predictor then reports the predicted EtS/APC value for the application with respect to the given number of compute resources (step 4).

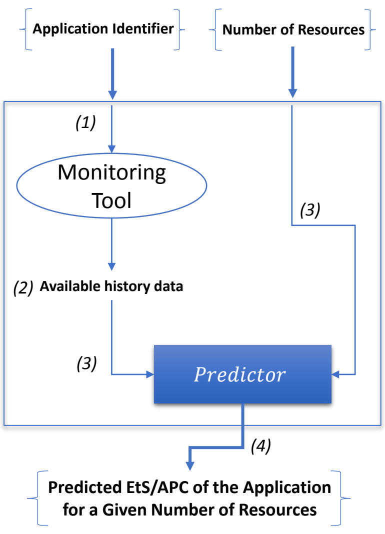

Application Identifier
Number of Resources
(1)
Monitoring Tool
(3)
(2) Available history data
(3)
Predictor
(4)
Predicted EtS/APC of the Application for a Given Number of Resources

Figure 3.5.: Overview of the *AEPCP* process

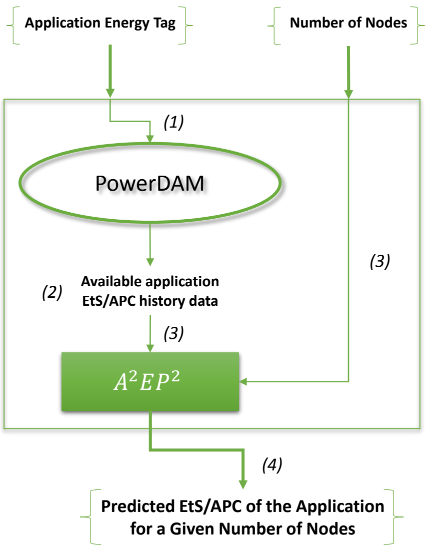

Application Energy Tag
Number of Nodes
(1)
PowerDAM
(3)
(2) Available application EtS/APC history data
(3)
A^2EP^2
(4)
Predicted EtS/APC of the Application for a Given Number of Nodes

Figure 3.6.: Overview of the *AEPCP* model

Figure 3.6 presents the *AEPCP* model based on the prediction process described above. The AEPCP model takes two inputs, namely:

- **application energy tag** used as an application unique identifier, which is supported by the IBM LoadLeveler [58] RMSS and is specified by the user on a unique-per-application basis; and
- **number of compute nodes** used as the number of compute resources, since the compute node is the smallest compute unit available to an application on the SuperMUC supercomputer which was used to validate the model (see Section 2.5).

The model uses the Adaptive Application Energy and Power Predictor (A^{2}EP^2) to estimate the EtS/APC consumption of a given HPC application for any given number of compute nodes. The A^{2}EP^2 requires the historical EtS/APC data of the given parallel application. The application energy tag is used by the system monitoring tool to retrieve the required history EtS/APC relevant profile data of the application (step 1, Figure 3.6). In our case, PowerDAM (see Section 2.4) is used as a monitoring tool for extracting this data. The workflow of the A^{2}EP^2, illustrated in Figure 3.7, is as follows. First the A^{2}EP^2 checks if the application has already been executed for a given number of compute nodes, i.e. if the EtS/APC consumption data for that given number of compute nodes is available in the PowerDAM database. If this is the case, then A^{2}EP^2 reports the averaged[1] value of all the available application history EtS/APC consumption data for that given number of compute nodes (step $Y1$, Figure 3.7).

If the PowerDAM database does not contain any application relevant EtS/APC entries for a given number of compute nodes, then A^{2}EP^2 queries the existing application specific history data (step $N1$, Figure 3.7). Once the application EtS/APC consumption history

[1]This can be modified to the maximum or the minimum depending on the use case.

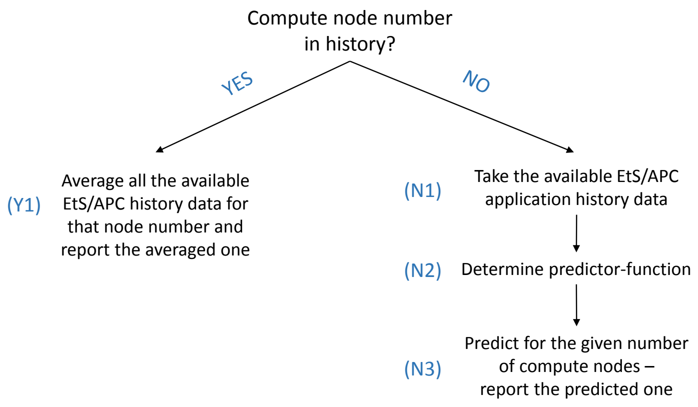

Compute node number in history?
YES
NO
(Y1)
Average all the available EtS/APC history data for that node number and report the averaged one
(N1)
Take the available EtS/APC application history data
(N2)
Determine predictor-function
(N3)
Predict for the given number of compute nodes – report the predicted one

Figure 3.7.: A^2EP^2 workflow

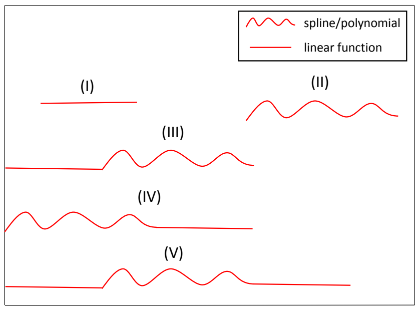


Figure 3.8.: A^2EP^2 estimation scenarios

data is obtained, A^{2}EP^2 tries to determine a predictor-function (step $N2$, Figure 3.7) which will have an allowed, user[2] specified, Percentage Root Mean Square Error (%RMSE). The %RMSE is calculated from the Root Mean Square Error (RMSE) [82] as follows:

$$\%RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i^{measured} - x_i^{predicted})^2} \cdot \frac{100 \cdot n}{\sum_{i=1}^{n} x_i^{measured}} \tag{3.7}$$

where

- n is the number of available real measurements
- $x_i^{measured}$ is the i^{th} measured real value
- $x_i^{predicted}$ is the i^{th} predicted value

Several estimation techniques (e.g. ordinary least squares, spline interpolation [83], etc.) accompanied with EtS/APC consumption specific constraints (e.g. strict positivity) are used by A^{2}EP^2 for determining the predictor-function. As implied from Section 3.2, the APC behavior for the both strong and weak scaling applications is of the $\mathrm{O}(n)$ linear order and the EtS behavior is of $\mathrm{O}(n)$ order for strong scaling applications and of $\mathrm{O}(n^2)$ order for weak scaling applications. For this reason, A^{2}EP^2 further analyzes the available history data and tries to find data points, from the obtained application specific history EtS/APC data, which would have a linear dependency. Depending on this analysis, A^{2}EP^2 divides the available history data set into linear and non-linear segments. A^{2}EP^2 distinguishes five different segmentations, as illustrated in Figure 3.8: *linear* (case I) is used for tracking the boundary curves described in Section 3.2; *non-linear* (case II) is used to track the transitional scaling phases between ideal scalability and no-scalability; *linear combined with non-linear* (case III) is used to track the mixture of boundary and transitional scaling behaviors; *non-linear combined with linear* (case IV) is used to track the mixture of transitional and boundary scaling behaviors; and *linear combined with non-linear combined*

[2]Or data center operator.

with linear (case V) is used to track the mixture of boundary-transitional-boundary scaling behaviors. For each linear segment, A^2EP^2 uses ordinary least squares to find a linear predictor-function which will have an allowed %RMSE with the available data set in that linear segment. For each non-linear segment, A^2EP^2 uses spline/polynomial interpolations (including also 1^{st} order splines/polynomials) in order to find a predictor-function which will have an allowed %RMSE rate with the history data points which are in that non-linear segment. Although, it could be argued that there is no need for estimating higher than 2^{nd} order splines/polynomials because of known theoretical boundaries, our experiments show that in the case of very limited application specific history consumption data, the higher order splines/polynomials are helpful and could result in a better prediction accuracy for a specific range of compute nodes.

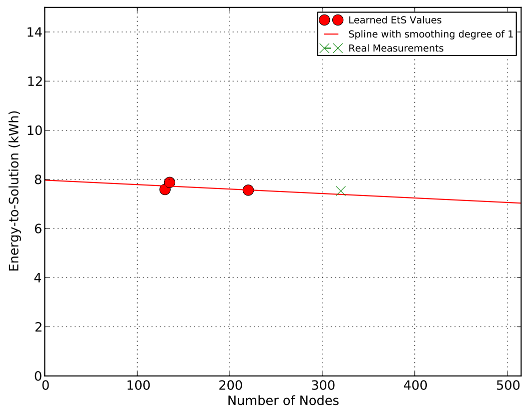
Once the predictor-function is determined by A^2EP^2, it is then used to predict the EtS/APC values of the application for a given number of compute nodes (step 4, Figure 3.6). As can be implied, while A^2EP^2 implementation approach is generic and can be used for any HPC application, A^2EP^2 produces individual results for each unique application. A^2EP^2 adapts with each additionally available EtS/APC data-point of a given application for improving the accuracy of the determined (for that application) predictor-function.

The following two sections present the AEPCP EtS and APC prediction results for two real-world scientific applications: HYDRO and EPOCH. These two applications were chosen specifically for the AEPCP model validation, since in contrast to many kernel and synthetic benchmarks, which are used to measure and test certain characteristics (e.g. processor power, communication rate, etc.) of the target platform, HYDRO and EPOCH as an actual scientific applications provide a better measure of a real-world performance. The history data points used throughout the sections were chosen on a random basis: *(i)* since the data center has no control on the resource configurations requested by the users; and *(ii)* to explicitly show that the AEPCP model is independent from any specific history data.

3.4. AEPCP Validation

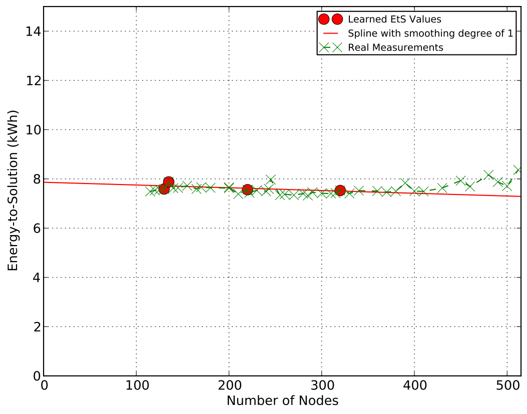
Assume that a user has executed HYDRO three times, with the same input problem size, on compute node counts: 130 with EtS of 7.6 kWh; 135 with EtS of 7.9 kWh; and 220 with EtS of 7.6 kWh. Assume further, that the application queue of the system RMSS contains a job of the same user with the same energy tag that was used for the previous three executions of HYDRO, but this time with a request of 320 compute nodes. In order to understand whether the corresponding execution of HYDRO will fit into the user's energy budget, the knowledge on potential EtS consumption of HYDRO with 320 compute nodes is required. It is worth noting, that currently, none of the existing RMSSs a priory (i.e. without explicitly executing HYDRO on 320 compute nodes) provides that knowledge. Whereas, Figure 3.9 shows the potential of the AEPCP model for provision of this knowledge. The x-axis in Figure 3.9 represents the compute node count and y-axis represents the corresponding EtS values in kWh. The red circle points correspond to the available (i.e. known) three EtS values of HYDRO when executed using the three different counts of compute nodes. The solid red line shows the predictor-function curve, which was determined by A^2EP^2 - a spline with a smoothing degree of 1 having an %RMSE of 1% (with the available

EtS values of node counts 130, 135, and 220). This predictor-function estimates a 7.4 kWh energy consumption for HYDRO when using 320 compute nodes. The green *'x'* point in Figure 3.9 corresponds to the measured EtS value (7.5 kWh) of HYDRO when executed on 320 compute nodes. As can be seen, the prediction error rate[3] for 320 compute nodes is 1.3%.



Note: available data points are for node numbers: 130, 135, *and* 220

Figure 3.9.: HYDRO EtS prediction for 320 compute nodes



Note: available data points are for node numbers: 130, 135, 220, *and* 320

Figure 3.10.: EtS prediction curve of HYDRO under strong scaling

Figure 3.10 illustrates the adaptability of the developed AEPCP model. It shows the AEPCP prediction results for the case when, in addition to the three EtS data points (for node counts: 130, 135, and 220), the EtS value for already executed HYDRO with 320 compute nodes is available to A^2EP^2. As in the previous case, a spline with a smoothing degree of 1, but with a different angle, having an %RMSE of 1% with the available four EtS points was determined by A^2EP^2 as a predictor-function (red solid line, Figure 3.10). The green *'-x-x-'* curve illustrates the real measured, and **not available** to A^2EP^2, EtS points for different compute node counts. As can be seen from Figure 3.10, the A^2EP^2 estimated predictor-function shows relatively small error rate up to 512 compute nodes. Table 3.1 summarizes the AEPCP prediction results for some compute node counts.

Figure 3.11 shows the measured TtS behavior of HYDRO under strong scaling. As can be seen, the obtained TtS curve adheres to the theoretical discussions presented in Section 3.2 (Equation 3.2). This further explains the close to ideal scaling EtS behavior of measured *'-x-'* points in Figure 3.10.

Figure 3.12 shows the HYDRO TtS behavior under strong scaling with a smaller input problem size. In this case, the application scaling saturation point is already reached with 256 compute nodes.

As usual, the red circle points in Figure 3.13 correspond to the available to A^2EP^2 EtS values (for node counts: 1,2, 4, 8, 16, 60, and 165). A spline with a smoothing degree of 2 was determined by the A^2EP^2 having a %RMSE of 1% with the available EtS data points. Although, the A^2EP^2 determined quadratic EtS behavior contradicts to the estimated the-

[3]Calculated as: (| predicted value − measured value | /measured value) ∗ 100.

Number of Nodes	Measured EtS Value (kWh)	Predicted EtS Value (kWh)	Prediction Error[3] (%)
115	7.5	7.7	2.7
200	7.7	7.6	1.3
285	7.5	7.3	2.7
300	7.4	7.5	1.4
340	7.5	7.5	0
400	7.5	7.4	1.3
460	7.7	7.3	5.1
500	7.7	7.3	5.2

Table 3.1.: EtS prediction results for some compute node counts when executing HYDRO under strong scaling

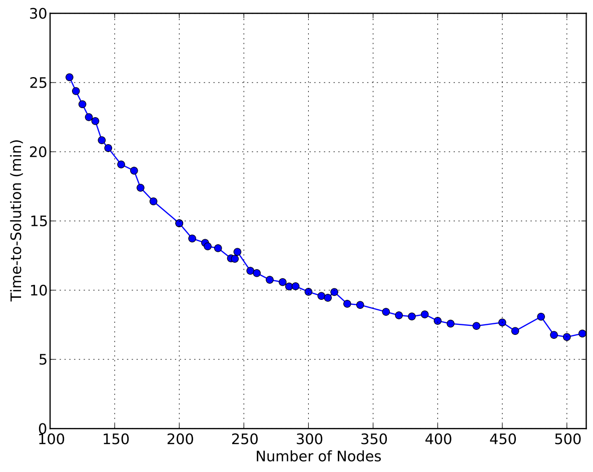


Figure 3.11.: Measured TtS of HYDRO under strong scaling

oretical linear boundary (Equation 3.4), it provides an approximation with relative small error rate when compared with the measured '-x-x-' data. On the other hand the A^{2}EP^2 estimated quadratic predictor shows relative high error rate when the application scaling is impacted by sequential-to-parallel synchronization and inter-node communication. While it can be argued that there is no reason for executing an application (and thus conducting a prediction) on a higher number of nodes than the node count on which the saturation point for a given application was observed (since no performance increase for that application will be recorded), A^{2}EP^2 might capture these synchronization and communication impacts when sufficient data is available.

A study found in [84] discusses in detail these impacts, and shows that even for applications having a parallelizable fraction p close to 1, the application speedup can decrease with higher counts of compute resources, i.e. there could be a k number of compute nodes, for which $\mathrm{TtS}(k) \leq \mathrm{TtS}(k+1)$. This behavior can also be seen for the considered HYDRO execution case, where a 34% runtime increase can be observed with 500 compute nodes when compared to 450 compute nodes (17 minutes vs 12 minutes , Figure 3.12). Figure 3.14 illustrates the mentioned A^{2}EP^2 option, when EtS values for executing HYDRO with 450 and 500 node counts were additionally available. As can be seen, the scalability limitation is tracked at the EtS value corresponding to 450 compute node count execution, and the

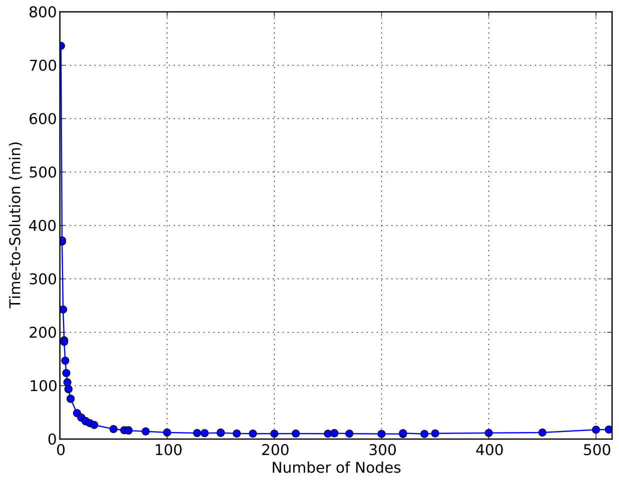
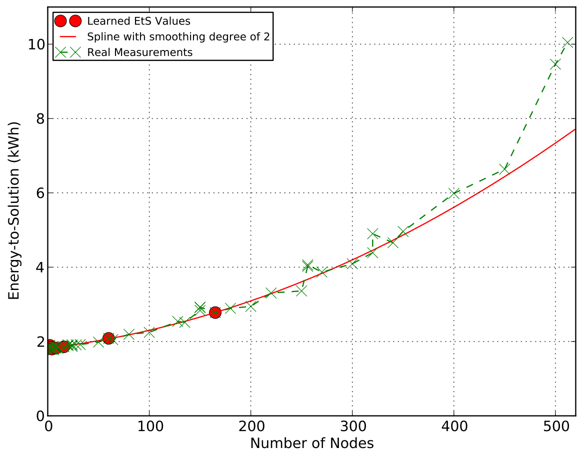
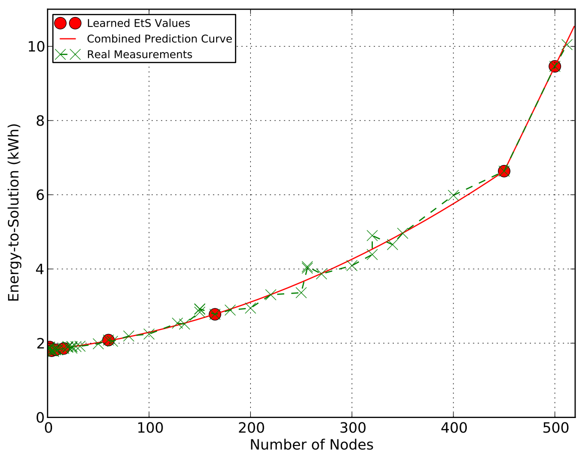


Figure 3.12.: Measured TtS of HYDRO under strong scaling *(smaller input problem size)*



Note: available data points are for nodes: 1, 2, 4, 8, 16, 60, *and* 165

Figure 3.13.: EtS prediction curve of HYDRO under strong scaling *(smaller input problem size)*



Note: additionally available data points are for nodes: 450 *and* 500

Figure 3.14.: Revisited EtS prediction curve of HYDRO under strong scaling *(smaller input problem size)*

previously estimated quadratic function is now combined with the linear function.

Figure 3.15 illustrates the expected TtS behavior of HYDRO under weak scaling (Figure 3.1). Two EtS values, for compute node counts 6 (with EtS of 0.54 kWh) and 32 (with EtS of 2.84 kWh), were available to AEPCP for conducting the prediction. Figure 3.16 illustrates the prediction results. As can be seen, the estimated linear predictor shows relative small error rate for up to 512 compute nodes. Table 3.2 summarizes these prediction results for some compute node counts.

Figure 3.17 shows the execution behavior of EPOCH under strong scaling. EPOCH is a plasma physics simulation code developed at the University of Warwick as part of the Extendable PIC Open Collaboration Project [85]. It is based upon the particle push and field

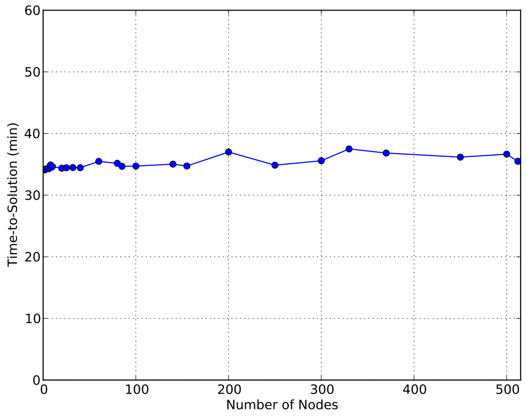
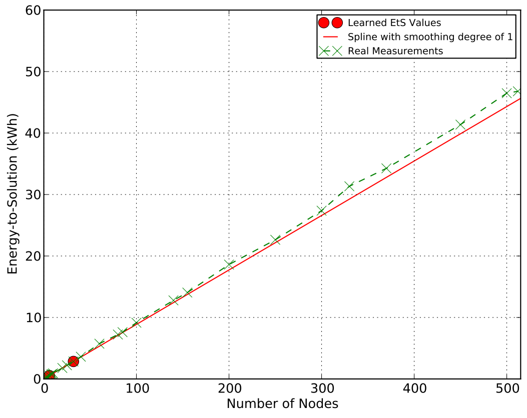


Figure 3.15.: Measured TtS of HYDRO under weak scaling



Note: available data points are for nodes: 6, and 32

Figure 3.16.: EtS prediction curve of HYDRO under weak scaling

Number of Nodes	Measured EtS Value (kWh)	Predicted EtS Value (kWh)	Prediction Error[3] (%)
10	0.9	0.9	0
25	2.2	2.2	0
40	3.6	3.6	0
80	7.2	7.1	1.4
100	9.1	8.9	2.2
200	18.6	17.7	4.8
370	34.3	32.8	4.4
450	41.4	39.9	3.6
512	46.8	45.4	3

Table 3.2.: EtS prediction results for some compute node counts when executing HYDRO under weak scaling

update algorithms [86]. EPOCH uses the Message Passing Interface (MPI)-parallelized explicit 2^{nd} order relativistic particle-in-cell method, including a dynamic MPI load balancing option.

Four EtS values, for node counts 64 with EtS of 8.6 kWh, 75 with EtS of 8.8 kWh, 90 with EtS of 8.7 kWh, and 128 with EtS of 8.7 kWh, were made available to A^2EP^2 for executing the prediction.

A linear predictor-function (solid red line, Figure 3.18), having an %RMSE of 0.8% with the available four EtS data points, was determined by A^2EP^2. As usual, the green *′-x-x-′* curve corresponds to the real measured, and not available to A^2EP^2 EtS data points. Table 3.3 recaps the prediction results for some compute node counts.

Figure 3.19 illustrates the execution behavior of EPOCH under weak scaling. Three EtS data points, for node counts 16 with EtS of 1.1 kWh, 40 with EtS of 2.9 kWh, and 64 with EtS of 4.5 kWh, were made available to A^2EP^2 for conducting the prediction.

Figure 3.20 shows the A^2EP^2 prediction results. Also for this case, a linear predictor-function (having a %RMSE of 2.2% with the available three EtS data points) is estimated by A^2EP^2. Table 3.4 summarizes the A^2EP^2 prediction results for this case.

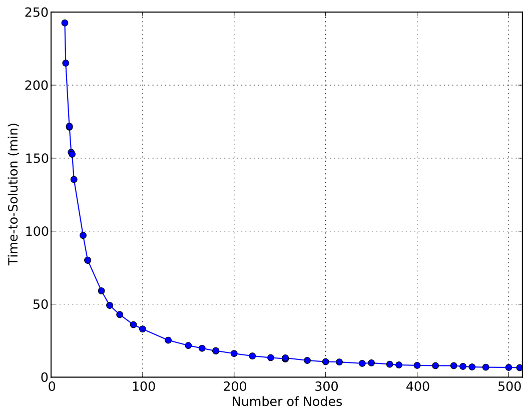
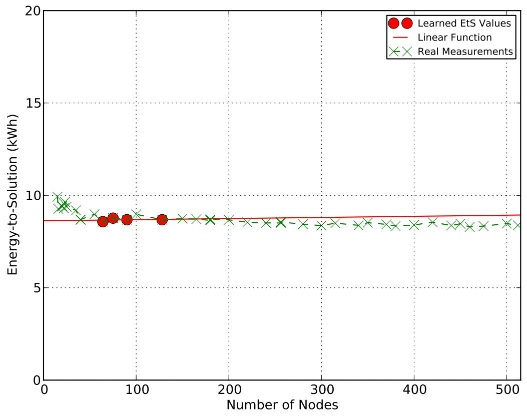


Figure 3.17.: Measured TtS of EPOCH under strong scaling



Note: available data points are for nodes: 64, 75, 90 *and* 128

Figure 3.18.: EtS prediction curve of EPOCH under strong scaling

Number of Nodes	Measured EtS Value (kWh)	Predicted EtS Value (kWh)	Prediction Error (%)
16	9.3	8.6	7.5
20	9.4	8.6	8.5
55	9	8.7	3.33
128	8.7	8.7	0
180	8.7	8.7	0
256	8.5	8.8	3.5
340	8.4	8.8	4.76
370	8.4	8.8	4.76
420	8.5	8.9	4.7
475	8.3	8.9	7.2
500	8.5	8.9	4.7
512	8.4	8.9	5.95

Table 3.3.: EtS prediction results for some compute node counts when executing EPOCH under strong scaling

Number of Nodes	Measured EtS Value (kWh)	Predicted EtS Value (kWh)	Prediction Error (%)
2	0.1	0.1	0
7	0.5	0.5	0
12	0.8	0.8	0
20	1.3	1.4	7.7
32	2.3	2.2	4.34
50	3.3	3.5	6
80	5.6	5.6	0
96	6.3	6.8	7.9
110	7.7	7.8	1.3
185	12.3	13.1	6.5
220	14.6	15.5	6.16
350	23.4	24.7	5.5
460	30.2	32.5	7.6
512	33.8	36.2	7.1

Table 3.4.: EtS prediction results for some compute node counts when executing EPOCH under weak scaling

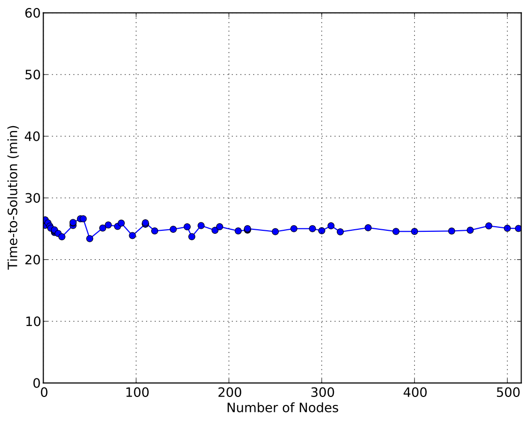
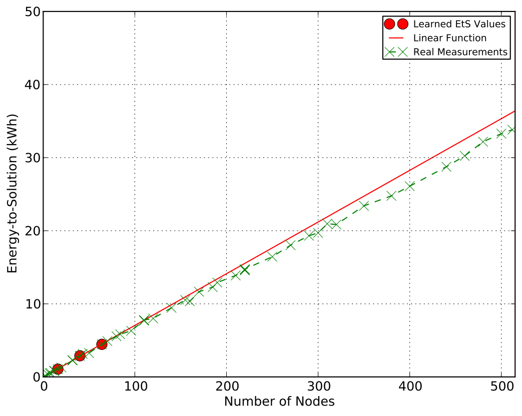


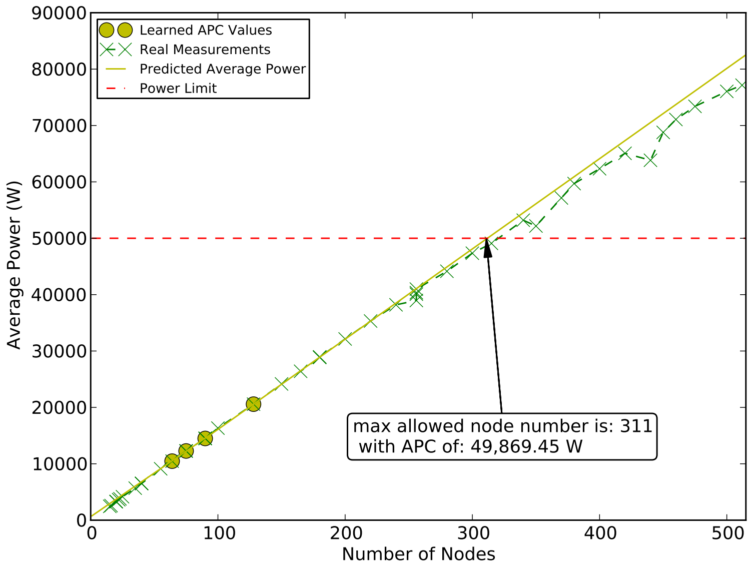
Figure 3.19.: Measured TtS of EPOCH under weak scaling



Note: available data points are for nodes: 16, 40, *and* 64

Figure 3.20.: EtS prediction curve of EPOCH under weak scaling

Figure 3.21 shows the AEPCP APC prediction results for EPOCH using the available APC values of four node counts: 64, 75, 90, and 128. The x-axis represents the compute node counts and y-axis represents the corresponding APC values in W. The yellow circle points correspond to the mentioned four available in PowerDAM database APC values. The yellow solid line in Figure 3.21 illustrates the curve of a predictor-function, estimated by the AEPCP model. The green *′-x-x-′* curve shows the measured APC values for different numbers of compute nodes. As can be seen, for this case the AEPCP estimated predictor-function shows a relatively small error rate for up to 512 compute node count.



Note: available data points are for nodes: 64, 75, 90 *and* 128

Figure 3.21.: APC prediction curve for EPOCH under strong scaling

In order to illustrate the usage of AEPCP model within the power capping problem domain, assume that the data center operator has set a power limit of 50 kW (red dashed line, Figure 3.21) that must not be violated at any given point in time. The question to answer here would be: *"what is the maximum allowed compute node count for running EPOCH on SuperMUC in the case of a* 50 *kW power cap?"* Currently, none of the available RMSSs can answer this question. As can be seen from Figure 3.21, by using the AEPCP model it is possible to find an answer - the maximum allowed compute node count for running EPOCH on SuperMUC in the case of a 50 kWh power cap is 311 with a predicted APC of $49,869.45$ W.

3.5. AEPCP and the Inhomogeneous Power Consumption of Homogeneous System Compute Resources

The conducted observations on the SuperMUC supercomputer (described in Section 2.5) show that the average power draw of the individual compute nodes differ when executing the same workload.

Figure 3.22 (a) shows the distribution of the average power draw of the individual compute nodes from SuperMUC thin island 5 when running the single-node MPRIME benchmark[4]. MPRIME is an application-benchmark that searches for Mersenne prime numbers, i.e. prime numbers of form $2^p - 1$, using Fast Fourier Transform [87] algorithm. It introduces an intense workload to processor and memory, and because of that reason is usually used for system stability testing [88]. All the cores of the compute nodes had a maximum CPU frequency of 2.3 GHz during these single-node benchmarks. The x-axis shows the observed average power consumption and the y-axis shows the compute node count.

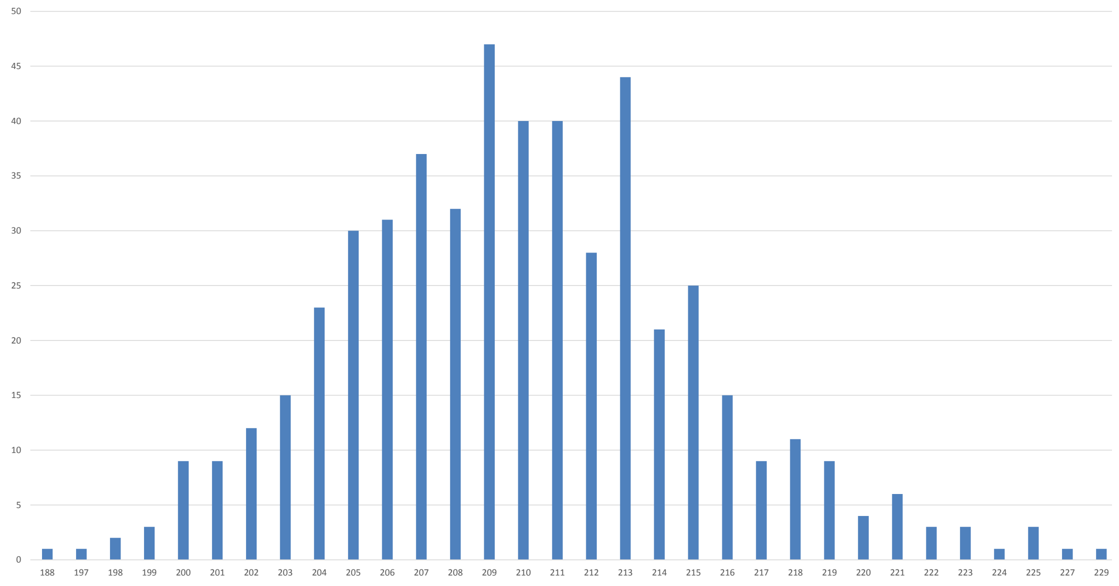
As can be seen, despite the hardware homogeneity across SuperMUC's island, there is a maximum of 41 W (which translates to 19.6%) difference (nodes $i05r05a19$ with 188 W and $i05r03c28$ with 229 W) in average power draw of compute nodes within 512 compute nodes of SuperMUC island 5. Figure 3.22 (b) shows the APC values of some compute nodes when running the MPRIME benchmark.

The observed variation in the APC of the compute nodes can be explained by the manufacturing tolerances and variations [89] during the fabrication of the integrated circuits. Since this variation is a hardware property, it will not change with different applications. It is worth noting that the observed difference could be even higher in Exascale systems, since the complexity of various hardware components will most likely increase [90].

This difference in the power draw of compute nodes under the same MPRIME workload was also observed on the CoolMUC system Linux cluster (described in Section 2.5), where the variation is around 8.4% within 178 compute nodes (nodes $lxa11$ with 240 W and $lxa46$ with 261 W), as shown in Figure 3.23.

If the discussed 'power-efficiency' pattern of system compute nodes is known (Figure 3.22, Figure 3.23), then the AEPCP model can predict an application's possible maximum and minimum APC values for a given compute node count in case of the RMSS application-assigned most and least efficient (in terms of average power draw) compute nodes. Figure 3.24 illustrates this option by extending the Figure 3.21.

[4]These benchmarks were carried out during the summer time, when the inlet temperature of the compute



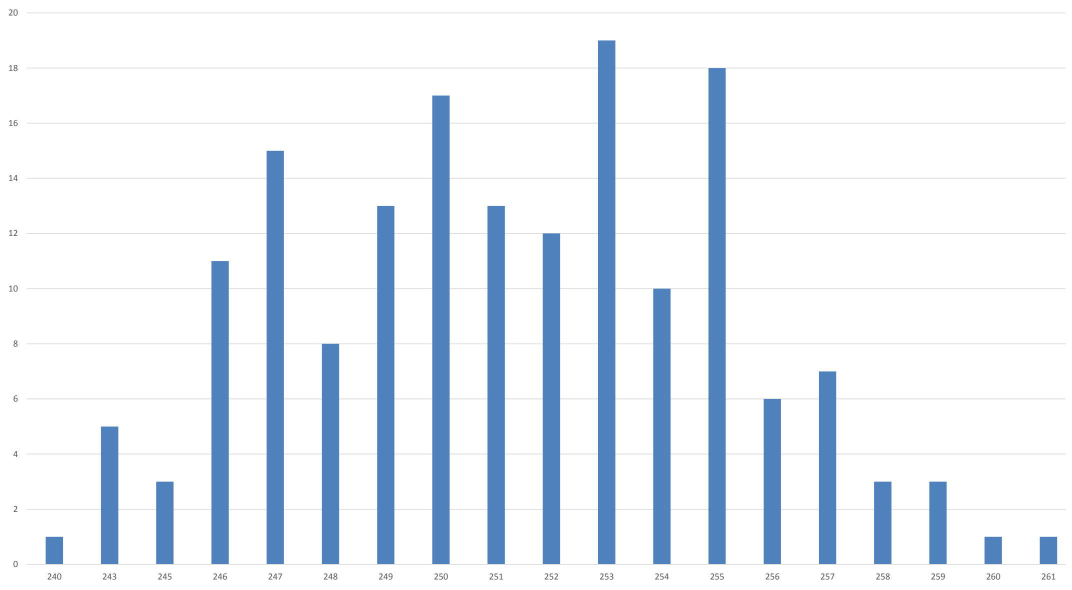
(a)

bode	Average Node Power (watt)
i05r01a03	208
i05r01a04	202
i05r01a05	207
i05r01a06	221
i05r01a07	203
i05r01a08	206
i05r01a09	203
i05r01a10	215
	■ ■ ■
i05r03c24	210
i05r03c25	212
i05r03c26	221
i05r03c27	211
i05r03c28	229
i05r03c29	216
	■ ■ ■
i05r05a16	218
i05r05a17	215
i05r05a18	203
i05r05a19	188
i05r05a20	211
i05r05a21	207
	■ ■ ■
i05r05a35	205
i05r05a36	207
i05r05a37	198
i05r05a38	219
i05r05c03	215
i05r05c04	218
i05r05c05	216
i05r05c06	215
i05r05c07	200
i05r05c08	206
	■ ■ ■

(b)

Figure 3.22.: Average power draw of compute nodes for the SuperMUC island 5 when executing MPRIME at 2.3 GHz maximum CPU frequency

nodes was set to 40 °C.



(a)

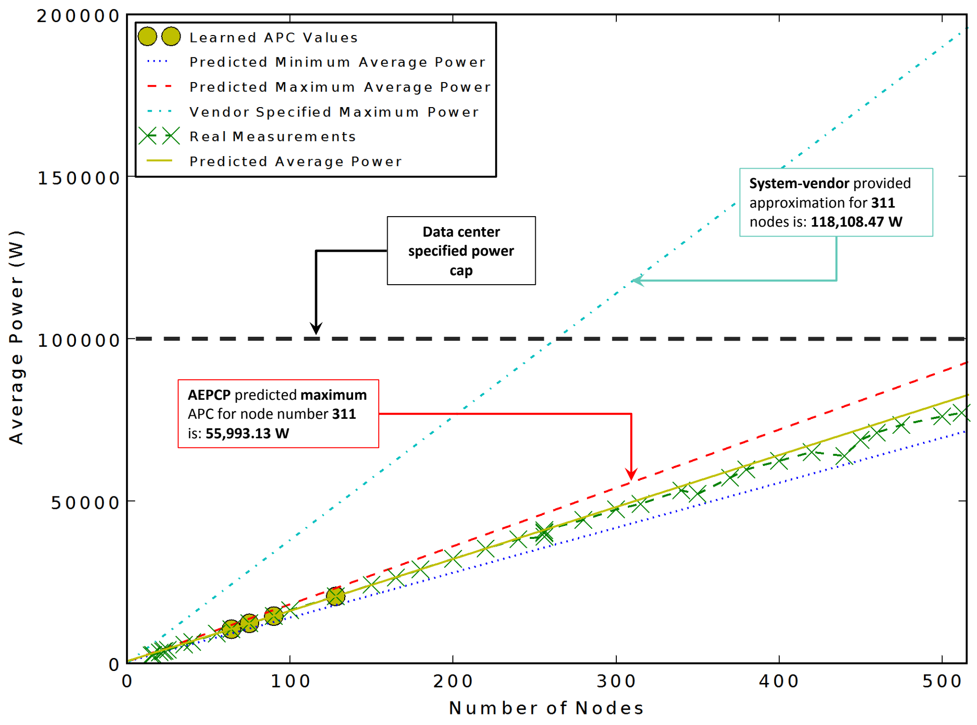
Node	Average Node Power (watt)
lxa9	250
lxa10	254
lxa11	240
lxa12	248
lxa13	247
lxa14	248
lxa15	243
lxa16	250
	■ ■ ■
lxa45	247
lxa46	261
lxa47	247
lxa48	246
lxa49	247
lxa50	251
lxa51	249
	■ ■ ■
lxa150	257
lxa151	243
lxa152	253
lxa153	251
lxa154	252
lxa155	256
	■ ■ ■

(b)

Figure 3.23.: Average power draw of compute nodes for the CoolMUC Linux cluster when executing MPRIME at 2.0 GHz maximum CPU frequency

The red dashed line in Figure 3.24 illustrates the predicted *worst-case* APC behavior, i.e. when all the compute nodes demonstrate the *least efficient* power behavior. This is accomplished through the Equation 3.8, where AEPCP normalizes the available history APC values to the usage of *the worst* (i.e. least efficient) compute node.

The blue dotted line in Figure 3.24 illustrates the predicted *best-case* APC behavior, i.e.



Note: available data points are for nodes: 64, 75, 90 *and* 128

Figure 3.24.: TDP vs AEPCP APC prediction results for EPOCH

when all the compute nodes demonstrate the *most efficient* power behavior. This is accomplished through the Equation 3.9, where AEPCP normalizes the available history APC values to the usage of *the best* (i.e. most efficient) compute node.

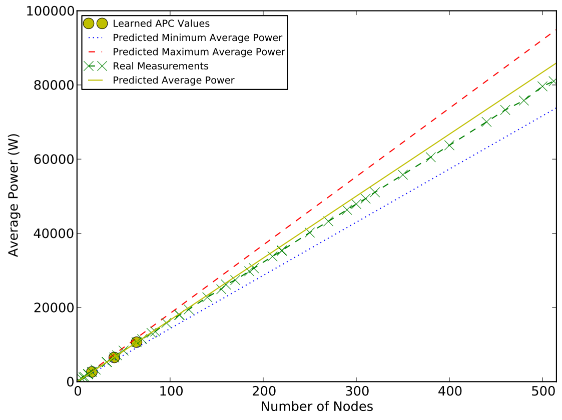
$$\| APC(J)^i \|_{max} = APC(J)^i + \sum_{\text{u utilized node of J}} (P_{max} - P_u) \tag{3.8}$$

$$\| APC(J)^i \|_{min} = APC(J)^i - \sum_{\text{u utilized node of J}} (P_u - P_{min}) \tag{3.9}$$

where

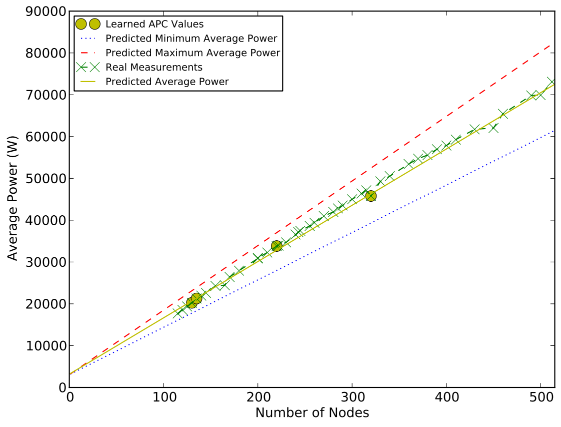
- $APC(J)^i$ - is the average power draw of job J using i compute nodes
- P_u - is the average power draw of compute node u obtained from the system compute node power classification
- P_{min} - is the average power draw of the most efficient (in terms of average power consumption) system compute node
- P_{max} - is the average power draw of the least efficient (in terms of average power consumption) system compute node

As can be seen, the real APC measurements (the green '-x-x-' line, Figure 3.24) and the AEPCP predicted APC values (yellow solid line, Figure 3.24) stay in between the predicted worst and best case APC behaviors. It could have further argued that the system-vendor provided TDP value [12] could have been used for estimating the worst-case APC behavior of an application for a given number of compute nodes - by multiplying the vendor provided $P_{\text{one node}}$ TDP value by the n number of application requested compute nodes. The cyan '-.-' line in Figure 3.24 illustrates these $P_{\text{one node}} \cdot n$ approximation. While correct, this approach is impractical. For example, in the case of a data center operator specified 100 kW power cap (black dashed line, Figure 3.24), the usage of system-vendor provided approximation will disallow the execution of EPOCH with 311 compute nodes, since it will estimate a power consumption of $118,108.47$ W. Whereas, the usage of AEPCP will show that even for the worst case scenario (i.e. when all the RMSS assigned compute nodes have the least power-efficiency), the execution of EPOCH with 311 compute nodes is allowed within the specified 100 kW power cap. As can be seen, the TDP based approach leads to roughly two times higher value as compared to the one estimated by the AEPCP model ($118,108.47$ W vs $55,993.13$ W).



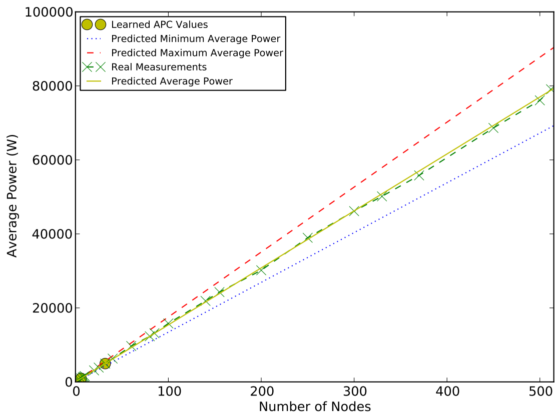
Note: available data points are for nodes: 16, 40 *and* 64

Figure 3.25.: Max and Min APC values for EPOCH under weak scaling



Note: available data points are for nodes: 130, 135, 220, *and* 320

Figure 3.26.: Max and Min APC values for HYDRO under strong scaling



Note: available data points are for nodes: 6, *and* 32

Figure 3.27.: Max and Min APC values for HYDRO under weak scaling

Figure 3.25, Figure 3.26, and Figure 3.27 show the AEPCP APC prediction results for EPOCH weak scaling, HYDRO strong scaling and HYDRO weak scaling correspondingly. As can be seen, all the three AEPCP estimated predictor-function curves show very small deviation rates from the measured green *'-x-'* APC values.

3.6. Measurement Accuracy on SuperMUC

The EtS/APC/TtS measurement data for the benchmarks on SuperMUC are obtained from the SuperMUC's RMSS LoadLeveler, which in its turn relies on the paddle cards. To ensure the measurement accuracy, four re-executions of the EPOCH benchmark on the **same** set of compute nodes were conducted. Table 3.5 summarizes the results. The first column of the table shows the utilized node count, the second column shows the measured EtS in kWh, the third column shows the TtS in minutes, and finally the forth column shows the observed maximum error between the measurements conducted for the same number of nodes.

Number of Nodes	Measured EtS Value (kWh)	TtS (min)	Maximum Error (%)
20	9.4	171	$<$ ***1.2***
	9.3	170	
	9.4	171	
	9.4	171	
90	8.7	36	$<$ ***1.2***
	8.6	36	
	8.7	36	
	8.7	36	
180	8.7	18	$<$ ***1.2***
	8.6	18	
	8.6	18	
	8.7	18	
256	8.5	12	$<$ ***1.2***
	8.5	13	
	8.5	13	
	8.6	13	

Table 3.5.: EPOCH strong scaling rerun on the same set of compute nodes

As can be seen, the measurement error per node count does not exceed 1.2%. The study found in [28] has also shown the high accuracy of the paddle card measurements, by using higher-level instrumentation points integrated in SuperMUC's PDUs. Thus, it can be further assumed that the quality of a single measurement (independently from the number of utilized compute nodes) is relatively high, thus dismissing the need of any benchmark re-execution.

3.7. AEPCP Features and Summary

An adaptive model, referred to as Adaptive Energy and Power Consumption Predictor (AEPCP), which allows for predicting the EtS and APC values of a given application with respect to the given number of utilized compute nodes was presented. This chapter also showed that there exists a variation in the average power consumption of the two newer *homogeneous* (in terms of the installed compute nodes) HPC systems' compute nodes.

The presented AEPCP model could be a very interesting solution for HPC data centers, since it requires no application specific information. The chapter showed the validity of the model using two real-world scientific applications. It also described the applicability of the AEPCP model for the discussed in Chapter 1 two most important use cases.

The following bullet points summarize the main features of the developed AEPCP model:

- the AEPCP modeling approach is application neutral, i.e. does not require any knowledge on application type (e.g. communication, computation, or memory intensive), on scaling properties, or on any other application internal property;
- the prediction accomplished through the AEPCP model does not require any application code instrumentation;
- AEPCP does not introduce any application performance degradation;
- AEPCP allows for ahead of time EtS/APC prediction of a given application for a given number of compute nodes and does not require any partial/phase application executions;
- AEPCP automatically captures the complexity of the underlying hardware platform by taking the input data directly from the system [32], i.e. does not require any manual tuning of application properties or architectural peculiarities of the target platform;
- AEPCP can predict the maximum and minimum energy and power consumptions of the application depending on the RMSS assigned *"worst"* and *"best"* (in terms of APC) compute nodes;
- AEPCP provides a generic solution that can be used for each application, but produces an application specific result;
- AEPCP adapts with each additionally available application specific EtS/APC data point, thus improving the prediction accuracy for that application; and
- the prediction accomplished through AEPCP can be done automatically (if required, transparent from user) for any queued or running set of HPC applications.

4. Advancing the Adaptive Model to Support for Prediction of Energy Consumption Relevant Indicators For Different Compute Resource Configurations☆

4.1. Preface

Chapter 3 presented an Adaptive Energy and Power Consumption Predictor (AEPCP) model, that allowed for the prediction of the APC and EtS of a given parallel application for a given number of compute nodes. The AEPCP model is based on analytical models [74, 78, 75] that do not take into account the possibility of the operating maximum CPU frequency modification.

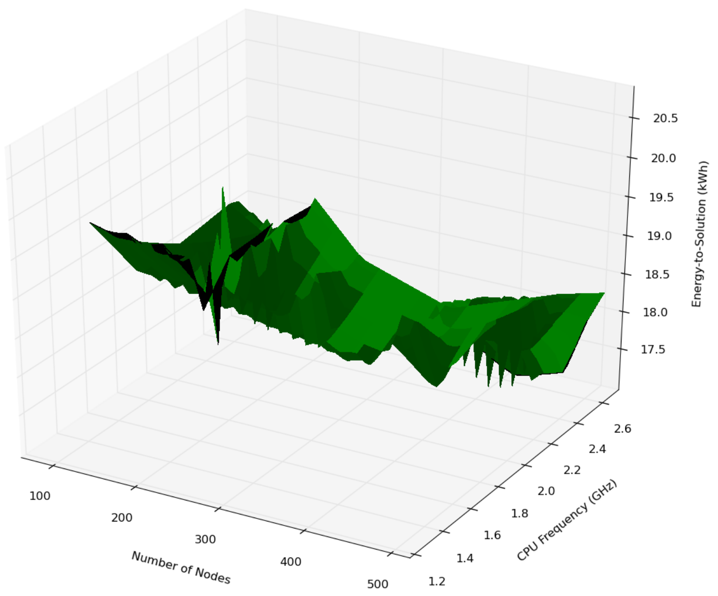


Figure 4.1.: EtS profile of HYDRO with different numbers of utilized compute nodes and CPU frequencies

In reality, these two metrics could be different for the same application when executed

☆This chapter is partly based on the following previous work of the author: *Hayk Shoukourian et al. Predicting Energy Consumption Relevant Indicators of Strong Scaling HPC Applications for Different Compute Resource Configurations. Proceedings of the* 23rd *High Performance Computing Symposium, Society for Modeling and Simulation International (SCS),* 2015.

with different Resource Configuration (RC), where RC is defined as a pair $(a;b)$ - a representing the number of compute resources and b indicating the maximum CPU frequency of the compute resources. Figure 4.1 and Figure 4.2 show this behavior for the EtS metric case for two different real-world HPC applications - correspondingly for HYDRO and EPOCH. As can be seen the EtS behaviors of HYDRO as well as EPOCH notably vary with different RC.

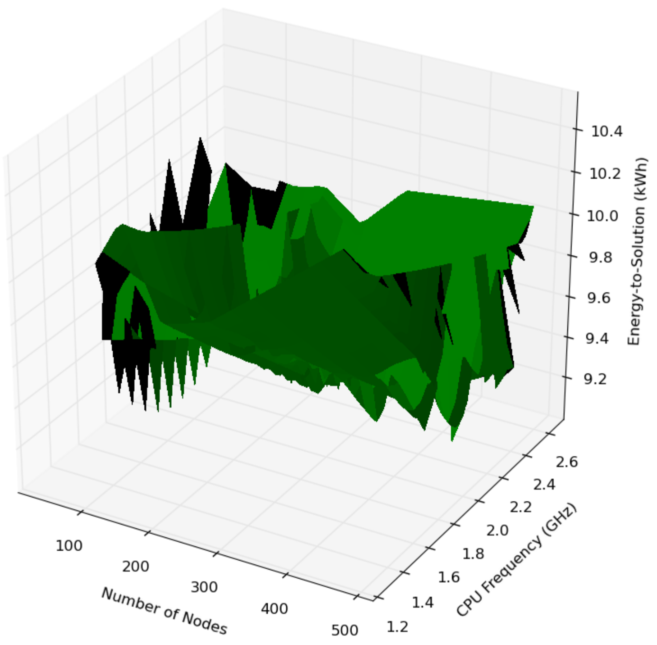


Figure 4.2.: *EtS* profile of EPOCH with different numbers of utilized compute nodes and CPU frequencies

This chapter will extend the previously developed AEPCP process to enable the development of analytical models for estimating the TtS, APC, and EtS metrics for large-scale applications, demonstrating strong scaling, as functions of RC. Based on these analytical models, an adaptive model, referred to as Lightweight Adaptive Consumption Predictor (LACP) will be presented that implements the extended prediction process. The LACP model allows for improved estimation of potential energy-performance costs and trade-offs of various applications and also, as will be seen in Chapter 6, assists in identifying the optimal RC for given applications with regard to specific data center boundary conditions.

4.2. Modeling Application KPIs: State of the Art and Perspectives

Ge et al. [91] suggest analytical models to estimate performance and energy consumption for a given parallel application on multi-core based power aware systems. The authors in [91] further split the serial $(1-p)$ fraction of the application (Section 3.2) into processor frequency dependent fraction: $(1-p)\alpha_s$, i.e. fraction that benefits from faster processor speed; and processor frequency independent fraction: $(1-p)(1-\alpha_s)$, i.e. fraction that does not benefit from the processor speed [1]. Similarly, the application parallel fraction p is split into processor frequency dependent fraction: $p\alpha_p$; and processor frequency independent fraction: $p(1-\alpha_p)$. Based on these notations, Ge et al. approximate the TtS execution time of an application for given n total number of cores, c per compute node allocated number of cores, and f processor frequency as follows:

$$TtS(n,c,f) = (1-p)(1-\alpha_s+\alpha_s\frac{f_0}{f}) + \frac{p}{n}(1-\alpha_p+\alpha_p\frac{f_0}{f}) + O_{n,c,f} \tag{4.1}$$

where f_0 is the base CPU frequency and $O_{n,c,f}$ is the parallelization overhead. Since for most supercomputers the compute node is the smallest compute resource a job can use, it is assumed that the number of cores c per compute node is constant and, therefore, the term c is omitted from further discussions and n is further considered as the number of compute nodes.

Although Ge et al. [91] derive the analytical models for an application execution time, power and energy consumptions, as well as investigate the application energy-performance efficiency, the derivation process of the parameters presented in their analytical models is not automated [91] and in general should be derived through linear regression using the data obtained from experimental benchmarks. It is worth noting that, for example, in order to derive the parallelizable and not parallelizable, or frequency dependent and independent fractions of a given parallel application, a usage of extra tools would be required to detect these application fractions. This process, besides causing execution time penalties/overheads for individual applications, might be impractical in real-world scenarios where typically myriad of applications with different characteristics are used.

In order to mitigate this problem, a further refinement of the application TtS, APC, and EtS metrics is presented. These refined functions will approximate the the mentioned three TtS, APC, and EtS metrics behaviors explicitly from the n number of compute nodes and f maximum operating CPU frequency (at which all the cores of all n compute nodes operate) and will treat the parallelizable and not parallelizable (as well as frequency dependent and independent) fractions of the application as constant fitting parameters.

Equation 4.1 can be approximated as [92]: TtS$(n,f) \in \mathrm{O}\big(1\big) + \mathrm{O}\big(\frac{1}{f}\big) + \mathrm{O}\big(\frac{1}{n}\big) + \mathrm{O}\big(\frac{1}{nf}\big) + \mathrm{O}\big(O_{n,f}\big)$. The term $O_{n,f}$, representing the parallel slowdown, can be further broken down and approximated using the following observations. First, the overhead $O_{n,f}$ arises due to the parallelizable fraction of the application. Second, during the execution of the parallelizable and frequency independent fraction of the application the resulting parallel overhead of n compute nodes is of the order of $\mathrm{O}\big(n\big)$. Whereas during the execution of the parallelizable and frequency dependent fraction the parallel overhead is of the order of $\mathrm{O}\big(\frac{n}{f}\big)$, since the overhead time gets reduced as the processor frequency increases. Thus complete parallel slowdown $O_{n,f}$ is of $\mathrm{O}\big(n+\frac{n}{f}\big)$ order, which further means that TtS$(n,f) \in \mathrm{O}\big(\frac{1}{f}+\frac{1}{n}+\frac{1}{nf}+n+\frac{n}{f}\big)$. Therefore, the TtS$(n,f)$ for given n compute nodes and f CPU frequency can be approximated as follows:

$$\mathrm{TtS}(n,f) = \frac{t_1}{f} + \frac{t_2}{n} + \frac{t_3}{nf} + t_4 n + t_5\frac{n}{f} + t_6 \tag{4.2}$$

where all t_i $(1 \le i \le 6)$ are constant fitting parameters.

[1]Some of the terminology defined in [91] has been borrowed for consistency reasons.

The proposed by Ge et al. [91] analytical model for estimating the node power consumption requires additional estimations of dynamic to idle power scaling factors (which are application and target hardware platform dependent). In addition, the model does not explicitly show the dependency from the operating f CPU frequency of the compute node. In order to cover these gaps, the following observations have been done. First, since CPU and memory are the major power consuming resources in a typical compute node [93], and also the ones, which could potentially vary with processor frequency, they are further treated individually as also done in [91]. The break down of one compute node average power consumption $APC(1, f)$ for a given frequency f can be done in the following way:

$$\text{APC}(1, f) = P_{CPU,dynamic} + P_{CPU,idle} + P_{memory,dynamic} + P_{memory,idle} + P_{other} \tag{4.3}$$

where $P_{CPU,dynamic}$ indicates the power sum of all cores of the compute node when all cores execute some workload; $P_{CPU,idle}$ quantifies the power sum of all cores of the compute node when there is no application running on any of the cores; $P_{memory,dynamic}$ represents the power consumption of compute node dynamic memory; $P_{memory,idle}$ represents the power consumption of compute node memory when it is idling; and P_{other} shows the aggregated power consumption of the other components of the compute node such as hard drives, Peripheral Component Interconnect (PCI) slots, etc. It is further assumed, that the CPU frequency modification is accomplished through the CPU multiplier and not through the processor base clock, thus this modification has no any impact on the operating memory frequency, and thus also on power consumption of memory, as well as on the power consumption of other compute node components.

The power consumption of an operating CPU at a given frequency f can be summarized through the following equation (as shown in [24]):

$$\begin{aligned} P_{cpu,dynamic} &= P_{switching} + P_{short-circuit} + P_{static} \\ &= \alpha C_L V_{dd}^2 f + P_{short-circuit} + P_{static} \end{aligned} \tag{4.4}$$

where α is the switching activity factor, i.e. the probability of the circuit node transition from 0 to 1; C_L is the load capacitance; V_{dd} is the supply voltage; and $P_{short-circuit}$ is the power leakage aroused due to the short-circuit current flowing from supply to ground when both P-type Metal-Oxide-Semiconductor (pMOS) and N-type Metal-Oxide-Semiconductor (nMOS) stacks are conducted. A survey conducted by H. Veendrick [94] provides an in-depth analysis of $P_{short-circuit}$ power dissipation. The P_{static} is the static power dissipation of the processor aroused due to the subthreshold leakage, the gate leakage, the junction leakage, and the contention current [24].

Due to the fact that, in the most of the current processor generations, the supply voltage V_{dd} linearly depends on the operating frequency f [95], the following is deducted: $P_{switching} \in \mathrm{O}\big(f^3\big)$. Since the dynamic power is usually dominated by $P_{switching}$ power of charging and discharging load capacitances as gates switch [24, 96], the $P_{short-circuit}$ and P_{static} can be considered as constant in this case, i.e. $P_{short-circuit} \in \mathrm{O}\big(1\big)$, $P_{static} \in \mathrm{O}\big(1\big)$ and thus $P_{cpu,dynamic} \in \mathrm{O}\big(f^3 + 1\big)$, leading to:

$$P_{cpu,dynamic} = a_1 f^3 + a_2 \tag{4.5}$$

where a_1, and a_2 are constant fitting parameters. On the other hand, when a processor is idle, the power dissipation is determined by the leakage, i.e. P_{static} [24]. Due to the DVFS done by the PCU of energy-saving features enabled processors [14], the power supply voltage of the processor is always reduced during the idling periods (thus also the frequency) to an optimal point, and hence, it can be assumed that $P_{static} \in \mathrm{O}\big(1\big)$. Therefor by using Equation 4.3 the average power consumption of one node can be derived as $APC(1,f) \in \mathrm{O}\big(f^3+1\big)$ for a given CPU frequency f, and the average power consumption of n nodes, i.e. $APC(n,f)$, for a given global frequency f will be of order $\mathrm{O}\big(nf^3+n\big)$, which further means that:

$$\mathrm{APC}(n,f) = k_1 n f^3 + k_2 n + k_3 \tag{4.6}$$

where k_1, k_2, and k_3 are constant fitting parameters. Below the summary of the TtS, APC, and EtS approximations for each fraction (as defined above) of a given application:

- ***fraction 1 - serial and frequency dependent.*** For this fraction $\mathrm{TtS}_1(n,f) \in \mathrm{O}\big(\frac{1}{f}\big)$[2]. Since only 1 node is active during the application serial fraction execution and the rest $n-1$ nodes are idling and thus consuming $\mathrm{O}\big(1\big)$ power, $APC_1(n,f) \in \mathrm{O}\big(f^3 + (n-1)\big) = \mathrm{O}\big(f^3+n\big)$, and thus $EtS_1(n,f) = TtS_1(n,f)APC_1(n,f) \in \mathrm{O}\big(f^2 + \frac{n}{f}\big)$;

- ***fraction 2 - serial and frequency independent.*** For this fraction $\mathrm{TtS}_2(n,f) \in \mathrm{O}\big(1\big)$[2]. In this case, $\mathrm{APC}_2(n,f) = \mathrm{APC}_1(n,f) \in \mathrm{O}\big(f^3+n\big)$, thus $\mathrm{EtS}_2(n,f) \in \mathrm{O}\big(f^3+n\big)$;

- ***fraction 3 - parallel and frequency dependent.*** For this fraction $\mathrm{TtS}_3(n,f) \in \mathrm{O}\big(\frac{1}{nf}\big)$[2]. In this case, all n compute nodes are active and consuming $\mathrm{O}\big(nf^3+n\big)$ power in total. Thus the energy consumption for a given number n of compute nodes and CPU frequency f, is given by: $\mathrm{EtS}_3(n,f) = \mathrm{O}\big(f^2 + \frac{1}{f}\big)$;

- ***fraction 4 - parallel and frequency independent.*** For this fraction $\mathrm{TtS}_4(n,f) \in \mathrm{O}\big(\frac{1}{n}\big)$[2], and $\mathrm{APC}_4(n,f) = \mathrm{APC}_3(n,f) = \mathrm{O}\big(nf^3+n\big)$. Thus for a given n number of compute nodes and f global CPU frequency the $\mathrm{EtS}_4(n,f)$ energy consumption for this case will be of order $\mathrm{O}\big(f^3+1\big)$;

- ***fraction 5 - parallel overhead.*** For this fraction $\mathrm{TtS}_5(n,f) \in \mathrm{O}\big(\frac{n}{f}+n\big)$[3]. Since all n compute nodes could be active during this period, the total power $\mathrm{APC}_5(n,f)$ and energy $\mathrm{EtS}_5(n,f)$ consumptions for the parallel overhead fraction for given n compute nodes when all are running at a given frequency f is: $\mathrm{APC}_5(n,f) = \mathrm{APC}_4(n,f) = \mathrm{APC}_3(n,f) \in \mathrm{O}\big(nf^3+n\big)$; $\mathrm{EtS}_5(n,f) = \mathrm{O}\big(n^2f^2 + \frac{n^2}{f} + n^2f^3 + n^2\big)$.

Thus the total EtS energy consumption of an application (representing the sum of all the five above listed fractions) for given n compute nodes when all the cores of which are running at a given frequency f can be approximated through the following equation:

[2]Implied from Equation 4.1.
[3]As discussed for Equation 4.2.

$$\begin{aligned}\mathrm{EtS}(n,f) = \sum_{i}^{5} \mathrm{EtS}_i(n,f) =& b_1 f^2 + b_2 \frac{n}{f} + b_3 f^3 + b_4 n + \frac{b_5}{f} + b_6 + \\ & b_7 n^2 f^2 + b_8 \frac{n^2}{f} + b_9 n^2 f^3 + b_{10} n^2\end{aligned} \tag{4.7}$$

where all b_i $(1 \leq i \leq 10)$ are constant fitting parameters.

4.2.1. Analysis of the Energy Model

In case an application demonstrates ideal scaling (i.e. when the application parallelizable fraction p equals to 1 and thus the serial fraction $(1-p)$ becomes 0) then the $\mathrm{EtS}_1 = \mathrm{EtS}_2 = 0$. When assuming also that the parallel slowdown (fraction 5) is negligible during the ideal scaling (i.e. $\mathrm{EtS}_5 = 0$), Equation 4.7 can be rewritten in the following way:

$$\mathrm{EtS}(n,f) = \mathrm{EtS}_3(n,f) + \mathrm{EtS}_4(n,f) = c_1 f^3 + c_2 f^2 + \frac{c_3}{f} + c_4 \tag{4.8}$$

where all c_i $(1 \leq i \leq 4)$ are constants. This further means that, in the case of ideal scaling, the usage of more compute nodes would potentially increase the performance (i.e. the TtS runtime, since in the case of ideal scaling: $\mathrm{TtS}(n,f) \in \mathrm{O}\left(\frac{1}{nf} + \frac{1}{n}\right)$) without any additional energy costs, as also mentioned in [91] and in Subsection 3.2.1. Meantime, the increase in CPU frequency would potentially bring a cubic rise in the energy costs.

Although application TtS improvement reduces the application EtS consumption, the further adjustment of TtS via CPU frequency scaling can still improve the performance by trading-off the energy consumption. Figure 4.3 illustrates this scenario for the case of a

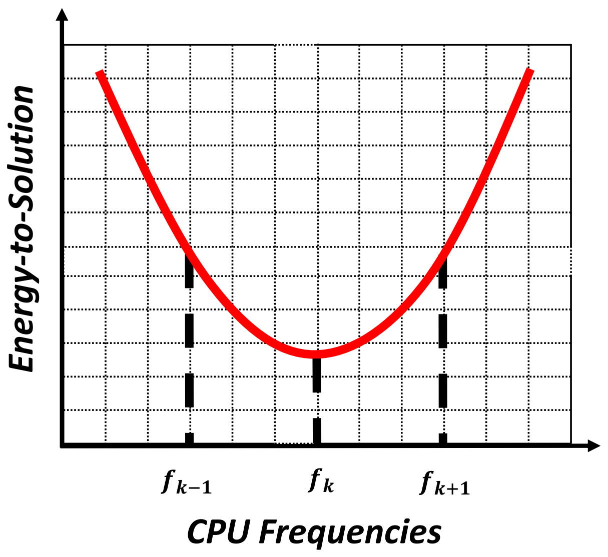


Figure 4.3.: The EtS behavior from the operating CPU frequency in case of a fixed n number of compute nodes

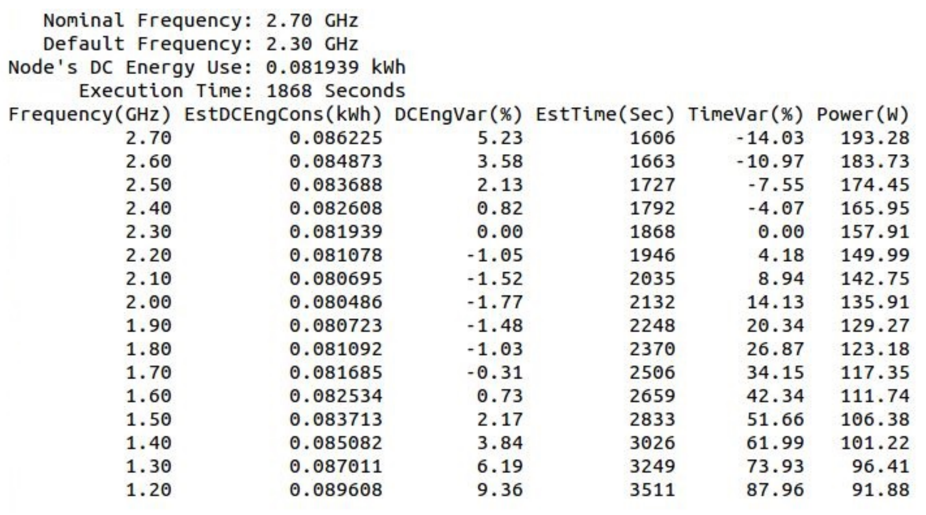
fixed n number of application to-be-utilized compute nodes, when the TtS can be approximated as[4] $\frac{k_1}{f}$ and APC as $k_2 \cdot f^3$, leading to the EtS = TtS $\cdot$ APC $= k_3 \cdot f^2$ approximation, where all the k_i ($1 \leq i \leq 3$) are constant fitting parameters.

Consider three different CPU frequencies: $f_{k-1} < f_k < f_{k+1}$ as illustrated in Figure 4.3. Since $f_k < f_{k+1}$ then $TtS(n, f_k) > TtS(n, f_{k+1})$, while as can be seen from the Figure 4.3, $EtS(n, f_k) < EtS(n, f_{k+1})$, which further means that the *decrease of the **execution time** does not lead to a corresponding decrease in **energy** consumption*. On the other hand, since $f_{k-1} < f_k$ then $APC(n, f_{k-1}) < APC(n, f_k)$, while, as can be seen from the Figure 4.3, $EtS(n, f_{k-1}) > EtS(n, f_k)$, meaning that the *decrease of the **average power** consumption does not necessarily lead to corresponding decrease in **energy** consumption* either.

4.3. IBM LoadLeveler Alone

IBM LoadLeveler [58] is the RMSS used in the SuperMUC supercomputer. LoadLeveler has a prediction model [35] for estimating the application runtime, power and energy consumptions when executed using different CPU frequencies for a fixed number of compute nodes. A recent study found in [28] shows the high accuracy rate of the predictions accomplished through the LoadLeveler model.

Figure 4.4 presents a sample LoadLeveler prediction output for the HYDRO application-benchmark, when it was executed using 210 compute nodes. As can be seen, for a given 210 compute node count execution of HYDRO, LoadLeveler estimates the per node energy consumption (second column, i.e. *"EstDCEngCons(kWh)"*), the total application runtime (fourth column, i.e. *"EstTime(Sec)"*), and the per node average power consumption (the last, sixth column, i.e. *"Power (W)"*) for all platform supported operating CPU frequencies.



```
    Nominal Frequency: 2.70 GHz
    Default Frequency: 2.30 GHz
 Node's DC Energy Use: 0.081939 kWh
       Execution Time: 1868 Seconds
Frequency(GHz) EstDCEngCons(kWh) DCEngVar(%) EstTime(Sec) TimeVar(%) Power(W)
          2.70          0.086225        5.23         1606     -14.03   193.28
          2.60          0.084873        3.58         1663     -10.97   183.73
          2.50          0.083688        2.13         1727      -7.55   174.45
          2.40          0.082608        0.82         1792      -4.07   165.95
          2.30          0.081939        0.00         1868       0.00   157.91
          2.20          0.081078       -1.05         1946       4.18   149.99
          2.10          0.080695       -1.52         2035       8.94   142.75
          2.00          0.080486       -1.77         2132      14.13   135.91
          1.90          0.080723       -1.48         2248      20.34   129.27
          1.80          0.081092       -1.03         2370      26.87   123.18
          1.70          0.081685       -0.31         2506      34.15   117.35
          1.60          0.082534        0.73         2659      42.34   111.74
          1.50          0.083713        2.17         2833      51.66   106.38
          1.40          0.085082        3.84         3026      61.99   101.22
          1.30          0.087011        6.19         3249      73.93    96.41
          1.20          0.089608        9.36         3511      87.96    91.88
```

Figure 4.4.: Sample output of LoadLeveler CPU prediction results for HYDRO when executed on 210 compute nodes

It can be further argued that the information regarding the application's TtS, APC, and

[4]It is assumed that the application has a non-zero processor frequency dependent fraction, since the opposite would indicate that there is no any computation routine within the application.

EtS with respect to a given compute node count and operating CPU frequency can be easily derived from the above data by multiplying these LoadLeveler results with corresponding numbers of compute nodes. Table 4.1 shows that this "simple-multiplication" approach leads to a low prediction accuracy.

The *first* column of Table 4.1 shows the compute node count used for comparison; the *second* column shows the CPU frequencies; the *third* column shows the measured energy consumption for a given resource configuration; the *fourth* column shows the predicted energy consumption using the above mentioned multiplication method for the corresponding resource configuration; and finally, the *fifth* column shows the prediction error rate of the used method calculated as: $\frac{|measured - predicted|}{measured} * 100$.

Number of Compute Nodes	**CPU Frequency (GHz)**	**Measured EtS Value (kWh)**	**LoadLeveler Predicted EtS Value (kWh)**	**Prediction Error Rate (%)**
150	1.6	18.16	12.38	31.8
	1.8	17.75	12.16	31.49
	2.3	17.76	12.29	30.79
	2.7	18.66	12.93	30.7
210	1.6	17.65	17.33	1.8
	1.8	17.21	17.03	1.05
	2.3	17.21	17.21	0
	2.7	17.97	18.11	0.78
320	1.6	18.48	26.41	43
	1.8	18.31	25.95	41.7
	2.3	17	26.22	54.23
	2.7	17.57	27.6	57

Table 4.1.: Prediction results for HYDRO using data obtained from IBM LoadLeveler for 210 compute nodes

The "$- - -$" highlighted circle in Table 4.1 illustrates the LoadLeveler predicted data for 210 compute nodes. The rows above and below the "$- - -$" highlighted circle in Table 4.1 show the predicted EtS (using the above mentioned "simple-multiplication" approach) for HYDRO when executed using 150 and 320 compute nodes[5] at different operating CPU frequencies. As can be seen, the suggested multiplication approach has a prediction error rate of more than 30% for less/greater than 210 count of compute nodes.

It can be further argued that for a fixed frequency f, the knowledge on application execution time with n compute nodes can be derived from the knowledge of application execution time with k compute nodes using the $p = 1$ special case of Amdahl's law, shown in Equation 3.2 (i.e. assuming that the considered application is completely parallelizable). Equation 4.9 shows this derivation, where the $\text{TtS}_{measured}(k, f)$ is the known application execution time with k compute nodes at frequency f.

$$\text{TtS}_{predicted}(n, f) = \frac{\text{TtS}_{measured}(k, f) \cdot k}{n} \tag{4.9}$$

Note, that according to Amdahl's law, in case of $p = 1$, the numerator of the fraction on the right hand side of the Equation 4.9 represents the application execution time with 1 compute node, i.e. $\text{TtS}(1)$.

[5]These node counts were chosen on a random basis.

Since LoadLeveler predicts the per node APC of an application when it is executed using k nodes (denoted as $\text{APC}_{measured}(k, f)_{for\ one\ node}$), the knowledge on application APC with n compute nodes can be derived by just multiplying this $\text{APC}_{measured}(k, f)_{for\ one\ node}$ by n count of compute nodes. Equation 4.10 shows this approximation.

$$\text{APC}_{predicted}(n, f) = \text{APC}_{measured}(k, f)_{for\ one\ node} \cdot n \tag{4.10}$$

EtS, being the product of TtS and APC, can be then calculated using the Equation 4.11.

$$\text{EtS}_{predicted}(n, f) = \text{TtS}_{predicted}(n, f) \cdot \text{APC}_{predicted}(n, f) \tag{4.11}$$

Table 4.2, Table 4.3, and Table 4.4 correspondingly show the TtS, APC, and EtS prediction results, for compute node counts[6] 105 and 510, correspondingly estimated through Equation 4.9, Equation 4.10, and Equation 4.11 using the LoadLeveler predicted data for 210 compute node count.

Number of Compute Nodes	CPU Frequency (GHz)	Measured TtS Value (sec)	LoadLeveler Based TtS Predicted Value (sec)	Prediction Error Rate (%)
105	2.3	3913	3736	4.52
210	2.3	1868	1868	0
510	2.3	859	769.17	10.45

Table 4.2.: Revisiting the IBM LoadLeveler based node scaling prediction for TtS

Number of Compute Nodes	CPU Frequency (GHz)	Measured APC Value (W)	LoadLeveler Based APC Predicted Value (W)	Prediction Error Rate (%)
105	2.3	17751.93	16580.55	6.6
210	2.3	33218.34	33161.1	0.17
510	2.3	74071	80534.1	8.72

Table 4.3.: Revisiting the IBM LoadLeveler based node scaling prediction for APC

Number of Compute Nodes	CPU Frequency (GHz)	Measured EtS Value (kWh)	LoadLeveler Based EtS Predicted Value (kWh)	Prediction Error Rate (%)
105	2.3	19.27	17.2	10.7
210	2.3	17.2071	17.20719	0
510	2.3	17.6741	17.2067	2.64

Table 4.4.: Revisiting the IBM LoadLeveler based node scaling prediction for EtS

Although, a drastic prediction quality improvement can be observed (as compared to the earlier suggested "simple-multiplication" method), Figure 4.5 shows that this second approach will also not be applicable in real world scenarios.

Indeed, assume that an application was executed using 100 compute nodes and the LoadLeveler TtS, APC, and EtS prediction data is available for any $(100, f)$ RC. Following

[6]These node numbers were chosen on a random basis.

the Equation 4.9, the execution time of this application for 200 compute nodes at a given f CPU frequency can be approximated as:

$$\mathrm{TtS}(200, f) = \frac{\mathrm{TtS}(100, f)}{2} \tag{4.12}$$

While this approximation will provide a relatively low error rate estimations in case the application continues to scale (i.e. record TtS reductions as the number of utilized compute nodes increases - $\mathrm{TtS}(100, f) > \mathrm{TtS}(200, f)$, blue solid line, Figure 4.5), it will end up with relative high prediction error rate (for this case with a factor of 2) in case the application demonstrates relative low scalability with compute node counts higher than 100, i.e. in the case $\mathrm{TtS}(100, f) \approx \mathrm{TtS}(200, f)$ (dashed line, Figure 4.5). In other words, the TtS approximation method shown in Equation 4.9 is not applicable in real-world scenarios, since in most of the cases, it won't be able to correctly derive the application scalability behavior only from a one TtS data point.

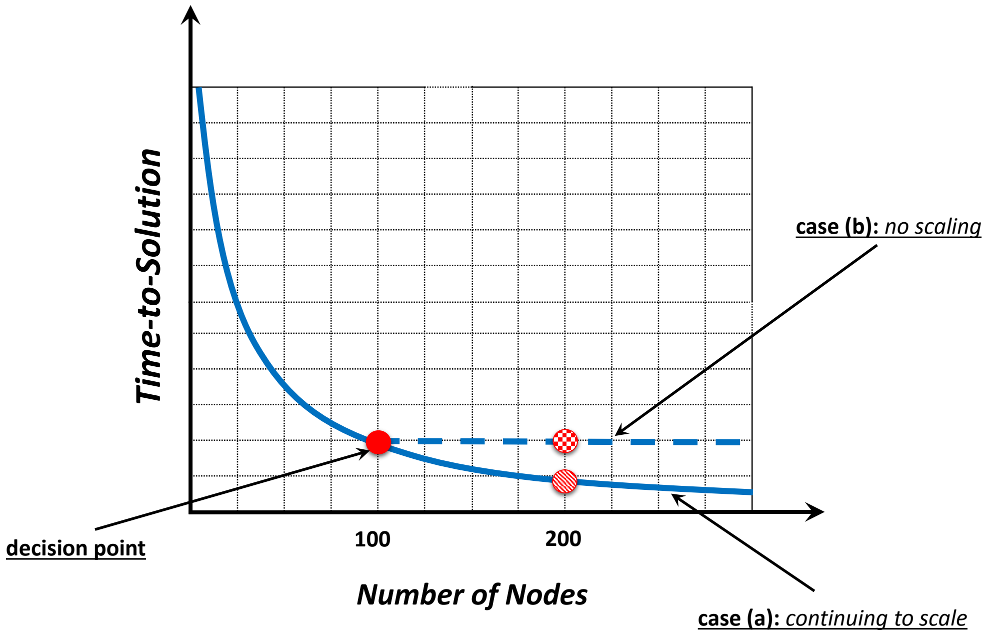


Figure 4.5.: TtS scaling scenarios

4.4. Lightweight Adaptive Consumption Predictor (LACP) Model

Figure 4.6 shows the prediction process of the LACP. Similarly to the AEPCP process, the LACP process takes as input:

- **application identifier** used for uniquely identifying the application;
- **number of compute resources** planned for application utilization; and in addition to AEPCP process inputs the

- **maximum CPU frequency** at which all the cores of the compute resources will operate.

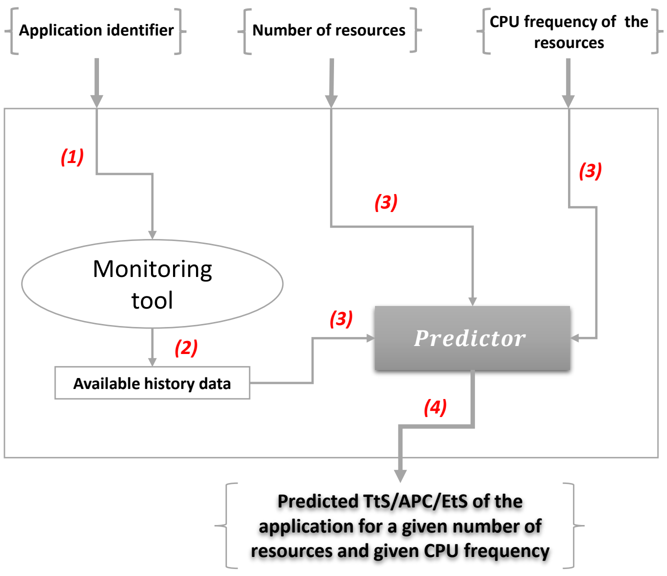

Application identifier
Number of resources
CPU frequency of the resources
(1)
(3)
(3)
Monitoring tool
(3)
Predictor
(2)
Available history data
(4)
Predicted TtS/APC/EtS of the application for a given number of resources and given CPU frequency

Figure 4.6.: Overview of the *LACP* process

As shown in Figure 4.6, the application identifier is used for uniquely querying the application execution relevant history information from the system monitoring tool (step 1). Once the history data is obtained (step 2), it, along with the number of compute nodes and the given CPU frequency, is passed to the predictor (step 3) for corresponding application TtS/APC/EtS prediction. Based on this input data, the predictor then estimates the required TtS/APC/EtS consumption values (step 4).

Figure 4.7 presents the *LACP* model [92] based on the prediction process described above. The LACP model takes three inputs, namely:

- **application energy tag prefix** used as an application unique identifier. As mentioned in Chapter 3, the application energy tag is a unique identifier for the application, supported by the LoadLeveler (the RMSS used in SuperMUC), and is specified by the user on a unique-per-application basis.

  A new energy tag could be specified by the user each time an application is submitted for execution with a different compute node configuration. This specification **is optional** and in case it is set, LoadLeveler will generate a corresponding prediction data for that specific node number configuration, thus making the available application-relevant-history data larger which in its turn would potentially lead to an increase in the accuracies of the corresponding predictions. This new tag should have a unique prefix to identify all the existing executions of that application. For example, $myAutoCrashSimulationETag\_178nodes$ and $myAutoCrashSimulationETag\_317nodes$ could be two energy tags for an application having a unique $myAutoCrashSimulationETag$ prefix identifying them both;

- **number of compute nodes** used as the number of compute resources; and
- **maximum CPU frequency** enforced for all used compute nodes.

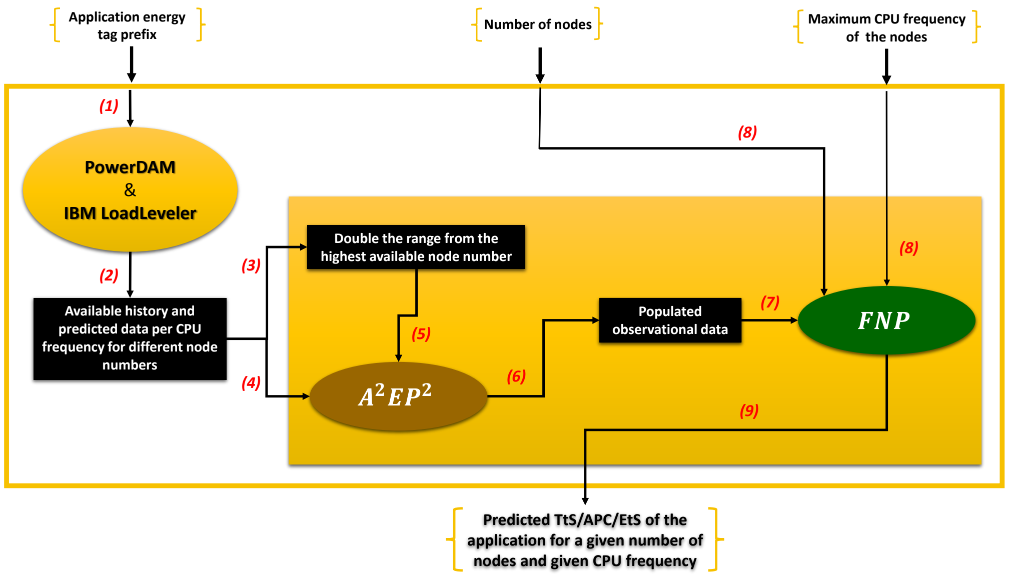

Application energy tag prefix
Number of nodes
Maximum CPU frequency of the nodes
(1)
PowerDAM & IBM LoadLeveler
(2)
Available history and predicted data per CPU frequency for different node numbers
(3)
(4)
Double the range from the highest available node number
(5)
A^2EP^2
(6)
Populated observational data
(7)
(8)
(8)
FNP
(9)
Predicted TtS/APC/EtS of the application for a given number of nodes and given CPU frequency

Figure 4.7.: Overview of the *LACP* model

As shown in Figure 4.7, the application energy tag prefix is used to uniquely retrieve the historical TtS, APC, and EtS relevant data of the application executions for different configurations from PowerDAM (step 1). Next, the available LoadLeveler prediction data is compiled with this historical data (step 2). As was mentioned above, this step is optional, and is used only for further population of the available history data, since the latter would potentially improve the prediction accuracy.

Once this application-relevant (compiled) history data is obtained, the maximum and minimum number of compute nodes are determined within this data, correspondingly doubled and halved (step 3), and together with the history data, passed to the A^{2}EP^2 (step 4 and step 5). Using this data, A^{2}EP^2 extends this data set in the following way. First, A^{2}EP^2 was extended to allow for the TtS metric prediction, using the fact that, in the case of a fixed CPU frequency, the execution time TtS(n) for n number of compute nodes, for the strong scaling case, can be approximated as TtS$(n) = \frac{a_1}{n} + a_2$, where a_1 and a_2 are constant fitting parameters[7]. This is achieved by using an interpolation technique to estimate the a_1 and a_2 constants in TtS(n) with respect to different compute node counts. Second, for each target platform supported CPU frequency f, A^{2}EP^2 predicts the TtS, APC, and EtS for all unknown number of compute nodes[8] which are in the range of $[\frac{min\ observed\ node\ number}{2}; 2 \times max\ observed\ node\ number]$. The double/half distance for prediction is used, since the A^{2}EP^2 has been proven to provide acceptable results within that range (Chapter 3).

This extended data (step 6) is then passed to the *Frequency and Node Number Predictor (FNP)* (step 7). Then the FNP, using the derived analytical behaviors of TtS (Equa-

[7]Implies from Amdahl's Law (Equation 3.2).
[8]In other words, number of compute nodes which are not in the available to A^{2}EP^2 input data set.

tion 4.2), APC (Equation 4.6), and EtS (Equation 4.7) metrics, estimates the values of the constant fitting parameters using polynomial multivariate interpolation [97]. The R programming language [98] was used for conducting the mentioned multivariate interpolations. Once these coefficients are determined, the FNP calculates the TtS/APC/EtS values of the application for a given n number of compute nodes and given f CPU frequency (steps 8 and 9).

4.5. LACP Validation

This section will show the validity and the adaptability of the developed LACP model, through the experimental results taken on real-world scientific HPC application, using randomly selected historical data points.

Assume, that a user has executed HYDRO three times, with the same input problem size using: 105; 170; and 240 compute nodes when all were operating at 2.3 GHz CPU frequency. Table 4.5 summarizes these three HYDRO execution results.

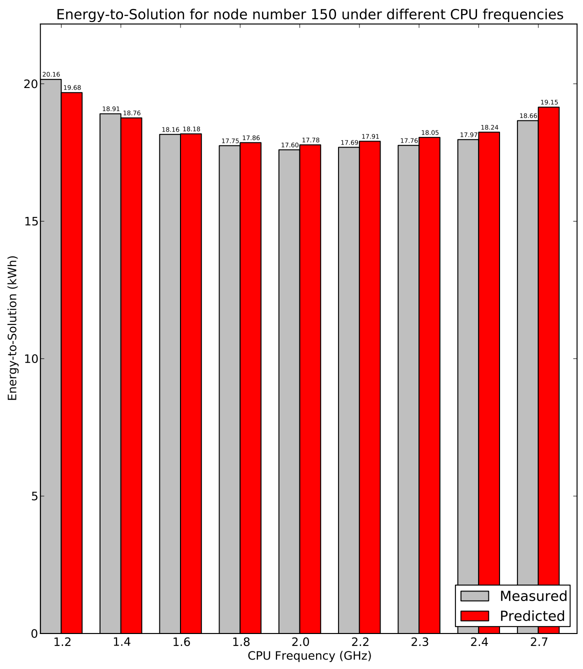
Number of Compute Nodes	**Maximum CPU Frequency (GHz)**	**Measured EtS Value (kWh)**	**Measured APC Value (W)**	**Measured TtS Value (min)**
105	2.3	19.28	17736.55	65.22
170	2.3	17.78	27375	38.98
240	2.3	17.36	37659	27.66

Table 4.5.: PowerDAM tracked HYDRO history data

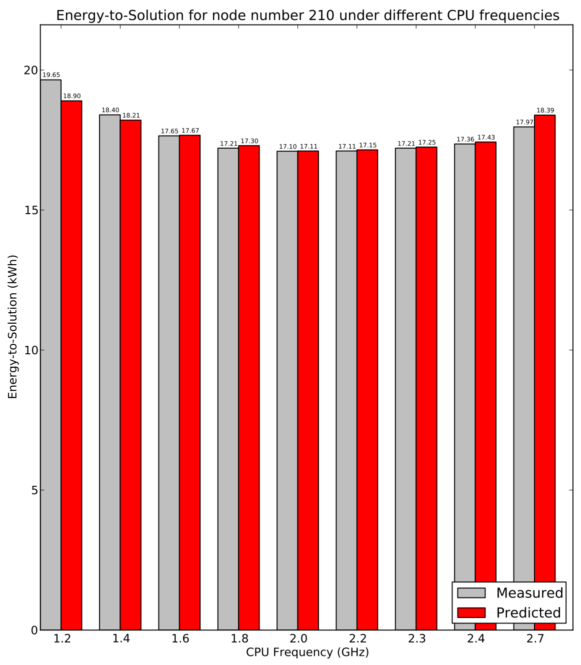
This historical data was further populated using the data obtained from the LoadLeveler (Figure 4.7). Figure 4.8a, Figure 4.8b, and Figure 4.8c show the LACP EtS prediction results for 150, 210, and 450 compute node counts for different CPU frequencies, when using the available, PowerDAM tracked, HYDRO history data shown in Table 4.5. The first bars of these figures (when counted from left to right), colored in gray, show the measured EtS, and the second bars (colored in red) show the LACP predicted EtS. As can be seen, the prediction error is relatively low for 150 and 210 compute node counts - less than 2.7% for 150 compute node count, and less than 4% for 210 compute node count. On contrary, the LACP EtS prediction error rate is relative high for 450 compute node count (Figure 4.8c).

Figure 4.9a, Figure 4.9b, and Figure 4.9c show the LACP HYDRO APC prediction results for the same 150, 210, and 450 compute node counts. The first bars (colored in green) of these figures show the measured APC, and the second bars (colored in yellow) show the LACP predicted APC values. As can be seen, in contrary to EtS prediction results, the LACP APC prediction results have high accuracy for all the three cases, with $(450, 2.7)$ configuration having the highest prediction error of about 7.2%.

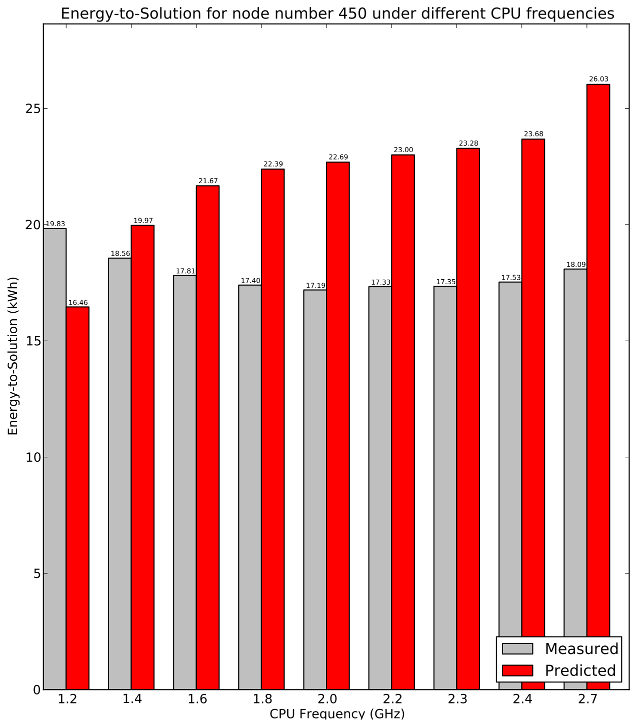
Figure 4.10a, Figure 4.10b, and Figure 4.10c show the LACP HYDRO TtS prediction results for the mentioned 150, 210, and 450 compute node counts, again, when using the available HYDRO history data shown in Table 4.5. The first bars of the mentioned Figures, colored in gray and labeled as *Transitively Predicted (EtS/APC)*, illustrate the prediction results obtained using the data presented in Figure 4.8 and Figure 4.9 (since $\text{TtS} = \frac{\text{EtS}}{\text{APC}}$); the second bars (cored in blue) show the measured TtS values; and finally the third bars (colored in white) labeled as *Directly Predicted* show the TtS prediction case when LACP



(a) For 150 compute nodes



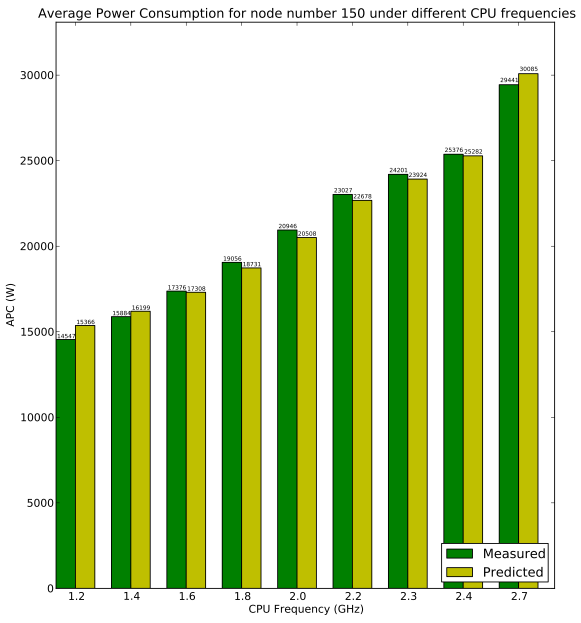
(b) For 210 compute nodes



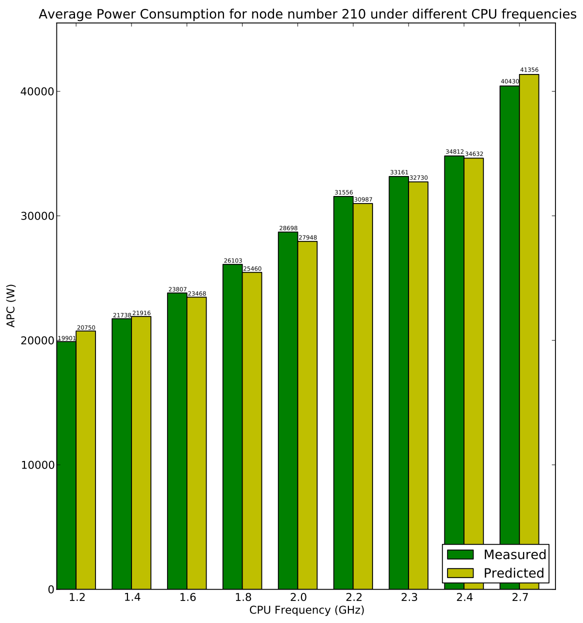
(c) For 450 compute nodes

Figure 4.8.: LACP EtS prediction results for HYDRO

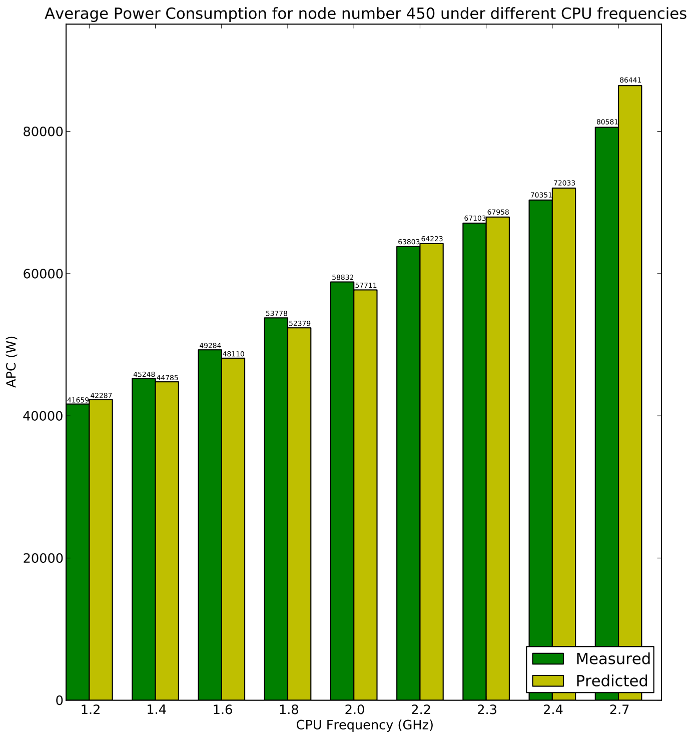
Note: available data points are for compute node counts: 105, 170, *and* 240



(a) For 150 compute nodes



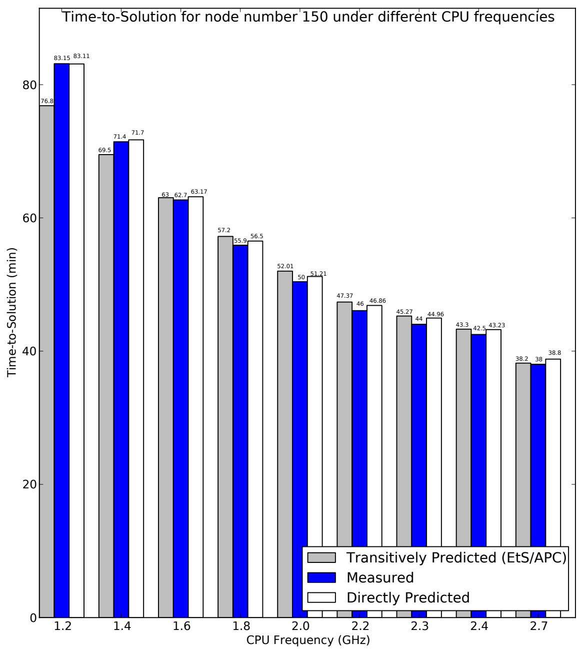
(b) For 210 compute nodes

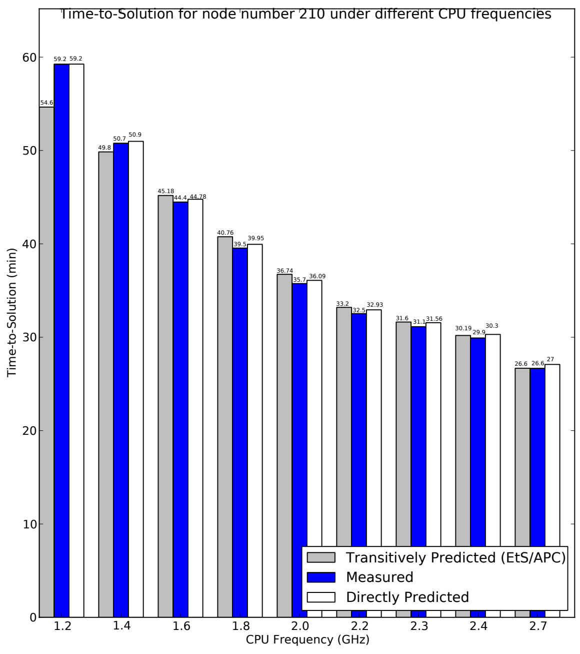


(c) For 450 compute nodes

Figure 4.9.: LACP APC prediction results for HYDRO

Note: available data points are for compute node counts: 105, 170, *and* 240


(a) For 150 compute nodes


(b) For 210 compute nodes

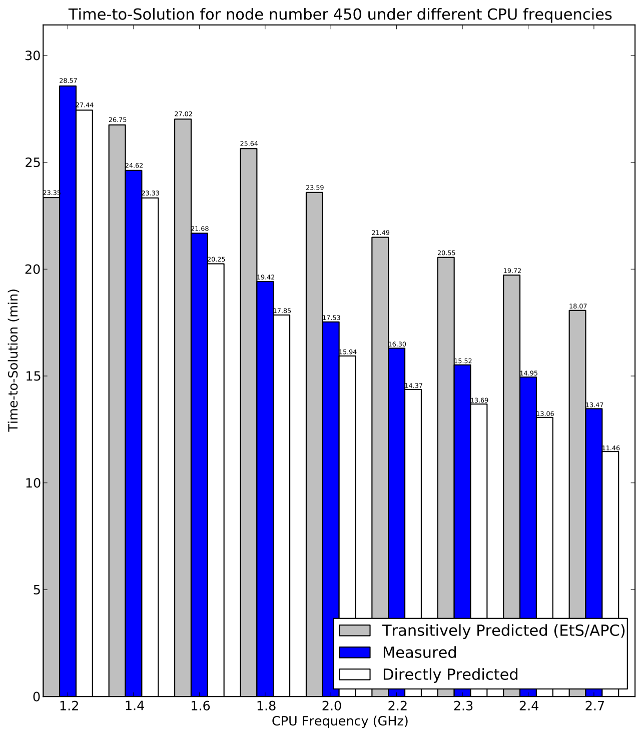
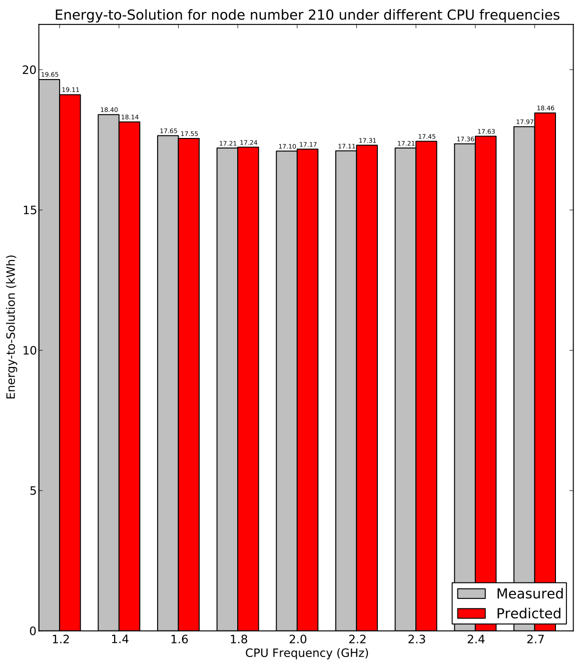

(c) For 450 compute nodes

Figure 4.10.: LACP TtS prediction results for HYDRO

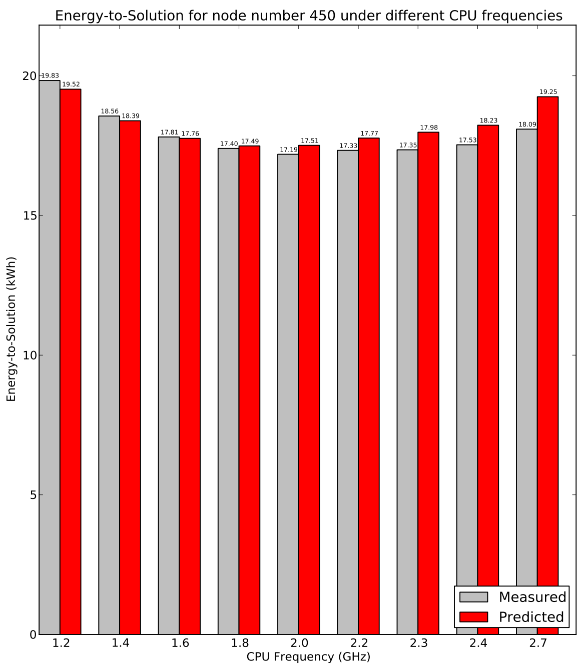
Note: available data points are for compute node counts: 105, 170, *and* 240

uses the available TtS history data from Table 4.5 for the prediction. As can be seen, in the case of 150 and 210 compute node counts, both, transitively and directly TtS predictions show high accuracy rate. Whereas in the case of 450 compute node count, the transitively predicted TtS is overall worse then the directly predicted one, but in summary both show low accuracy rate having more then 10% error in the TtS predictions.

In order to show the adaptability feature of the LACP model, a measurement data for a randomly selected compute node count of 370 (with measured TtS value of 18.38 minutes, APC value of 55817 W, and EtS value of 17.1 kWh) at 2.3 GHz CPU frequency was appended to the available HYDRO history data presented in Table 4.5. Figure 4.11a and Figure 4.11b show the LACP HYDRO EtS prediction results after the extension of the input data with the TtS, APC, and EtS values of 370 compute node HYDRO execution at 2.3 GHz maximum CPU frequency. As can be seen, the prediction accuracy for 210 compute node count stays high, as compared to Figure 4.8b, whereas for the 450 compute node count case, it has evidently improved (as compared to Figure 4.8c) with an overall prediction error rate of less than 6.5%.



(a) For 210 compute nodes

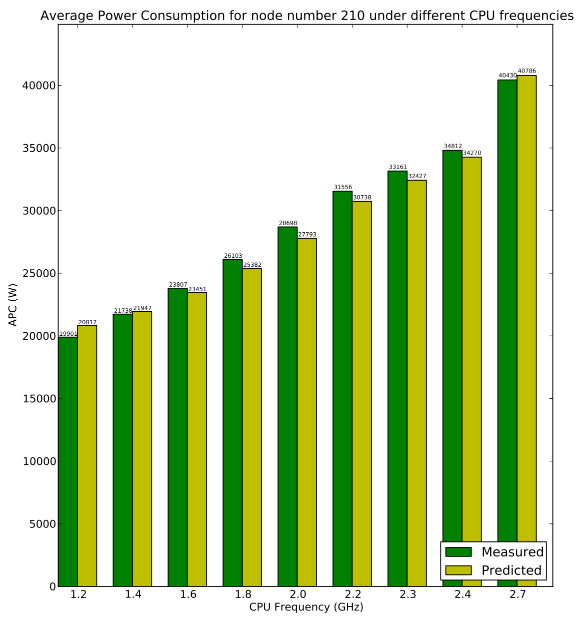


(b) For 450 compute nodes

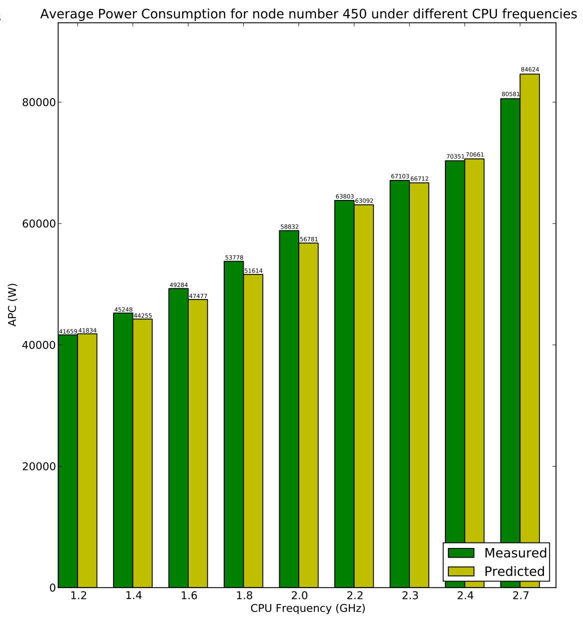
Figure 4.11.: Adapted LACP EtS prediction results for HYDRO

Note: available data points are for compute node counts: 105, 170, 240, *and* 370

Figure 4.12a and Figure 4.12b show the LACP HYDRO APC (and Figure 4.13a and Figure 4.13b show the LACP HYDRO TtS) prediction results for 210 and 450 compute node counts, when additionally to Table 4.5 history data, the APC (and TtS) information regarding the HYDRO execution with $(370, 2.3)$ configuration was made available. As was for the case of the EtS prediction results, the prediction accuracy for 210 compute node counts (for both APC and TtS prediction results) remained high, while the improved overall APC and TtS prediction accuracy for 450 compute node count case.



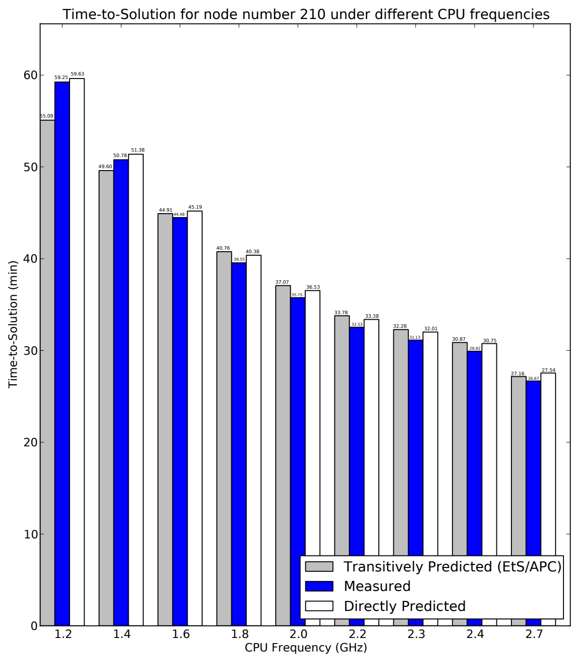
(a) For 210 compute nodes



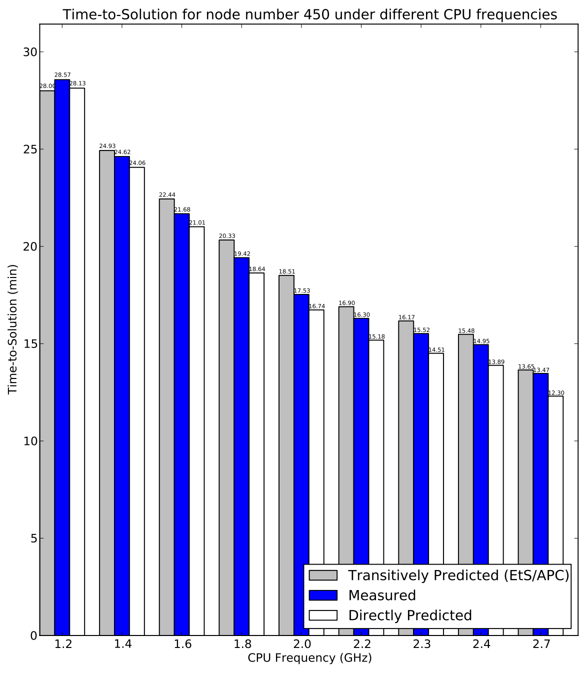
(b) For 450 compute nodes

Figure 4.12.: Adapted LACP APC prediction results for HYDRO

Note: available data points are for compute node counts: 105, 170, 240, *and* 370



(a) For 210 compute nodes



(b) For 450 compute nodes

Figure 4.13.: Adapted LACP TtS prediction results for HYDRO

Note: available data points are for compute node counts: 105, 170, 240, *and* 370

4.5.1. Some LACP Prediction Statistics

HYDRO: In total, 3 EtS, 1 APC, 5 Transitive TtS, and 9 Direct TtS HYDRO predictions out of the 68 measurements showed more than a 10% error, when LACP had available TtS, APC, and EtS information of HYDRO for compute node counts 105, 170, 240, and 370 (all executed at maximum CPU frequency of 2.3 GHz and then populated using the data from LoadLeveler).
EPOCH: In total, 1 EtS, 3 APC, 8 Transitive TtS, and 5 Direct TtS EPOCH predictions out of the 106 measurements showed more than a 10% error, when LACP had available TtS, APC, and EtS information of EPOCH for 74, 112, and 210 compute node counts (all executed at maximum CPU frequency of 2.3 GHz and then populated using the data from LoadLeveler). Some LACP EPOCH prediction results are further discussed in Chapter 5.

Note that due to the shown adaptability feature of the LACP model, the prediction accuracy for a given application is improved with each additional execution (for a new configuration) of that application.

4.6. Power Model in Case of Disabled DVFS Feature

Section 4.2 considered the static power of the idling processor as a constant, since it was assumed that due to the default energy-saving features of the processor, the PCU will reduce the supply voltage during the idling periods. While this usually holds true for the majority of processors used in HPC, this section will also consider the case when the processor does not support the DVFS feature by default; which is, for example, true for ARM Cortex-A15 dual-core processors [99] used in the prototype system being built within the scope of the Mont-Blanc project [100] (Figure 4.14).

A study conducted by Gonzalez et al. [101] describes the static power dissipation of a processor as the product of the supply voltage and the leakage current sum of the all gates:

$$P_{cpu,static} = \sum_i W_i I_s e^{\frac{V_{th}}{V_0}} V_{dd} \tag{4.13}$$

where W_i is the transistor width of the gate i; I_s is the zero-threshold leakage current; V_{th} is the threshold voltage; and V_0 is the subthreshold slope. Blaauw et al. [96] show that the V_{th} threshold voltage depends linearly on V_{dd} source voltage and V_{bs} voltage applied between body and source of the transistor:

$$V_{th} = b_1 - b_2 V_{dd} - b_3 V_{bs} \tag{4.14}$$

where all b_i $(1 \leq i \leq 3)$ are constant fitting parameters. Based on this, Blaauw et al. [96] provide a further approximation of static power dissipation through the following equation:

$$P_{cpu,static} = V_{dd} k_1 e^{k_2 V_{dd}} e^{k_3 V_{bs}} + |V_{bs}|(I_{bn} + I_{jn}) \tag{4.15}$$

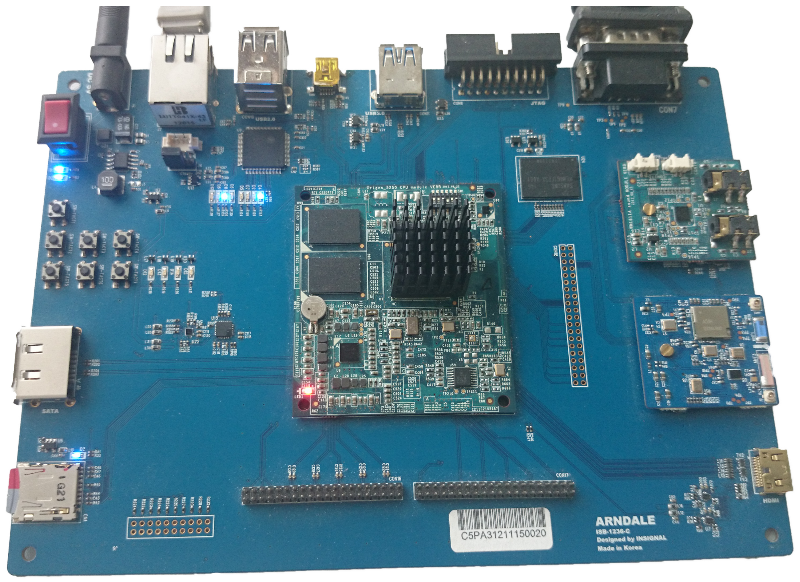


Figure 4.14.: Samsung Exynos 5 Dual Arndale Board featuring a dual-core 1.7 GHz mobile CPU built on ARM Cortex-A15 architecture plus an integrated ARM Mali-T604 GPU for increased performance density and energy efficiency

where I_{bn} is the source to body junction leakage; I_{jn} is the drain to body junction leakage current, and all k_i $(1 \leq i \leq 3)$ are the constant fitting parameters.

Since the V_{th} depends linearly on V_{dd} [102], V_{bs} depends linearly on V_{dd}, which on its turn depends linearly on the f operating processor frequency [95], the $P_{cpu,static}$ static power can be further approximated via the following equation (when also considering the fact that $I_{bn} + I_{jn}$ can be approximated as constant as stated in [96]):

$$P_{cpu,static} = r_1 f e^{r_2 f} + r_3 f \tag{4.16}$$

A further constraint can be introduced on the interpolation procedure, in order to assure that the static power does not dominate the switching power approximated, in Section 4.2, as $\mathrm{O}\big(f^3\big)$, i.e.

$$a_1 f^3 > a_2 f e^{a_3 f} + a_4 f \tag{4.17}$$

where all a_i $(1 \leq i \leq 4)$ are constant fitting parameters.

4.7. LACP Features and Summary

This chapter analysed the three application execution relevant metrics, namely: *(i)* TtS, *(ii)* APC, and *(iii)* EtS. The following bullet points summarize the main contributions that have been made to the state of the art:

- the derivation of three analytical models for estimating an application's TtS, APC, and EtS as functions of the number of compute nodes and the maximum CPU frequency at which all the cores of all compute nodes operate;

- the development of the Lightweight Adaptive Consumption Predictor (LACP) model implementing the extended AEPCP process and allowing for application TtS, APC, and EtS ahead of time estimation with respect to a given (compute resource number, maximum CPU frequency) configuration for a given parallel application; and
- the discussion on static power derivation for systems not supporting the DVFS feature by default.

Being an extension of the previously developed AEPCP model, the LACP model supports all the features outlined in Section 3.7. The developed LACP model was validated with different application benchmarks and thus serves as an ideal building block for a real-world implementation of energy-aware RMSS.

The information on the possible execution time for a given configuration can yield tighter bounds for the wall-clock time that users specify in the application submission scripts. This in turn could improve the efficiency of the backfilling process of the RMSS, thus minimizing the waiting time (i.e. the time until the application starts its actual execution) for the users.

The LACP model can assist users/customers in controlling their power/energy budgets and can help data centers in implementing energy-driven charging policies as an alternative to currently existing CPU-hour based charging policies.

5. The First Steps Towards Tackling the Execution Time, Energy and Power Consumption Tradeoffs☆

5.1. Preface

As mentioned before, none of the currently available RMSS completely supports energy/power capping, since none of them has an a priori knowledge on power or energy consumption of the applications to be scheduled with respect to different compute resource configurations. By having validated the LACP model in Chapter 4, this chapter will further explore the possible application range of the model, will particularly examine the model usage for system, user, and/or data center energy and power budgeting in real world situations, which will in its turn allow for a complete implementation of energy and power capping.

5.2. Tackling the Energy Capping

Assume that a user is left with 9 kWh monthly energy budget after executing EPOCH using 74, 112, and 210 compute nodes[1] - all at maximum operating CPU frequency of 2.3 GHz. Table 5.1 recaps the PowerDAM tracked data for these three EPOCH runs.

Number of Compute Nodes	**Maximum CPU Frequency (GHz)**	**Measured EtS Value (kWh)**	**Measured APC Value (W)**	**Measured TtS Value (min)**
74	2.3	9.2	12115	45.58
112	2.3	9.21	18281.16	30.23
210	2.3	9.05	34014.268	15.96

Table 5.1.: PowerDAM tracked EPOCH history data

Now, user wants to run EPOCH with 220 compute nodes - in order to do so the client needs to answer the following question: *is it possible to run EPOCH with 220 compute nodes within my 9 kWh energy budget?* Currently none of the available

[1]As usual, these node counts were chosen on a random basis.

☆This chapter is partly based on the following previous work of the author: *Hayk Shoukourian et al. Predicting Energy Consumption Relevant Indicators of Strong Scaling HPC Applications for Different Compute Resource Configurations. Proceedings of the 23rd High Performance Computing Symposium, Society for Modeling and Simulation International (SCS),* 2015.

RMSS's can answer this question. Figure 5.1 shows that the usage of LACP can lead to an answer.

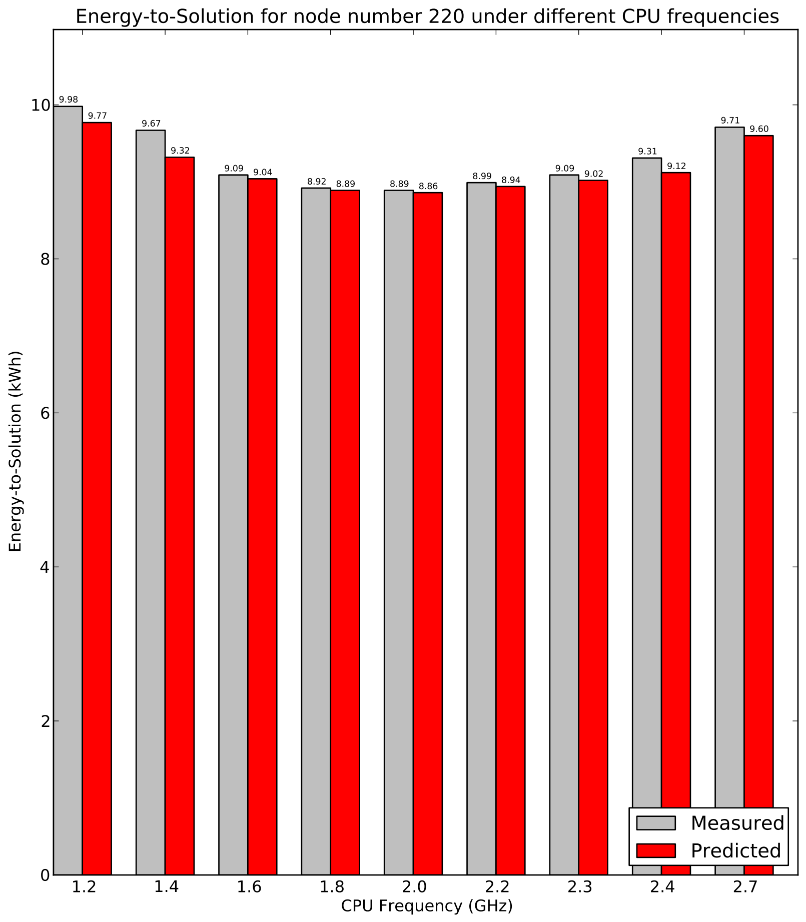


Figure 5.1.: LACP EtS prediction results for EPOCH when executed with 220 compute nodes

Figure 5.1 illustrates the LACP EtS prediction results for EPOCH 220 compute node count execution, when using the available, PowerDAM tracked, EPOCH history data shown in Table 5.1. As can be seen, the execution of EPOCH with 220 compute nodes within the 9 kWh energy budget is possible, if the user restricts the maximum operating CPU frequency of the compute node cores to the $[1.8; 2.2]$ GHz range, since any execution that uses $(220, f)$ configuration, where $f \in [1.8; 2.2]$, according to the LACP model (and also to the measured data - first gray bar, Figure 5.1), will not violate the currently available 9 kWh user energy budget.

5.3. Tackling the Power Capping

Assume a data center has a 32 kW power consumption limit, that must not be violated at any given point in time. When continuing with the EPOCH example presented in Section 5.2, the question that must be answered in this context would be: *is it possible to run EPOCH with* 220 *compute nodes without violating the predefined* 32 *kW power cap?* Currently none of the available RMSS's can answer this question either. As can be seen from Figure 5.2, by using the LACP model it is possible to find an answer.

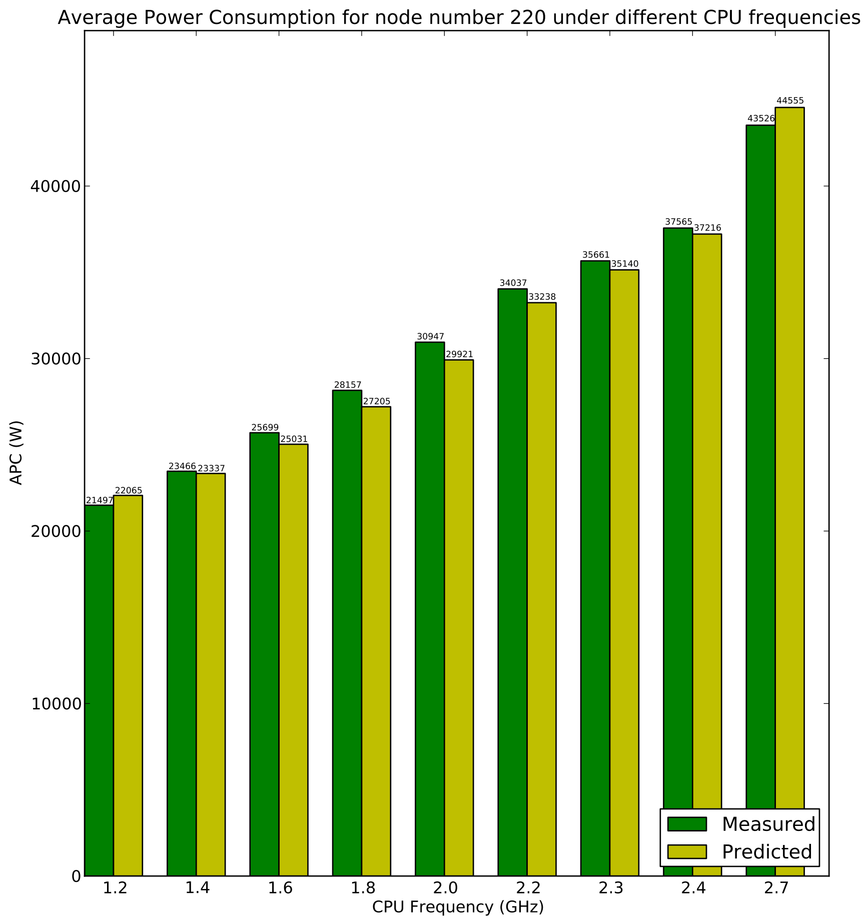


Figure 5.2.: LACP APC prediction results for EPOCH when executed with 220 compute nodes

Figure 5.2 illustrates the LACP APC prediction results for EPOCH 220 compute node count execution, when using the available, PowerDAM tracked, EPOCH history data presented in Table 5.1. As can be seen, the execution of EPOCH with 220 compute nodes is possible as long as the maximum CPU frequency is restricted to $[1.2; 2.0]$ GHz operating range, since any execution that uses $(220, f)$ configuration, where $f \in [1.2; 2.0]$, according to the LACP model (and also to the measured data - first green bar, Figure 5.2), will not violate the predefined 32 kW power consumption constraint. When also considering the user 9 kWh energy budget restriction (Section 5.2), the allowed maximum CPU frequency range for the compute node cores becomes: $[1.8; 2.2] \cap [1.2; 2.0] = [1.8; 2.0]$ GHz.

The AEPCP model presented in Chapter 3 could also answer the posed questions in Section 5.2 and in Section 5.3, but only for the fixed CPU frequency - in this case for 2.3 GHz, since all the user executions of EPOCH where conducted at a maximum frequency of 2.3 GHz (Table 5.1). Note that, for example in the case of the discussed 32 kW power cap, the usage of AEPCP would have disallowed the execution of EPOCH with 220 compute nodes, since the predicted $\mathrm{APC}(220, 2.3) =$ 33 kW and thus violates the predefined 32 kW power constraint. This once again shows the importance of application EtS/APC/TtS prediction also with respect to operating CPU frequency, which LACP model supports.

5.4. Tackling Execution Time-Energy Consumption Tradeoff

As observed in Section 5.2 and in Section 5.3, several resource configurations can satisfy given energy or power caps. While in the most cases the users would prefer to execute their applications at the maximum possible CPU frequency[2], as seen in Section 5.2, with the notion of energy budgeting this might not be always the case. In order to do the final decision and to understand the possible tradeoffs between different resource configuration executions the knowledge on application execution time with respect to various potentially unknown (i.e. not executed before) configuration is required.

When revisiting the EPOCH EtS prediction results presented in Section 5.2, the execution with either of $(220; 1.8)$ or $(220; 2.0)$ configurations would not violate the user available 9 kWh energy budget. *So which configuration should be chosen?* In order to answer this question, the LACP TtS prediction results for EPOCH with 220 compute nodes are further considered (Figure 5.3).

As can be seen, the EPOCH execution with 220 compute nodes at CPU frequency of 1.8 GHz will result in TtS of 19.29 minutes (directly predicted TtS white bar, Figure 5.3), whereas at 2.0 GHz - in 17.46. Combining this information with the data illustrated in Figure 5.1, the user can decide to trade almost 2 minute execution time for 0.03 kWh (= 1800 Wmin) energy cost.

5.5. Tackling Execution Time-Energy Consumption-Power Consumption Tradeoff

This section briefly discusses the usage of LACP model in determination of possible tradeoffs between the user available energy budget, application execution time, and system power cap.

Assume, that there is a 45 kW power consumption constraint on the target HPC system (for example, due to some data center infrastructure maintenances). Assume further, that there is a user who has three times executed EPOCH with different resource configurations (summarized in Table 5.1), is left with 9 kW, and still

[2]Since this would lead to application execution time reduction.

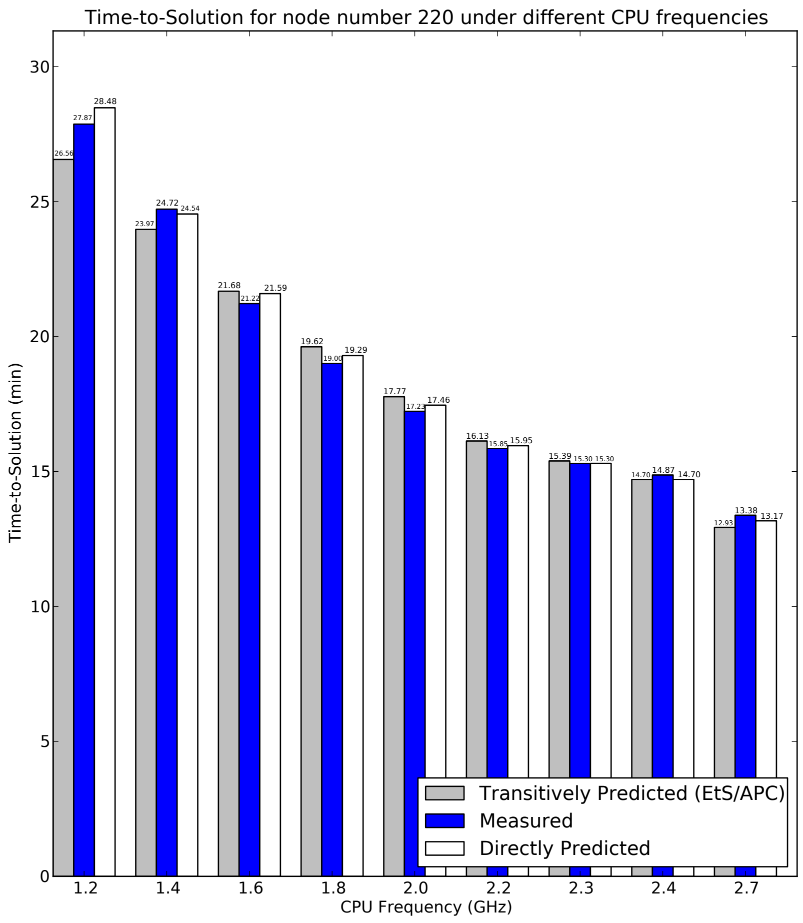
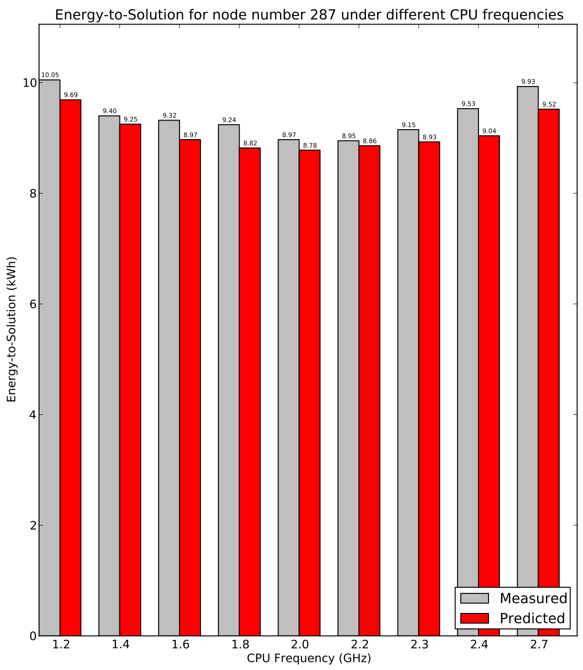


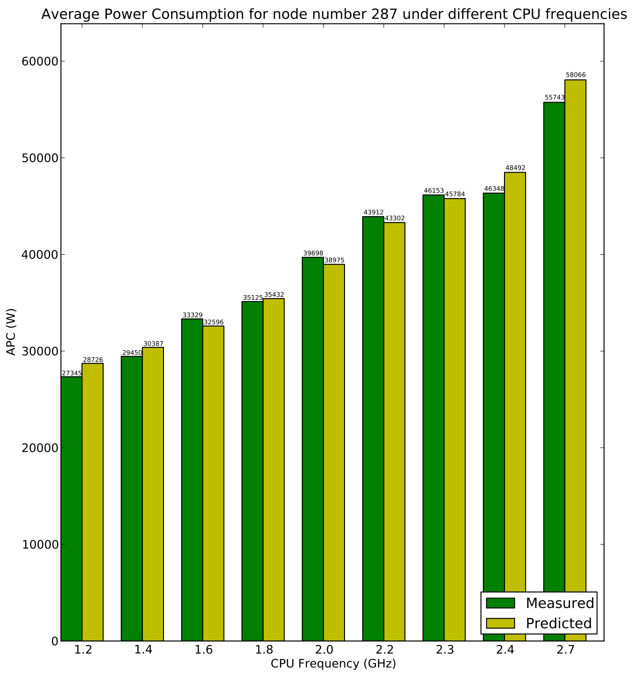
Figure 5.3.: LACP TtS prediction results for EPOCH when executed with 220 compute nodes

wants to execute EPOCH with 287 compute nodes and have the minimum possible execution time. The question that needs to be answered here would be: *what is the TtS-wise optimal operating CPU frequency for EPOCH using 287 compute nodes, execution at which will not violate the 9 kWh user energy budget and 45 kW system power cap?* Again, none of the currently available RMSS's can support the user (or data center operator) in making this decision.

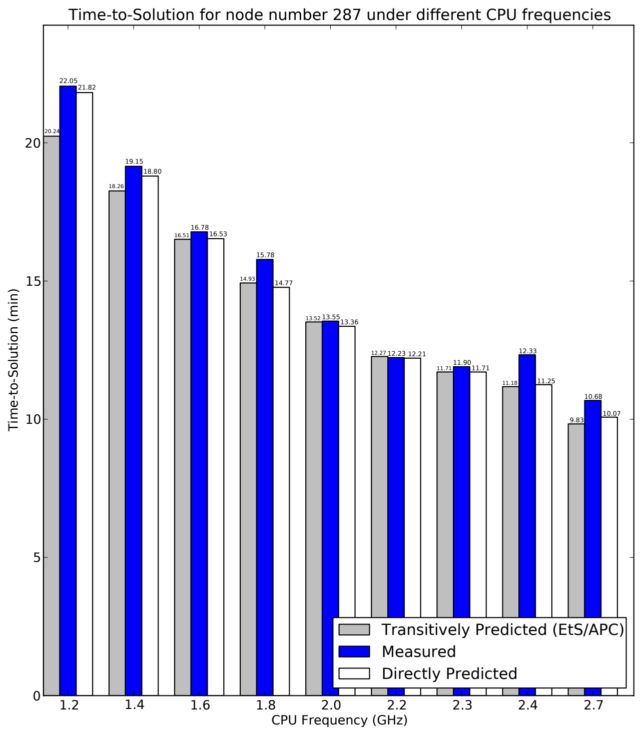
Figure 5.4a, Figure 5.4b, and Figure 5.4c present the LACP EtS, APC, and TtS prediction results for EPOCH when executed with 287 compute nodes. As can be seen, because of the 45 kW system power cap, the allowed CPU frequency of all cores of all 287 compute nodes must be restricted to $[1.2; 2.2]$ GHz operating frequency range. On the other hand, due to the user 9 kWh energy budget, this range is further shrunk to $[1.6; 2.2]$ GHz range (Figure 5.4a). Since the $(287, 2.2)$ configuration will yield the minimum possible execution time of 12.21 minutes within the allowed $[1.6; 2.2]$ GHz CPU frequency range (Figure 5.4c), it can be concluded that the 2.2 GHz is the answer to the above posed question.



(a) EtS prediction results



(b) APC prediction results



(c) TtS prediction results

Figure 5.4.: LACP prediction results for EPOCH with 287 compute nodes

Note: available data points are for compute node counts: 74, 112, *and* 210

5.6. Summary

This chapter outlined the use cases of the developed LACP model in real-world scenarios. More specifically, the chapter showed how can the LACP model be applied in energy and power capping problem domains and how the model can assist in evaluation of possible tradeoffs between application execution time, power, and energy consumptions.

Having shown the model applicability, the next Chapter will further scratch the surface of the LACP model, by designing a framework for controlling the average power consumption of individual HPC systems deployed in the data center. It will present a RMSS plug-in, that per-application basis, efficiently selects energy consumption wise optimal resource configuration for user specified application execution time constraint and for data center operator defined power cap. The proposed plug-in can then be used for automatic control of data center operator and user specified power consumption and execution time constraints.

Part IV.

Towards a Unified Implementation of Software Defined Power Capping

"I have learned to seek my happiness by limiting my desires, rather than in attempting to satisfy them."
JOHN STUART MILL

6. Software Defined Power Capping For Modern HPC Data Centers[☆]

6.1. Preface

Having developed models allowing for application beforehand execution time, average power, and energy consumption estimations as well as shown their practical applicability in real-world domains, this chapter will further explore their potential for the enhancement of the existing energy-aware resource management and scheduling solutions. More specifically, the chapter will discuss: *(i)* the problem of finding the optimal resource configuration for a given application that will minimize the amount of consumed energy, under pre-defined constraints on application execution time (typically specified by users) and instantaneous average power consumption (typically specified by the data center operators); and *(ii)* the impact of the power variation, across the compute resources of homogeneous HPC systems, on the mentioned constraint scheduling problem. The chapter will present an algorithm that will efficiently solve the mentioned problem. Based on this algorithm, the chapter will present a plug-in, referred to as Configuration Adviser (CA) [103], which operates on top of a given RMSS to advise on energy-wise optimal resource configuration for a given application, execution using which, will adhere to the specified runtime and power consumption constraints.

The main goal of the CA plug-in is to enhance the current resource management and scheduling solutions for the support of power capping while meeting the requirements of users/clients. The presented CA plug-in is in complement to the existing in RMSS energy/power saving techniques and can be used in conjunction with any other energy/power savings efforts.

6.2. Revisiting the Problem Statement

Software defined power capping requires the ability of beforehand estimation of the APC of applications to be scheduled for given RCs. Moreover, if there exists a user-specified application maximum execution time constraint, the beforehand estimation of TtS of the applications-to-be-scheduled is also required. Lightweight Adaptive Consumption Predictor (LACP), described in Chapter 4, is a model allowing for these beforehand estimations of application execution characteristics.

[☆]This chapter is partly based on the following previous work of the author: *Hayk Shoukourian et al. Power Variation Aware Configuration Adviser for Scalable HPC Schedulers. Proceedings of the* 13 *International Conference on High Performance Computing & Simulation, HPCS,* 2015.

Figure 6.1 highlights the constraints and the solution space for the software defined power capping. The graphs (a), (b), and (c) illustrate the application TtS, APC, and EtS behaviors with respect to the operating CPU frequency, since in case of a **fixed** n number of application to-be-utilized compute nodes, the following approximations hold true: TtS $= \frac{k_1}{f}$ and APC $= k_2 \cdot f^3$, leading to the EtS $=$ TtS $\cdot$ APC $= k_3 \cdot f^2$ approximation, where all the k_i $(1 \leq i \leq 3)$ are constant fitting parameters (Subsection 4.2.1). These further mean, that higher CPU frequencies lead to reduced execution times Figure 6.1 (a) and higher average power consumptions Figure 6.1 (b).

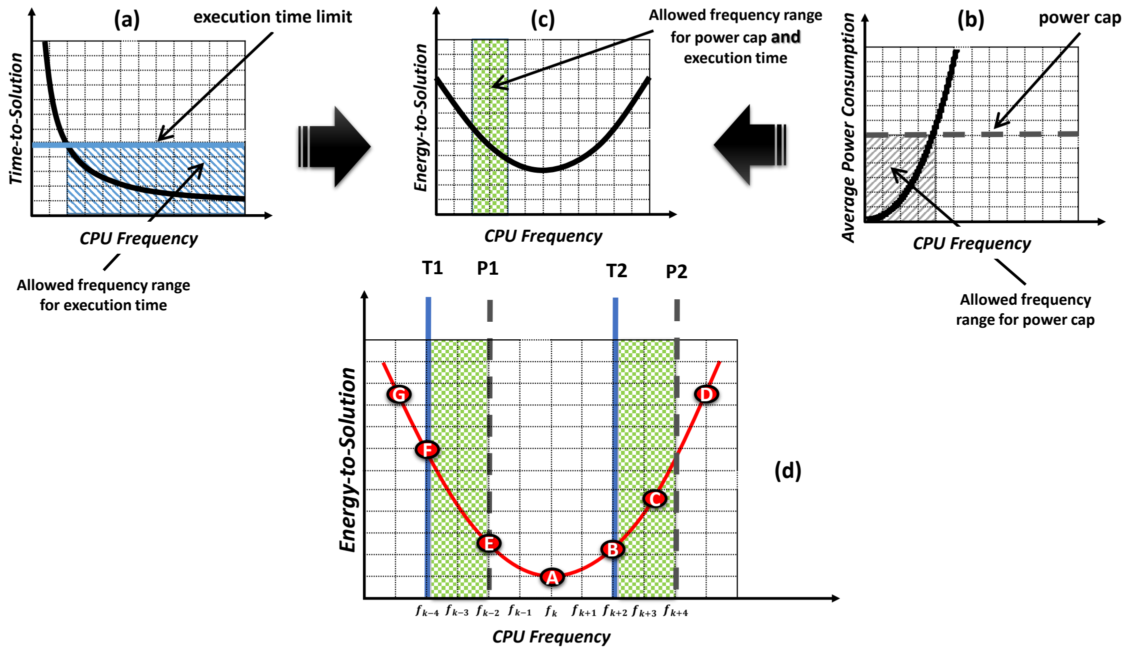

(a)
execution time limit
Time-to-Solution
CPU Frequency
Allowed frequency range for execution time
(c)
Allowed frequency range for power cap and execution time
Energy-to-Solution
CPU Frequency
(b)
power cap
Average Power Consumption
CPU Frequency
Allowed frequency range for power cap
T1
P1
T2
P2
G
F
E
A
B
C
D
(d)
Energy-to-Solution
f_{k-4} f_{k-3} f_{k-2} f_{k-1} f_k f_{k+1} f_{k+2} f_{k+3} f_{k+4}
CPU Frequency

Figure 6.1.: Overview of the problem statement

Assume there are predefined constraints on the application execution time and average power consumption. The blue solid and the gray dashed horizontal lines in graphs Figure 6.1 (a) and Figure 6.1 (b) correspondingly illustrate these constraints. The problem of energy-wise optimal operating CPU frequency estimation for the application to-be-utilized compute resources, within these specified TtS and APC constraints, is further examined. This examination does not consider the possibility of compute node count variation, since in the real-world situations the specification of the number of application-to-be-utilized compute nodes is a requirement and is expected to be set by the user in application submission script [5].

The hashed sections beneath the horizontal lines of Figure 6.1 (a) and Figure 6.1 (b) illustrate the allowed CPU frequency range, execution at which will meet the specified TtS and APC constraints. The green hashed section in Figure 6.1 (c) illustrates the intersection of the allowed CPU frequency ranges from Figure 6.1 (a) and Figure 6.1 (b). Now, a frequency value must be chosen from this hashed green segment of Figure 6.1 (c), such that it would result in the minimal EtS consumption value.

Figure 6.1 (d) illustrates two example intersection scenarios of TtS and APC allowed CPU frequency ranges. The first one is the frequency ranged within the $[T1; P1]$ hashed segment. For this case, the two example f_{k-4} and f_{k-2} frequencies will meet the pre-defined execution constraints but the frequency point f_{k-2} will be the correct answer to the above stated problem, since the execution at the frequency f_{k-2} will lead to the minimal E energy costs. In the case where the intersection of TtS and APC allowed frequency ranges results in $[T2; P2]$ hashed segment, Figure 6.1 (d), the energy-wise optimal operating frequency will be f_{k+2}. Note that in the both cases, the solution points are the ones that are within the intersection and closest to the global energy-optimal frequency point f_k having EtS of A - which, in both cases, is not within the intersection of the specified TtS and APC frequency ranges and thus cannot be considered as a solution. The non-schedulability of the application could arise in the case where the intersection of allowed frequency ranges of execution time and average power consumption results in an empty set.

The possible solution space for the considered problem can be formally defined as:

$$\begin{cases} \textit{(a)} & CPU_{lowest\ frequency} \leq f \leq CPU_{highest\ frequency} \\ \textit{(b)} & APC(n, f) \leq P_c \\ \textit{(c)} & TtS(n, f) \leq T_d \end{cases} \tag{6.1}$$

where the goal is to find a (n, f) RC satisfying the above mentioned conditions and having the minimal possible EtS(n, f) consumption value within these power consumption and execution time constraints.

The conditions defined in (6.1) are as follows: ***(a)*** the boundaries for the processor operating frequency f are set. It is further assumed that all the processors of the compute nodes support only a finite set of maximum frequencies. Note that this assumption is realistic since most of the current DVFS mechanisms only support a finite set of states [104]. The condition ***(b)*** indicates the system power consumption constraint (power cap) where P_c is the currently available to the application power amount, defined as a difference of power cap and current system average power consumption; and the condition ***(c)*** shows the execution time constraint (T_d) specified by the user in the application submission script.

6.3. Power Distribution Variation with Operating Frequencies

Section 3.5 showed on two different *homogeneous* HPC systems that node power consumption, under the same workload, is *inhomogeneous* for a fixed maximum operating CPU frequency, set across all the cores of the compute nodes. This section further investigates this variation, in particular the distribution behavior at different maximum operating CPU frequencies.

Figure 6.2 shows the distribution of the average power draw (i.e. APC) of the individual compute nodes from SuperMUC island 5 when running the single-node benchmark FIRESTARTER on each of them[1]. All the cores of the compute nodes had a maximum CPU frequency of 1.4 GHz. FIRESTARTER is a tool containing specifically optimized routines for $x86\_64$ processors. It generates near peak power consumption for a compute node and has a steady workload character without fluctuations, in contrast to MPRIME, that consists of different computational routines and could potentially create a non-constant load [105].

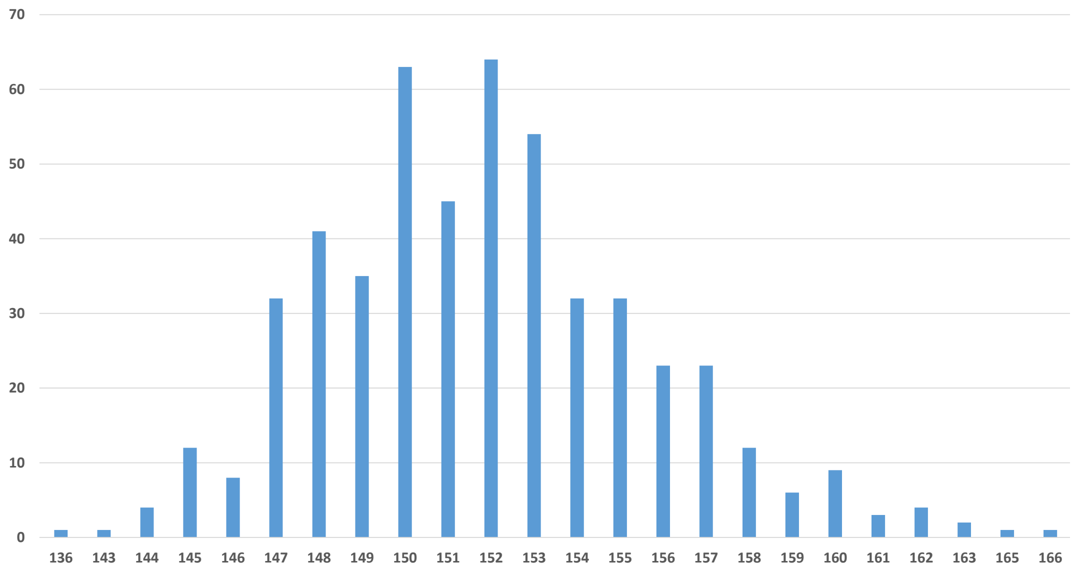


Figure 6.2.: Average power draw of compute nodes for the SuperMUC Island 5 when executed FIRESTARTER at 1.4 GHz maximum CPU frequency

The x-axis in Figure 6.2 shows the observed average power consumption and the y-axis shows the compute node count. Although, the measurements obtained from SuperMUC's paddle cards, as presented in Section 3.6, show a high accuracy, Figure 6.2 illustrates the averaged data obtained from 5 different measurements[2]. As can be seen, the power draw for the same application varies around 19.8% among the compute nodes[3].

Figure 6.3 and Figure 6.4 show the average power distribution of the same thin island 5 compute nodes but when all the cores of the compute nodes are operating at maximum CPU frequencies of 2.3 GHz and 2.7 GHz correspondingly.

As can be seen, the distribution shown in Figure 6.2 changes with respect to the operating maximum CPU frequency of compute node cores. This further means that the *"power-efficiency"* pattern of a given compute node can change with CPU frequency. Figure 6.5 illustrates this observation for some of the compute nodes

[1]These benchmarks were carried out during the winter time, when the inlet temperature of the compute nodes was set to 33.1 °C.

[2]The same is true for the subsequently presented Figure 6.3 and Figure 6.4.

[3]Calculated as $[|\ value_1 - value_2 \ | / ((value_1 + value_2)/2)] * 100$.

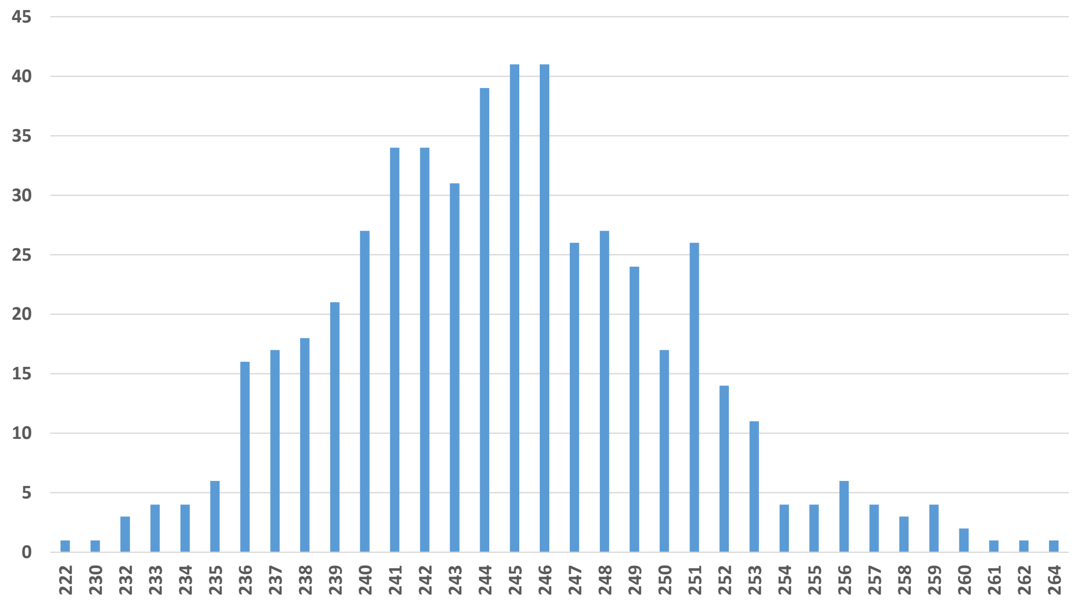


Figure 6.3.: Average power draw of compute nodes for the SuperMUC Island 5 when executed FIRESTARTER at 2.3 GHz maximum CPU frequency

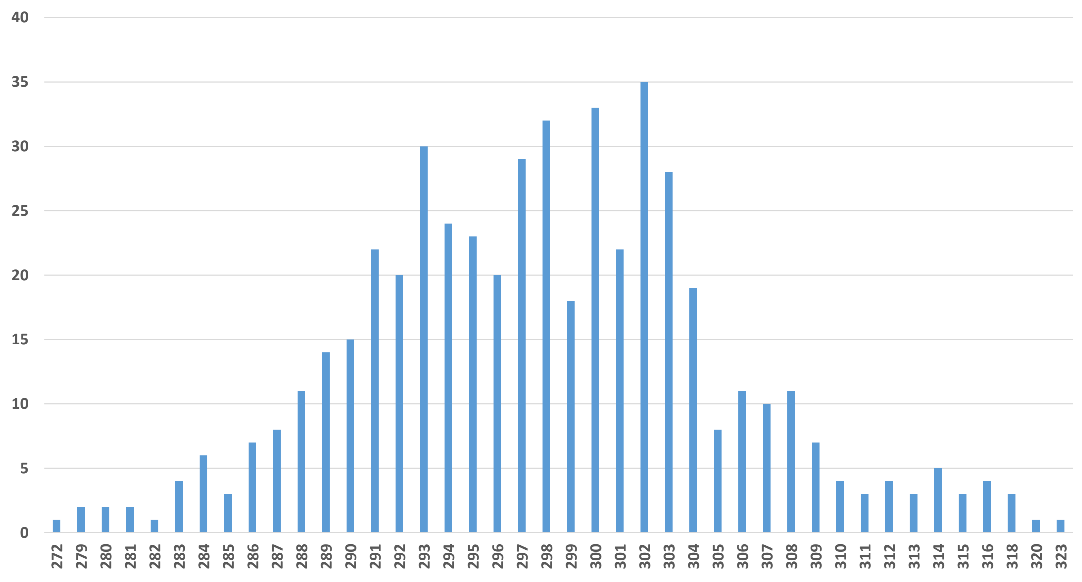


Figure 6.4.: Average power draw of compute nodes for the SuperMUC Island 5 when executed FIRESTARTER at 2.7 GHz maximum CPU frequency

from the same thin island of SuperMUC, again, using the FIRESTARTER workload, at different, processor supported, maximum operating CPU frequencies.

The illustrated in Figure 6.5 non-uniform rise of the APC values of the compute nodes leads to the following observation: *if one node n_1 is considered more power efficient then another compute node n_2 at a given CPU frequency x, then at another CPU*

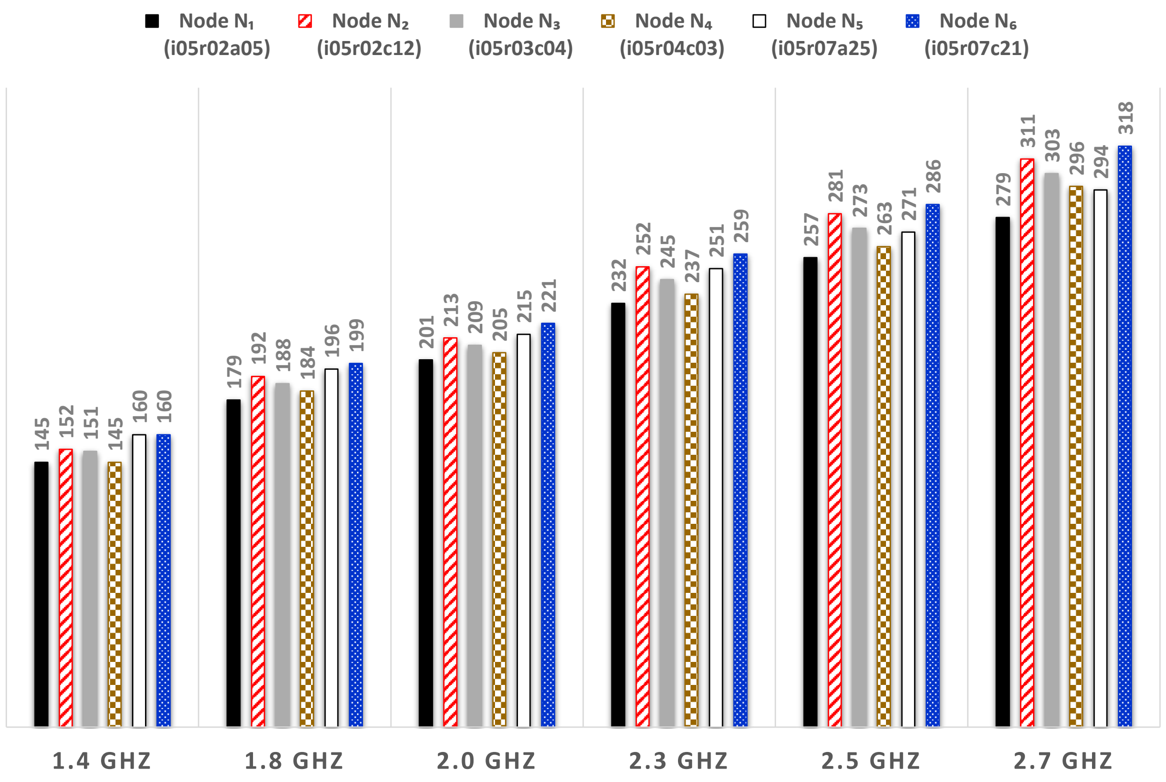


Figure 6.5.: Average power draw of some SuperMUC Island 5 compute nodes when executed FIRESTARTER with different maximum CPU frequencies

frequency y this could be not true. For example, the compute node N_2 (red hatched bar) at maximum frequency 1.4 GHz, consumes 152 W which is 5.13% less than the 160 W power draw of the N_5 compute node (white bar). But this variation changes for 2.7 GHz. The APC of N_2 reaches 311 W which is 5.62% higher than the 294 W APC of N_5. On the other hand, the compute node pair N_1 (black bar) and N_4 (brown "chess-like" hatched bar) show identical power draw at 1.4 GHz, whereas the difference in APC reaches 5.9% at 2.7 GHz (279 W vs 296 W). This behavior can be explained by the fact that on the lower CPU frequencies the power consumption is dominated by the memory, hard drives, PCI slots, and by the other components present on the compute node; whereas on the higher frequencies the power consumption of the compute node is mainly dominated by the CPU power consumption. Note that these variations (value wise) might become even bigger for air cooled systems, since the latter ones are less efficient than water cooled systems [106], implying higher CPU temperatures, which in their turn bring to higher power consumption rates.

The observed power variation means that the consumed energy and average power of an application for a given (n, f) configuration will be different when executed on different sets of n system compute nodes. Thus EtS and APC metrics also depend on a certain set S of n compute nodes, i.e. $EtS(n, f, S)$, $APC(n, f, S)$. Table 6.1 shows that this is not the case for TtS metric, more specifically it presents the measured APC and TtS values of the High Performance Linpack benchmark [6] when executed on four different compute nodes[4]. The HPL benchmark is used to create the TOP500 [1] list of the fastest supercomputers in the world. It solves a dense system of linear equations and shows the measure of achieved performance on a given system. The system of linear equations is represented as a matrix divided into small pieces, referred to as tiles, which are distributed across the processors of the compute nodes of the system.

Compute Node	**Compute Node Selection Policy**	**Measured APC Value (W)**	**Measured TtS Value (min)**	**Node Performance (GigaFLOPS)**
i05r05a19-ib	node with the lowest APC value under FIRESTARTER	199.35	30.0	250.693
i05r03c28-ib	node with the highest APC value under FIRESTARTER	239.76	29.6667	253.009
i05r01a13-ib	random	216.73	29.85	250.9
i05r01a11-ib	random	221	29.88	250.779

Table 6.1.: Negligible difference in TtS among compute nodes when all executed at maximum operating CPU frequency of 2.3 GHz

As can be seen, APC varies between the compute nodes but the differences in TtS and GigaFLOPS[5] values are negligible. On 512 compute node scale, a maximum of 3% variation in TtS of single-node HPL runs was observed.

6.4. Examining the Potential of Energy Savings With Node Power Variation

This section looks into some simple energy-saving techniques that rely on compute node pre-selection.

Table 6.2 shows the execution results of HYDRO when executed on differently selected sets of 256 compute nodes. Two selection policies are used in this table: $WorstExistingNodes(n, f)$ defined the list of system n compute nodes which have the **highest** APC value at given f frequency; and $BestExistingNodes(n, f)$ defined the list of system n compute nodes which have the **lowest** APC value at given f frequency, in this case for 2.3 GHz.

Compute Node Selection Policy	**Measured EtS Value (kWh)**	**Measured APC Value (W)**	**Measured TtS Value (min)**
WorstExistingNodes(256,2.3)	18.28	41078.65	27.7
BestExistingNodes(256,2.3)	17.43	39168.54	27.69

Table 6.2.: Difference of EtS and APC values for HYDRO when executed on different sets of compute nodes for 2.3 GHz

[4]Chosen from the same island 5 of SuperMUC.

[5]1 GigaFLOPS = 10^9 FLOPS.

As can be seen, the best nodes have a power (thus also energy) advantage of 4.76%, while there is no significant difference in the execution times. Table 6.3 shows the measurement results for EPOCH, when the same two selection policies are considered.

Compute Node Selection Policy	Measured EtS Value (kWh)	Measured APC Value (W)	Measured TtS Value (min)
WorstExistingNodes(256,2.3)	9.2	42086	13.116
BestExistingNodes(256,2.3)	8.83	40341.12	13.133

Table 6.3.: Difference of EtS and APC values for EPOCH when executed on different sets of compute nodes

As can be observed, also in the case of EPOCH, there could be a gain of 4.1% in energy savings. Although, it might be argued that this gain is negligible, especially for the case when aiming for the maximum application throughput and complete utilization of all the available compute resources in the system at any point in time (for example, when combining the HYDRO 850 Wh energy savings with 370 Wh EPOCH energy savings, resulting in 480 Wh total savings) it is notable that:

- due to the variation fact of compute node power efficiency for a given CPU frequency and diversity in compute resource requirements of different applications, as well as the difference in application energy and power consumption profiles, the possibility of scheduling more applications within given energy, power, and/or execution time constraints at a given point in time could increase with compute node pre-selection methods;
- the benefit in energy savings in the case of compute node pre-selection becomes even more vivid when considering the case of application execution on a smaller number of compute nodes. Table 6.4 shows this behavior for High Performance Conjugate Gradient (HPCG) benchmark.

Compute Node Selection Policy	Measured EtS Value (kWh)	Measured APC Value (W)	Measured TtS Value (min)
WorstExistingNodes(10,2.3)	0.658	2134	18.5
BestExistingNodes(10,2.3)	0.575	1865	18.5

Table 6.4.: Difference of EtS and APC values for HPCG when executed on different sets of compute nodes

The HPCG is a synthetic benchmark [107] which generates and solves a 3D sparse linear system using a local symmetric Gauss-Seidel preconditioned conjugate gradient method. It is specifically composed of computations and data access patterns which are commonly used in real-world scientific applications for providing a better measure of application performance.

- the knowledge of power variability could be especially useful when defining the interactive and batch partitions of the system - since typically interactive partitions consist from the smaller amount of compute nodes, the ones showing relative high power consumption could potentially be moved to the interactive partition, which then could be switched off (or set to idle mode) more frequently, for example during the weekends or night time, when user activity is relatively low[6];
- the observed variation could be applicable in a cloud computing domain as well, since the environment virtualization makes the task migration and ON/OFF switching of the compute nodes more flexible - which further means that the compute nodes showing the worst "power-efficiency" could be used the least.

The study found in [108] provides a further detailed discussion on three techniques that do take into account the shown node power variation (for a fixed frequency) for achieving some energy savings.

6.5. Determining the Energy-Optimal Application Resource Configuration

This section describes the suggested software defined power capping algorithm, proves its optimal termination, and derives the time complexity.

6.5.1. The Algorithm

The developed algorithm takes as an input:

- (i) the application identifier;
- (ii) the user specified n number of compute nodes;
- (iii) application execution time deadline T_d; and
- (iv) the currently allowed system average power consumption P_c.

If the specified constraints can be met, then the algorithm returns a pair consisting of:

- the set S of n compute nodes and
- the maximum operating CPU frequency f (at which all the cores of n compute nodes should operate)

[6]Under the assumption of negligible timezone variations among the system users.

that minimizes the EtS for the application scheduling.

The algorithm requires a model that can estimate the EtS, APC, and TtS values of a given application for a given n number of compute nodes, f CPU frequency, and if specified, the concrete set S of n compute nodes. If the node set S is not specified, then an average value (compute node set independent) should be reported. LACP (described in Chapter 4) is an example of such a model. It is further assumed, that the EtS, APC, and TtS values of a given application for a given resource configuration can be estimated in $\mathrm{O}(1)$ constant time.

In short, the algorithm first estimates the energy-optimal maximum operating frequency point and then, by stepwise frequency increases/decreases (and by correspondingly adjusting the corresponding compute node set), adapts this found energy-optimal frequency point until the specified TtS and APC constraints are met.

6.5.2. Detailed Description

Let $\{f_1, f_2, ..., f_q\}$ set to be a sorted in ascending order list of processor supported operating frequencies, i.e. $f_1 < f_2 < ... < f_q$. Further, for each f_i frequency value ($1 \leq i \leq q$), there exists a S_i set which contains all the system compute nodes and is sorted from *"the best to the worst"* in terms of average power consumption for that particular frequency f_i, as estimated in Section 6.3. The following auxiliary functions are further used:

- get**BestExisting**$Nodes(f_i, n)$ - for a given frequency f_i returns the **first** n compute nodes from the S_i set. It is assumed that this function has $\mathrm{O}(1)$ constant processing time;
- get**WorstExisting**$Nodes(f_i, n)$ - for a given frequency f_i returns the **last** n compute nodes from the S_i set. It is assumed that this function has $\mathrm{O}(1)$ constant processing time;
- get**BestAvailable**$Nodes(f_i, n)$ - for a given frequency f_i returns the **first** n currently **available** compute nodes from the S_i set. If the system has m compute nodes, then this operation will have the worst case processing time of $\mathrm{O}(m)$.
- is**Increase**$Possible(f_i)$ - for a given frequency f_i checks if a possible stepwise **increase** is possible, i.e. returns *True* if $f_i < f_q$, and *False* otherwise. It is assumed that this function has $\mathrm{O}(1)$ constant processing time;
- is**Decrease**$Possible(f_i)$ - for a given frequency f_i checks if a possible stepwise **decrease** is possible, i.e. returns *True* if $f_i > f_1$, and *False* otherwise. It is assumed that this function has $\mathrm{O}(1)$ constant processing time;
- **increase**$Frequency(f_i)$ - for a given frequency f_i returns the stepwise **next** frequency, i.e. f_{i+1}, where $1 \leq i < q$. It is assumed that this function has $\mathrm{O}(1)$ constant processing time; and

- **decrease**$Frequency(f_i)$ - for a given frequency f_i returns the stepwise **preceding** frequency, i.e. f_{i-1}, where $1 < i \leq q$. It is assumed that this function has $O(1)$ constant processing time.

The detailed workflow of the algorithm is as follows. First, the energy-optimal operating CPU frequency point is determined by iterating through all possible q frequency values in the worst case. Once the f_{minEtS} energy-optimal frequency point is found, the set S gets assigned with the best available compute nodes for that frequency value, i.e. $S \leftarrow get$**BestAvailable**$Nodes(f_{minEtS}, n)$ (line 2). Thus in total, the worst case processing time complexity of line 2 is: $O(q + m)$. The algorithm then checks if the execution time constraint is specified (line 4). If the application execution time T_d constraint is specified, then the algorithm checks if the found f_{minEtS} frequency (in line 2) satisfies that T_d constraint. If the constraint is not satisfied, the algorithm starts adapting the found f_{minEtS} frequency to the T_d constraint by stepwise increasing it[7] (the while loop of lines 4-13). If these stepwise increases do not lead to the satisfiability of the execution time constraint, the algorithm terminates with the application being not schedulable (line 11).

Once a f_{minEtS} frequency satisfying the execution time constraint is found (line 13) and the compute node set S, if necessary, correspondingly modified (lines $14 - 17$), the algorithm checks whether the currently estimated f_{minEtS} frequency satisfies the current power cap value with the currently available n best compute nodes from set S (line 19). If this is not the case, then the algorithm:

- ***(c1)*** checks if the scheduling will be possible with the usage of the best existing nodes for the current frequency value of f_{minEtS}. If this is the case, then it saves (1) the frequency f_{minEtS}, (2) the get**BestExisting**$Nodes(f_{minEtS}, n)$ set, and (3) sets the $schedulingWithWaitingPossible$ boolean flag; and

- ***(c2)*** stepwise decreases the f_{minEtS} until the specified power constraint is satisfied (lines $19 - 42$).

During each frequency decrease, the compute node set S and the integer $frequencyDecreaseCounter$ get correspondingly updated (lines 28 and 29). The later value is then used to increase the existing user budget (e.g. CPU hours, energy budget, etc.) in order to compensate for execution time penalties arisen due to the specified data center power cap.

If during the stepwise decreases no frequency point satisfying the current power cap value is found, the algorithm, depending on the $schedulingWithWaitingPossible$ boolean flag value, terminates with **either**:

- ***(a1)*** a (S_{tmp}, f_{tmp}) pair of n currently **not available** compute nodes[8] and frequency, execution using which will satisfy the specified constraints (line 35) but will require waiting; **or**

[7]Since the increase in the CPU frequency will decrease the execution time (Subsection 4.2.1).

[8]Meaning that application must wait until the required compute resources will be available.

ALGORITHM 1. Algorithm for estimating energy-optimal operating CPU frequency for a given job within the specified TtS and APC constraints

Data: A job j with a given number n of compute nodes, TtS deadline T_d, and current APC constraint P_c

Result: Set of n compute nodes S and a CPU frequency f such that:

1. $APC(n, f, S) \leq P_c$;
2. $TtS(n, f) \leq T_d$; and
3. $EtS(n, f, S)$ is minimized, i.e. $\forall f'$ frequency and $\forall S'$ available compute node set, satisfying (1) and (2), the following holds true: $EtS(n, f, S) \leq EtS(n, f', S')$

```
$S \leftarrow \emptyset$;
find $f_{minEtS}$ frequency and currently **available** node set $S$ such that
$EtS(n, f_{minEtS}, S) \leq EtS(n, f', S')$ $\forall f'$ frequency and $\forall S'$ currently **available** node set;
$frequencyDecreaseCounter \leftarrow 0$; $modified_{f_{minEtS}} \leftarrow False$;
while $T_d \neq NULL$ and $TtS(n, f_{minEtS}) > T_d$ do
    if $isIncreasePossible(f_{minEtS}) = True$ then
        $f_{minEtS} \leftarrow increaseFrequency(f_{minEtS})$;
        $modified_{f_{minEtS}} \leftarrow True$;
    end
    else
        /* not schedulable within the given TtS constraint */
        return $(NULL, NULL)$;
    end
end
if $modified_{f_{minEtS}} = True$ then
    $S \leftarrow getBest\textbf{Available}Nodes(f_{minEtS}, n)$;
    $modified_{f_{minEtS}} \leftarrow False$;
end
$schedulingWithWaitingPossible \leftarrow False$;
while $P_c \neq NULL$ and $APC(n, f_{minEtS}, S) > P_c$ do
    if $schedulingWithWaitingPossible = False$ and
    $APC(n, f_{minEtS}, getBest\textbf{Existing}Nodes(f_{minEtS}, n)) \leq P_c$ then
        $schedulingWithWaitingPossible \leftarrow True$;
        $f_{tmp} \leftarrow f_{minEtS}$;
        $S_{tmp} \leftarrow getBest\textbf{Existing}Nodes(f_{minEtS}, n)$;
        $frequencyDecreaseCounter_{tmp} \leftarrow frequencyDecreaseCounter$
    end
    if $isDecreasePossible(f_{minEtS}) = True$ and ($T_d \neq NULL$ and
    $TtS(n, decreaseFrequency(f_{minEtS})) \leq T_d$) then
        $f_{minEtS} \leftarrow decreaseFrequency(f_{minEtS})$;
        $S \leftarrow getBest\textbf{Available}Nodes(f_{minEtS}, n)$;
        $frequencyDecreaseCounter \leftarrow frequencyDecreaseCounter + 1$;
    end
    else
        if $schedulingWithWaitingPossible = True$ then
            /* scheduling possible - requires waiting */
            IncreaseUserBudget($frequencyDecreaseCounter_{tmp}$);
            return $(S_{tmp}, f_{tmp})$;
        end
        else
            /* not schedulable within the given TtS and APC constraints */
            return $(NULL, NULL)$;
        end
    end
end
IncreaseUserBudget($frequencyDecreaseCounter$);
return $(S, f_{minEtS})$;
```

- ***(a2)*** application being not schedulable result (line 39).

But if a frequency f_{minEtS} satisfying power cap is found, then the algorithm increases the user budget according to the $frequencyDecreaseCounter$ counter[9] (line 43) and returns the required (S, f_{minEtS}) pair.

The optimal termination of the algorithm is implied from the fact that the algorithm stepwise diverges the energy-optimal point until the specified execution time and average power consumption constraints are satisfied. When summarizing, it is easy to see, that the worst case processing time $T(q, m)$ of the presented algorithm (where q is the number of distinct frequencies that the processor could scale to and m is the total number of system compute nodes) is of the following order: $\mathrm{O}\big(1\big)_{line\ 1}+\mathrm{O}\big(q+m\big)_{line\ 2}+\mathrm{O}\big(1\big)_{line\ 3}+\mathrm{O}\big(q\big)_{lines\ 4-13}+\mathrm{O}\big(m\big)_{lines\ 14-17}+\mathrm{O}\big(1\big)_{line\ 18}+\mathrm{O}\big(qm\big)_{lines\ 19-42}+\mathrm{O}\big(1\big)_{line\ 43}+\mathrm{O}\big(1\big)_{line\ 44}=\mathrm{O}\big(qm\big)$.

6.5.3. Example Execution

Assume that the intersection of the allowed CPU frequency ranges for the specified execution time and current power cap constraints results in $[T1; P1]$ segment as illustrated in Figure 6.1 (d). This further means that the algorithm has to find an operating CPU frequency f that is in this segment (i.e. $T1 \leq f \leq P1$) and results in the minimal possible energy consumption value. The red solid graph illustrates the example application EtS behavior. As was mentioned, the EtS, as well as APC and TtS values of a given application for a given RC can be estimated beforehand by the LACP model.

The following describes the execution of the algorithm for this case. First it estimates the global energy-optimal CPU frequency point, i.e. $f_{minEtS} \leftarrow f_k$ (line 2). Then the algorithm checks if the execution time constraint can be satisfied with the current value of f_{minEtS}. Since the $f_{minEtS} > T1$, then the execution time constraint is satisfied (lines $4-13$). Contrary to this, the average power consumption constraint is not met, since $f_{minEtS} = f_k > P1$. For that reason, the algorithm starts to stepwise decrease the frequency f_{minEtS} to f_{k-1}, until it reaches a frequency point f_{k-2} which is less than or equal to P_1 (lines $19-43$). Once the $f_{minEtS} \leftarrow f_{k-2}$ and the set S is correspondingly adjusted to the best available compute nodes for that f_{k-2} frequency, the algorithm correspondingly increases the user budget and returns the required (S, f_{k-2}) pair which will result in EtS of E kWh, as seen in Figure 6.1 (d).

If the intersection of the allowed CPU frequency ranges for the specified constraints results in $[T2; P2]$ segment, then the algorithm starts again from the global energy-optimal frequency point f_k, but since $f_k < T2$, it stepwise increases this f_k value until the $f_{k+2} \geq T2$ is reached. Once the $f_{minEtS} \leftarrow f_{k+2}$ and the compute node set S is correspondingly adjusted, the algorithm returns the required (S, f_{k+2}) pair (since $f_{k+2} < P_2$) resulting in EtS of B kWh as seen in Figure 6.1 (d).

[9]The $IncreaseUserBudget(counter)$ is a customizable, by data center operator, function.

6.5.4. Optimality of the Solution

Although, the optimality is implied through the direct consequence of the TtS and APC being monotone and the resulting EtS function being convex, this section presents the detailed formal proof of the energy-wise optimality of the presented algorithm.

Assume the opposite - for a given application A with n number of compute nodes request, the determined by the algorithm pair (S_k, f_k) is not the energy-optimal one, i.e. $\exists(S', f')$ such that $EtS(n, f', S') < EtS(n, f_k, S_k)$ and $APC(n, f', S') \leq P_c$ and $TtS(n, f') \leq T_d$ for given P_c power consumption and T_d execution time constraints. The following will show that this assumption is wrong.

The algorithm starts with determining the energy-wise optimal f'' CPU frequency for the given application A with n number of compute nodes request. This means that $\forall f \in \{f_1, ..., f_q\}$ target CPU supported frequencies, the $EtS(n, f'', S'') \leq EtS(n, f, S)$, where S is any set of size n currently available compute nodes, and S'' is the set of n compute nodes determined by the function $getBestAvailableNodes(f'', n)$. Depending on the specified T_d and P_c constraints, the following four cases are possible:

$$\textbf{I}\begin{cases} TtS(n, f'') > T_d \\ APC(n, f'', S'') \leq P_c \end{cases}$$

$$\textbf{II}\begin{cases} TtS(n, f'') \leq T_d \\ APC(n, f'', S'') \leq P_c \end{cases}$$

$$\textbf{III}\begin{cases} TtS(n, f'') \leq T_d \\ APC(n, f'', S'') > P_c \end{cases}$$

$$\textbf{IV}\begin{cases} TtS(n, f'') > T_d \\ APC(n, f'', S'') > P_c \end{cases}$$

- **I case.** $TtS(n, f'') > T_d$ implies $f' > f''$, since the opposite would mean that $TtS(n, f') \geq TtS(n, f'') > T_d$[10] which would contradict the assumption of $TtS(n, f') \leq T_d$. According to the algorithm, $EtS(n, f'', S'') \leq EtS(n, f', S')$ and the estimated f'' will be stepwise increased until the $TtS(n, f'') \leq T_d$ (while checking that the P_c constraint does not start to violate). Note that at each stepwise increase phase the $EtS(n, f'', S'') \leq EtS(n, f', S')$[11]. Since according to the above assumption $\exists f'$ such that $TtS(n, f') \leq T_d$ and $APC(n, f', S') \leq P_c$, then this process will terminate with $f_k \leq f'$ such that $TtS(n, f_k) \leq T_d$. Thus the assumption of $EtS(n, f', S') < EtS(n, f_k, S_k)$ was incorrect.

[10]For a fixed n, $TtS(n, f)$ is a monotonically non-increasing function, since $TtS(n, f) = \frac{k}{f}$, where k is a constant fitting parameter (Subsection 4.2.1).

[11]Since for a fixed n, S, and $\forall f > f''$ $EtS(n, f, S) = TtS(n, f) \cdot APC(n, f, S)$ is a monotonically non-decreasing function.

- **II case.** Since $TtS(n, f'') \leq T_d$ and $APC(n, f'', S'') \leq P_c$ then the algorithm terminates with the (S'', f'') configuration. Since $\forall f \in \{f_1, ..., f_q\}$ target CPU supported frequencies $EtS(n, f'', S'') \leq EtS(n, f, S)$ (where S is any set of size n currently available compute nodes) then $EtS(n, f'', S'') \leq EtS(n, f', S')$ which also contradicts the above assumption.

- **III case.** $APC(n, f'', S'') > P_c$ implies that $f' < f''$, since the opposite would imply that $APC(n, f', S') \geq APC(n, f'', S'') > P_c$[12] which would contradict the above $APC(n, f', S') \leq P_c$ assumption. According to the algorithm, $EtS(n, f'', S'') \leq EtS(n, f', S')$ and the estimated f'' will be stepwise decreased (and the compute node set S'' correspondingly updated) until $APC(n, f'', S'') \leq P_c$ (while checking that the T_d constraint does not start to violate). Note that at each stepwise decrease phase the $EtS(n, f'', S'') \leq EtS(n, f', S')$[13]. Since according to the above assumption $\exists(S', f')$ pair such that $APC(n, f', S') \leq P_c$, then this process will terminate with $f_k \geq f'$ such that $APC(n, f_k, S_k) \leq P_c$. Thus the assumption of $EtS(n, f', S') < EtS(n, f_k, S_k)$ was incorrect.

- **IV case.** In this case the algorithm will terminate with a "not schedulable" result, since:
  - $TtS(n, f'') > T_d$ implies $f' > f''$; and
  - $APC(n, f'', S'') > P_c$ implies that $f' < f''$

  contradicting the above existence assumption.

All the four possible cases showed that the contradicting assumption was wrong, which proves the optimality of the algorithm.

6.5.5. Reducing the Processing Time Complexity

The $T(q, m) = \mathrm{O}(qm)$ total processing time complexity, which arises due to the selection of the best **available** compute nodes for a given frequency f (line 28) during each stepwise frequency decrease, can be further reduced to the $\mathrm{O}(q + m)$ via the following modifications:

- ***Considering The Worst Compute Nodes.*** Line 28 can be modified to consider the worst existing compute nodes instead of the best available ones, i.e. $S \leftarrow getWorstExistingNodes(f_{minEtS}, n)$. Since, if a given frequency f with the worst existing n compute node set $S_{worstExisting}$ satisfies the average power consumption constraint, then it will satisfy that constraint with the best available. Thus the line 44 must be further modified to return the

[12]For a fixed n and S, $APC(n, f, S)$ is a monotonically increasing function, since $APC(n, f, S) = k \cdot f^3$, where k is a constant fitting parameter (Subsection 4.2.1).

[13]Since for a fixed n, S, and $\forall f < f''$ $EtS(n, f, S)$ is a monotonically non-decreasing function (Subsection 4.2.1).

$(getBestAvailableNodes(f_{minEtS}, n), f_{minEtS})$ pair. The drawback of this modification is that the algorithm will not always find the optimal resource configuration that will make the full use of the still available power.

- ***Considering The Average Power Behavior.*** The while loop of lines $19 - 42$ can be modified to consider the application APC independent of the n compute node set. This modification will lead to "closer-to-optimal" solution, but will require an additional power constraint satisfiability check for the estimated CPU frequency with the currently available best compute node set, since the latter could result in a higher value than the one estimated on the average basis.

6.6. Configuration Adviser - A Framework of Energy Efficient Constraint Scheduling

Figure 6.6 illustrates a simple scheduling workflow and the role of the suggested Configuration Adviser (CA) framework that serves as a plug-in for a given Resource Management and Scheduling System (RMSS).

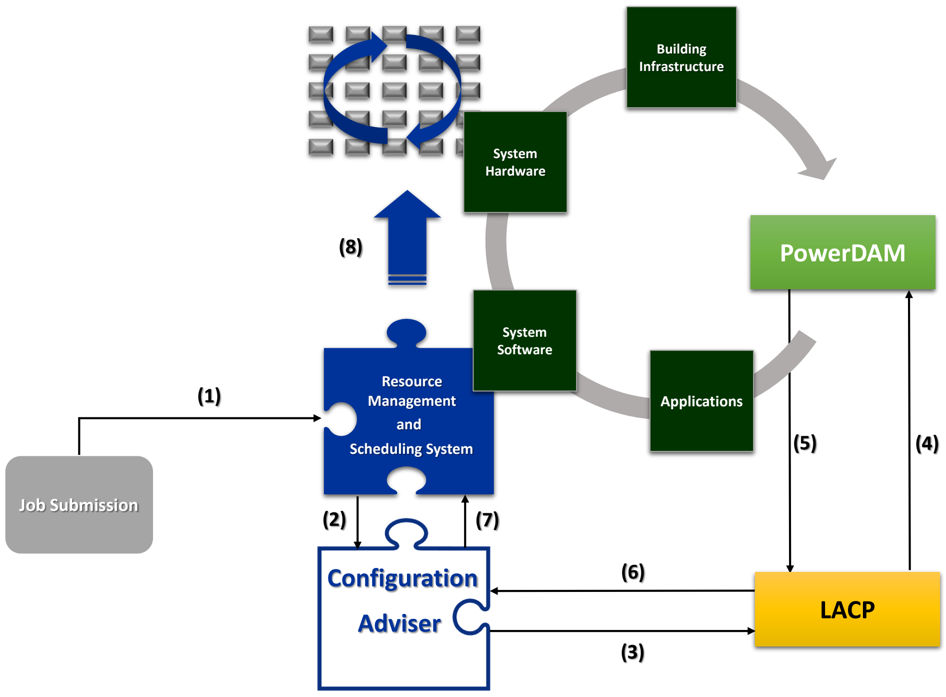


Figure 6.6.: Reference flow of the Energy Consumption Management Adviser framework

In a simple scheduling workflow scenario, based on the RMSS-specific configurations (e.g. application priority, ratio of available and requested number of compute resources, user priority, etc.), the scheduler selects an application and passes the application relevant specification, i.e.

- application identifier,
- number of to-be-utilized compute resources,
- specified maximum execution time, along with
- currently available power budget of the target HPC system (obtained through PowerDAM)

to the CA (step 2, Figure 6.6). The CA on its turn passes these data to the LACP model for obtaining the application EtS, APC, and TtS behavior describing functions (step 3), i.e. functions that will report the values of the mentioned three metrics for any given n number of compute nodes, f CPU frequency, and if specified, set S of n compute nodes. For this reason, LACP queries the application-relevant history data from PowerDAM (step 4). Once the history data is obtained (step 5), LACP determines the EtS, APC, and TtS application-specific functions and returns them to CA (step 6). Having the application energy, power, and execution time describing functions, the CA uses the described above algorithm to find an energy-optimal RC that will meet the specified APC and TtS constraints. Figure 6.7 recaptures the detailed workflow of the CA.

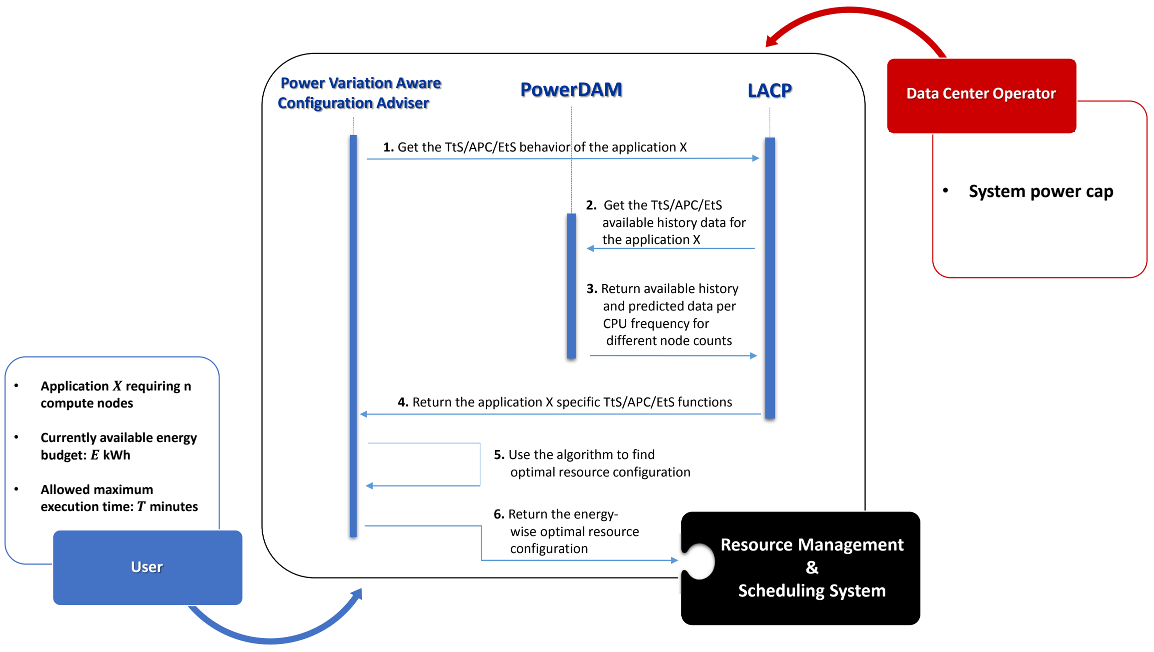

Power Variation Aware Configuration Adviser
PowerDAM
LACP
Data Center Operator
System power cap
1. Get the TtS/APC/EtS behavior of the application X
2. Get the TtS/APC/EtS available history data for the application X
3. Return available history and predicted data per CPU frequency for different node counts
4. Return the application X specific TtS/APC/EtS functions
5. Use the algorithm to find optimal resource configuration
6. Return the energy-wise optimal resource configuration
Application X requiring n compute nodes
Currently available energy budget: E kWh
Allowed maximum execution time: T minutes
User
Resource Management & Scheduling System

Figure 6.7.: Configuration Adviser Workflow

Once an energy-optimal RC is found, the CA transfers this data to the RMSS of the target HPC system (step 7, Figure 6.6). The obtained RC is then used by the RMSS for scheduling the application (step 8). CA sends a "application not schedulable" message to the RMSS in case there does not exist any RC satisfying the pre-defined power and execution time constraints.

6.7. Summary

This chapter presented a lightweight, software-side framework, referred to as Configuration Adviser (CA), that allows for controlling the average power consumption of a large-scale HPC system by implementing a RMSS plug-in, that selects per-application energy consumption wise optimal resource configuration for user specified application execution time constraint. The following bullet points summarize the main contributions that were maid throughout this chapter.

- It was shown that the distribution of the APC of the compute nodes of a given *homogeneous* (in terms of the installed compute nodes) HPC system changes with the maximum operating CPU frequency. The shown node power variation is also an important consideration point when determining the node sample size required for estimating the power of the complete HPC system.
- It was shown that simple compute node pre-selection techniques can already lead to a significant amount of energy/power savings.
- An efficient algorithm (that took into consideration the *inhomogeneous* behavior of the compute nodes of a *homogeneous* system) was presented for determining a possible operating frequency which would minimize the aggregated energy consumption (i.e. Energy-to-Solution) of a given application preserving the predefined application execution time limit and system average power consumption constraint.
- Due to the variations in the power efficiency of the compute nodes and diversity in compute resource requirements and power/energy profiles of the applications the possibility of scheduling (in an energy-wise optimal way) more applications within given power and/or execution time constraints at a given point in time could increase when the suggested algorithm is applied.

The suggested CA plug-in can also be used as a testing and admission-for-execution control framework by a RMSS of the target HPC system, for estimating the potential violations achieved by different resource configurations for a given application.

The proposed CA framework is an important feature for future Exascale HPC data centers where the operating costs would force to support only the expected average system power and not the peak consumption.

7. Conclusion

7.1. Summary

Efficient control of energy and power consumption flows within the modern HPC data centers has become a major challenge. This dissertation provides a mature solution for making the power and energy capped HPC data center a reality. It presented the prerequisites and a framework, referred to as ***Energy Consumption Management Adviser (ECMA)***, for assessing and tuning the energy efficiency of a data center depending on the specified operational policies and user requirements.

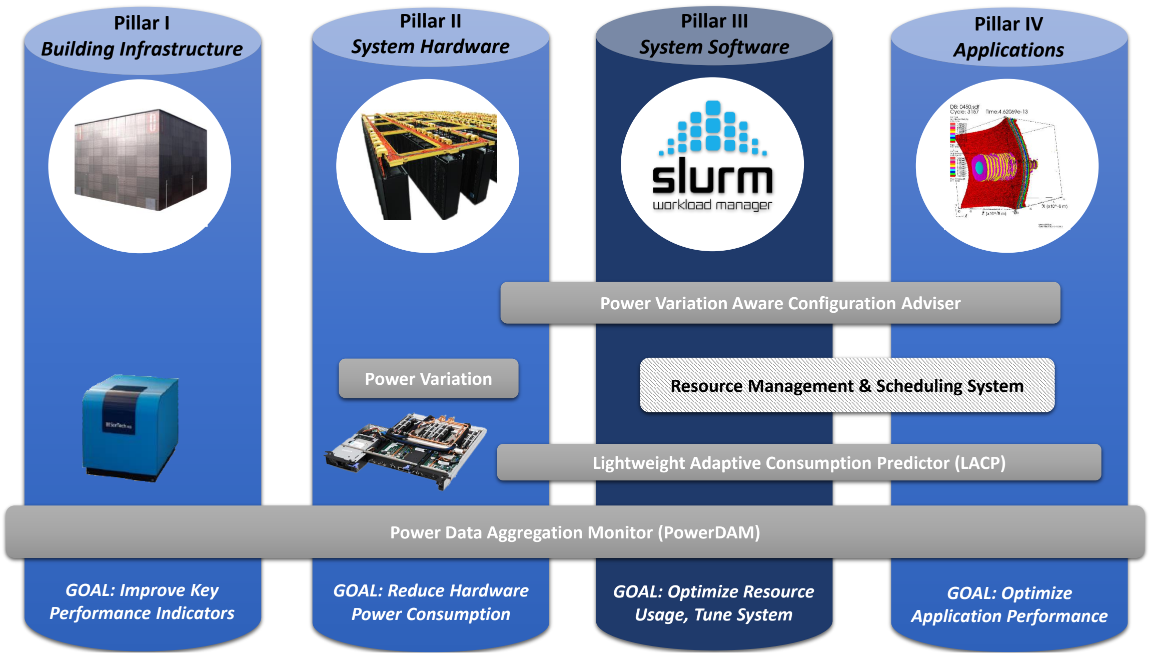

Pillar I
Building Infrastructure
Pillar II
System Hardware
Pillar III
System Software
Pillar IV
Applications
slurm
workload manager
Power Variation Aware Configuration Adviser
Power Variation
Resource Management & Scheduling System
Lightweight Adaptive Consumption Predictor (LACP)
Power Data Aggregation Monitor (PowerDAM)
GOAL: Improve Key Performance Indicators
GOAL: Reduce Hardware Power Consumption
GOAL: Optimize Resource Usage, Tune System
GOAL: Optimize Application Performance

Figure 7.1.: Dissertation coverage area

For achieving this goal it was necessary to cover all the related aspects and factors as identified through the 4 pillar framework (Figure 1.3, Figure 7.1). Figure 7.1 illustrates the main input of this dissertation in strengthening the 4 pillars.

The subsequent paragraphs describe the main milestones that were accomplished throughout the ECMA framework development.

A unified energy measuring and evaluation toolset, referred to as Power Data Aggregation Monitor (PowerDAM), was developed. The developed toolset is aimed towards collecting and correlating energy consumption-relevant data from all the identified 4 pillars (Figure 7.1) of the data center: *from* building infrastructure (e.g. cold and warm water cooling loops, heat reuse technologies, etc.) *over* deployed IT systems (e.g. compute node power, load, temperature, etc.) and their software stack (e.g. Resource Management and Scheduling System (RMSS), etc.) *to* running applications (e.g. utilized compute nodes, power/energy consumption, execution time, etc.).

Further, a model, referred to as Adaptive Energy and Power Consumption Predictor (AEPCP), was developed to allow for predicting the application power and energy consumptions for a given number of compute nodes. The developed AEPCP model takes as an input the history execution time/power/energy profiles of the application under the assumption of a fixed or adjusted application input problem sizes (i.e. AEPCP model is validated for applications demonstrating strong or weak scaling). Seeing that the application execution time, power and energy consumptions are also subject to the operating CPU frequency of the compute resources, the AEPCP model was further extended to a model (referred to as Lightweight Adaptive Consumption Predictor (LACP)) in order to allow for estimating application execution time, power, and energy consumptions for a given number of compute nodes and the maximum operating CPU frequency of the compute node cores. The developed model also takes into account the found inhomogeneous compute node power behavior of a given homogeneous HPC system for providing higher accuracy rate in predictions.

Later, a framework, referred to as Configuration Adviser (CA), that operates on top of a given RMSS as a plug-in for increasing the energy-efficient resource management, scheduling, and backfilling decisions was elaborated. This CA plug-in effectively determines the energy-optimal resource configuration for a given to-be-scheduled application under specified execution time and average power consumption constraints.

In overall, the following bullet points summarize the main contributions that have been made to the state of the art:

- explanation of how the application TtS, APC, and EtS boundary curves can be defined from the known theoretical works and how this information can be applied in practice;
- derivation of three analytical models for estimating an application's TtS, APC, and EtS as functions of the number of compute nodes and the CPU frequency at which all the cores of all compute nodes operate;
- demonstration of the concept applicability for application execution time (i.e. TtS), power (i.e. APC), and energy (i.e. EtS) consumption prediction for unknown resource configuration from previously observed data;
- demonstration of *inhomogeneous* compute node power consumption on two different *homogeneous* HPC systems;
- showed that the distribution of the observed power variation of the compute nodes changes with the operating CPU frequency;

- a way of possibly using the inhomogeneous power consumption behavior of compute nodes for power and energy predictions and its integration to AEPCP/LACP models;
- showed that the selection method of compute nodes can lead to significant energy and power savings - in the considered case, at least 4.7%.

The following recaps the main features of the suggested ECMA framework:

✓ dynamic power, energy, and thermal monitoring;

✓ assessment of the execution time, energy and power consumptions of large-scale applications;

✓ assessment of the data center energy efficiency and the affect verification of the applied improvements;

✓ prediction of execution time, power and energy consumptions of large-scale applications for a given RC;

✓ increase of user awareness of application power and energy consumptions;

✓ energy-driven budgeting/charging policies and support for energy capping;

✓ system power threshold violation alerts;

✓ system power capping;

✓ enhanced backfilling for RMSS of the target system; and

✓ energy efficient application scheduling in case of predefined application average power consumption and/or execution time constraints.

7.2. Future Work

The presented results throughout this dissertation opened new research questions and motivated further implementation work. The following bullet points highlight the currently foreseen future research and development directions.

- **Introduction of EtS/APC measurement quality**
  Due to noisy power sensor readings (which can be present in large-scale systems being not equipped with high quality power measurement reporting devices) some of the calculated EtS/APC could be not completely accurate and at the same time could be not completely false. The specification of the measurement "quality" as a weight in the set of available EtS/APC estimations of the applications in the PowerDAM database, will allow for a better accuracy in predictions conducted by the LACP model.

- **Extension of the derived analytical models of APC and EtS, behind the LACP model, with regard to the inlet temperature**

  Figure 7.2 shows the differences between CPU package temperatures and average node power consumption of air cooled compute node (cooled at 23 °C) and water cooled compute node at different inlet temperatures [23].

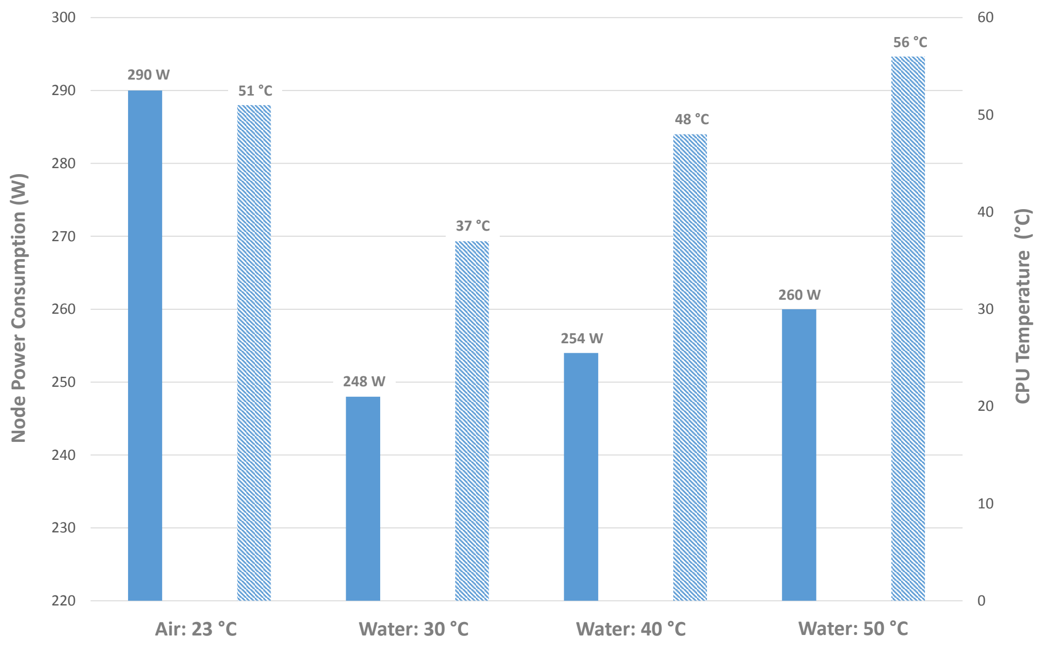


Figure 7.2.: Comparison of node average power consumption and CPU temperature of air-cooled nodes and direct liquid cooled nodes at different inlet temperatures on CoolMUC

  As can be seen, the average power draw for liquid cooled compute nodes stays below the APC of air cooled node, clearly showing the advantage of direct liquid cooling approach versus air cooling. The CPU package temperature also stays below the CPU temperature of air cooled nodes unless the inlet water temperature is higher than 45 °C. These further mean that the APC and EtS of a given application will vary with different inlet temperatures. Thus, the support for predicting the application EtS and APC additionally with respect to the water inlet temperature (which is known to vary rarely) will further improve the overall accuracy of the LACP estimations.

- **ECMA framework tuning to future hybrid HPC systems**
  The power variation in hybrid compute nodes (CPU with accelerators), that are foreseen in the next generation systems, needs to be further analysed.

- **Integration of the developed plug-in with particular RMSS**
  Quantification of the applicability range for the developed plug-in.

- **Integration of the ECMA framework with the power monitoring and management infrastructure of future processors**

  As was described in Chapter 1, the PCU of modern processors already monitors the power consumption of various functional blocks and based on the monitored data performs dynamic power allocations to the various individual system-components. Power management infrastructure of the processors should be enriched to reflect the interaction/interface with the ECMA framework. The presence of such an interface would allow for further dynamic assistance to the existing in-node power/energy consumption management decisions.

- **Development of a web-based graphical user interface for PowerDAM**
  This interface will allow for an easy access and better view of correlated data to data center operators, policy makers, as well as ordinary users.

The suggested *Energy Consumption Management Adviser (ECMA)* framework will be integrated to the software stack at LRZ (pillar III, Figure 7.1) in order to support for energy efficient supercomputing covering and optimizing the full set of in influencing parameters: from building infrastructure over the system hardware to user applications.

Appendices

A. Acronyms

A^2EP^2	Adaptive Application Energy and Power Predictor
AEC	Accumulated Energy Consumption
AEPCP	Adaptive Energy and Power Consumption Predictor
APC	Average Power Consumption
API	Application Programming Interface
ASHRAE	American Society of Heating, Refrigerating and Air-Conditioning Engineers
CA	Configuration Adviser
COP	Coefficient of Performance
CPU	Central Processing Unit
CRAC	Computer Room Air Conditioning
CUE	Carbon Usage Effectiveness
DCIM	Data Center Infrastructure Management
DPM	Dynamic Power Management
DVFS	Dynamic Voltage and Frequency Scaling
DWPE	Data center Workload Power Efficiency
ECMA	Energy Consumption Management Adviser
ERE	Energy Reuse Effectiveness
EtS	Energy-to-Solution
FLOPS	Floating Point Operations Per Second
FNP	Frequency and Node Number Predictor
FPGA	Field Programmable Gate Array
GCS	Gauss Center for Supercomputing
GPU	Graphics Processing Unit

HPC	High Performance Computing
HPCG	High Performance Conjugate Gradient
HPL	High Performance Linpack
IP	Internet Protocol
IT	Information Technology
KPI	Key Performance Indicator
LACP	Lightweight Adaptive Consumption Predictor
LRZ	Leibniz Supercomputing Centre
MPI	Message Passing Interface
MPP	Massively Parallel Processing
MQTT	Message Queue Telemetry Transport
nMOS	N-type Metal-Oxide-Semiconductor
OS	Operating System
PCI	Peripheral Component Interconnect
PCU	Package Control Unit
PDU	Power Distribution Unit
pMOS	P-type Metal-Oxide-Semiconductor
PSNC	Poznan Supercomputing and Networking Center
PowerDAM	Power Data Aggregation Monitor
PRACE	Partnership for Advanced Computing in Europe
%RMSE	Percentage Root Mean Square Error
PUE	Power Usage Effectiveness
RAM	Random-Access Memory
RC	Resource Configuration
RMSE	Root Mean Square Error
RMSS	Resource Management and Scheduling System

SIMOPEK	Simulation and Optimization of Data Center Energy Flows (Simulation und Optimierung des Energiekreislaufs von Rechenzentrums-Klimatisierungsnetzen unter Berücksichtigung von Supercomputer-Betriebsszenarien)
SLURM	Simple Linux Utility For Resource Management
SSD	Solid-State Drive
TCO	Total Cost of Ownership
TCP	Transmission Control Protocol
TDP	Thermal Design Power
TtS	Time-to-Solution
UPS	Uninterruptible Power Supply
URL	Uniform Resource Locator
WP	Work Package
WUE	Water Usage Effectiveness

B. Author's Publications List

Journal Publications

1. Hayk Shoukourian, Torsten Wilde, Axel Auweter, and Arndt Bode. **Monitoring Power Data: A first step towards a unified energy efficiency evaluation toolset for HPC data centers**. Journal of Environmental Modelling & Software, Elsevier, Vol 56, pages 13 – 26, 2013 (impact factor: 4.538). ISSN: 1364 – 8152. DOI: http://dx.doi.org/10.1016/j.envsoft.2013.11.011.

2. Hayk Shoukourian, Torsten Wilde, Axel Auweter, and Arndt Bode. **Predicting the Energy and Power Consumption of Strong and Weak Scaling HPC Applications**. Supercomputing Frontiers and Innovations. Vol 1, No 2, pages 20 – 41, 2014. ISSN: 2313 – 8734. DOI: http://dx.doi.org/10.14529/jsfi1402.

Conference Proceedings

3. Hayk Shoukourian, Torsten Wilde, Axel Auweter, Arndt Bode, and Petra Piochacz. **Towards a unified energy efficiency evaluation toolset: an approach and its implementation at Leibniz Supercomputing Centre (LRZ)**. ICT4S 2013: Proceedings of the First International Conference on Information and Communication Technologies for Sustainability, Zurich, Switzerland, pages 276 – 282, 2013. ISBN: 978 – 3 – 906031 – 24 – 8. DOI: http://dx.doi.org/10.3929/ethz-a-007337628.

4. Torsten Wilde, Axel Auweter, Hayk Shoukourian. **The 4 Pillar Framework for energy efficient HPC data centers**. Computer Science - R&D, Springer, Vol 29, pages 241 – 251, 2013. DOI: http://dx.doi.org/10.1007/s00450-013-0244-6.

5. Torsten Wilde, Axel Auweter, Michael Patterson, Hayk Shoukourian, Herbert Huber, Arndt Bode, Detlef Labrenz, and Carlo Cavazzoni. **DWPE, a new data center energy-efficiency metric bridging the gap between infrastructure and workload**. International Conference on High Performance Computing & Simulation, HPCS, pages 893 – 901, 2014. DOI: http://dx.doi.org/10.1109/HPCSim.2014.6903784.

6. Hayk Shoukourian, Torsten Wilde, Axel Auweter, Arndt Bode, and Daniele Tafani. **Predicting Energy Consumption Relevant Indicators of Strong Scaling HPC Applications for Different Compute Resource Configurations**. Proceedings of the 23rd High Performance Computing Symposium, Society for Modeling and Simulation International (SCS), ACM, 2015.

7. Torsten Wilde, Axel Auweter, Hayk Shoukourian, and Arndt Bode. **Taking advantage of node power variation in homogenous HPC systems to save energy**. In Supercomputing. Springer International Publishing, 2015.

8. Hayk Shoukourian, Torsten Wilde, Axel Auweter, and Arndt Bode. **Power Variation Aware Configuration Adviser for Scalable HPC Schedulers**. Proceedings of the 13 International Conference on High Performance Computing & Simulation (HPCS), IEEE, 2015.

Articles in Media

9. Hayk Shoukourian, Torsten Wilde, Axel Auweter, and Arndt Bode. **A Path To Energy Efficient HPC Datacenters**. ***Featured*** article in HPCWire, 2013. URL: `http://www.hpcwire.com/2013/10/29/path-energy-efficient-hpc-datacenters/`.

Bibliography

[1] Top500. http://top500.org/, 2015.

[2] Jonathan G Koomey, Christian Belady, Michael Patterson, Anthony Santos, and Klaus-Dieter Lange. Implications of recent trends in performance, costs, and energy use for servers. *The Green Computing Book: Tackling Energy Efficiency at Large Scale*, page 297, 2014.

[3] Marc Wouters, James C. Anderson, and Finn Wynstra. The adoption of total cost of ownership for sourcing decisions–a structural equations analysis. *Accounting, Organizations and Society*, 30(2):167–191, February 2005.

[4] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power Provisioning for a Warehouse-sized Computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 13–23, New York, NY, USA, 2007. ACM.

[5] Leibniz Supercomputing Centre (LRZ) of the Bavarian Academy of Sciences and Humanities. http://www.lrz.de/, 2015.

[6] Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet. The LINPACK Benchmark: Past, Present, and Future. *Concurrency and Computation: Practice and Experience*, 15:803-820, 2003.

[7] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R. Stanley Williams, Katherine Yelick, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Stephen Keckler, Dean Klein, Peter Kogge, R. Stanley Williams, and Katherine Yelick. ExaScale Computing study: Technology Challenges in Achieving Exascale Systems, Peter Kogge, Editor & Study Lead, 2008.

[8] Bates N., Ghatikar G., Abdulla G., Koenig G., Bhalachandra S., Sheikhalishahi M., Patki T., Rountree B., and Poole S. The Electrical Grid and Supercomputing Centers: An Investigative Analysis of Emerging Opportunities and Challenges, 2014.

[9] Chung-Hsing Hsu and Wu-Chun Feng. A Power-Aware Run-Time System for High-Performance Computing. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, SC '05, Washington, DC, USA, 2005. IEEE Computer Society.

[10] Vasanth Venkatachalam and Michael Franz. Power reduction techniques for microprocessor systems. *ACM Computing Surveys (CSUR)*, 37(3):195–237, 2005.

[11] Anshul Gandhi, Mor Harchol-Balter, Rajarshi Das, and Charles Lefurgy. Optimal power allocation in server farms. *SIGMETRICS Perform. Eval. Rev.*, 37(1):157–168, June 2009.

[12] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2012.

[13] Padmanabhan Pillai and Kang G Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 89–102. ACM, 2001.

[14] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann. Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge. *Micro, IEEE*, 32(2):20–27, March 2012.

[15] Advanced Configuration and Power Interface Specification. http://www.acpi.info/, November 13, 2013.

[16] IBM Knowledge Center. CPU performance states (P-states) and CPU operating states (C-states). https://www-01.ibm.com/support/knowledgecenter/linuxonibm/liaai.cpufreq/CPUPerformanceStates.htm.

[17] Torsten Wilde, Axel Auweter, and Hayk Shoukourian. The 4 Pillar Framework for energy efficient HPC data centers. *Computer Science - Research and Development*, 29(3-4):241–251, 2013.

[18] American Society of Heating, Refrigerating and Air-Conditioning Engineers. https://www.ashrae.org/home, 2015.

[19] The Green Grid. http://www.thegreengrid.org/, 2015.

[20] Johnson Controls. http://www.johnsoncontrols.com/, 2015.

[21] Siemens. SIMATIC winCC SCADA system. http://w3.siemens.com/mcms/human-machine-interface/en/visualization-software/simatic-wincc-open-architecture/Pages/Default.aspx, 2015.

[22] Mauro Marinoni, Mario Bambagini, Francesco Prosperi, Francesco Esposito, Gianluca Franchino, Luca Santinelli, and Giorgio Buttazzo. Platform-aware bandwidth-oriented energy management algorithm for real-time embedded systems. In *Proceedings of the 16th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–8. IEEE, 2011.

[23] Lennart Johnsson, Gilbert Netzer, Eric Boyer, Stephan Graf, Wilhelm Homberg, Giannis Koutsou, Josip Jakic, Radek Januszewski, Nikola Puzovic, Thomas Roeblitz, Ole Widar Saastad, Björn Schembera, Georg Schwarz, Hayk Shoukourian, Volker Strumpen, Stephane Thiell, Guillaume Colin de Verdière, and Torsten Wilde. PRACE 1IP-WP9 D9.3.3 Report on prototypes evaluation. http://www.prace-ri.eu/IMG/pdf/d9.3.3.pdf, 2013.

[24] Neil H.E. Weste and David Money Harris. *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2005.

[25] Intel. Node Manager - A Dynamic Approach To Managing Power In The Data Center. https://communities.intel.com/docs/DOC-4766?wapkw=intelligent+power+node+manager+whitepaper, 2010.

[26] Barry Rountree, Dong H Ahn, Bronis R de Supinski, David K Lowenthal, and Martin Schulz. Beyond DVFS: A First Look at Performance Under a Hardware-Enforced Power Bound. In *Proceedings of the Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, pages 947–953. IEEE, 2012.

[27] 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 43rd International Conference on Parallel Processing (ICPP)*, pages 371–380. IEEE, 2014.

[28] Axel Auweter, Arndt Bode, Matthias Brehm, Luigi Brochard, Nicolay Hammer, Herbert Huber, Raj Panda, Francois Thomas, and Torsten Wilde. A Case Study of Energy Aware Scheduling on SuperMUC. In *Supercomputing*, pages 394–409. Springer, 2014.

[29] Osman Sarood, Akhil Langer, Abhishek Gupta, and Laxmikant Kalé. Maximizing throughput of overprovisioned HPC data centers under a strict power budget. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 807–818. IEEE Press, 2014.

[30] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel computing*, 22(6):789–828, 1996.

[31] Osman Sarood, Akhil Langer, Laxmikant Kalé, Barry Rountree, and Bronis De Supinski. Optimizing Power Allocation to CPU and Memory Subsystems in Overprovisioned HPC Systems. In *Cluster Computing (CLUSTER)*, pages 1–8. IEEE, 2013.

[32] Engin Ipek, Bronis R De Supinski, Martin Schulz, and Sally A McKee. An approach to performance prediction for parallel applications. In *Euro-Par Parallel Processing*, pages 196–205. Springer, 2005.

[33] G.L. Tsafack Chetsa, L. Lefèvre, J.M. Pierson, P. Stolf, and G. Da Costa. Exploiting performance counters to predict and improve energy performance of HPC systems. *Future Generation Computer Systems*, 2013.

[34] Georg Hager, Jan Treibig, Johannes Habich, and Gerhard Wellein. Exploring performance and power properties of modern multi-core chips via simple machine models. *Concurrency and Computation: Practice and Experience*, 2014.

[35] Luigi Brochard, Raj Panda, and Sid Vemuganti. Optimizing performance and energy of HPC application on POWER7. *Computer Science - Research and Development*, 25(3-4):135–140, 2010.

[36] Can Hankendi and Ayse K Coskun. Adaptive power and resource management techniques for multi-threaded workloads. In *Proceedings of the 27th International Symposium on Parallel and Distributed Processing Workshops & PhD Forum*, pages 2302–2305. IEEE Computer Society, 2013.

[37] Catherine Mills Olschanowsky, Tajana Rosing, Allan Snavely, Laura Carrington, Mustafa M Tikir, and Michael Laurenzano. Fine-grained energy consumption characterization and modeling. In *Proceedings of the High Performance Computing Modernization Program Users Group Conference (HPCMP-UGC), 2010 DoD*, pages 487–497. IEEE, 2010.

[38] Shuaiwen Leon Song, Kevin Barker, and Darren Kerbyson. Unified performance and power modeling of scientific workloads. In *Proceedings of the 1st International Workshop on Energy Efficient Supercomputing*, E2SC '13, pages 4:1–4:8, New York, NY, USA, 2013. ACM.

[39] Axel Auweter, Arndt Bode, Matthias Brehm, Herbert Huber, and Dieter Kranzlmüller. Principles of Energy Efficiency in High Performance Computing. In *Proceedings of the first international conference on information and communication on technology for the fight against global warming*, ICT-GLOW'11, pages 18–25, Berlin, Heidelberg, 2011. Springer-Verlag.

[40] G. Da Costa, J.-P. Gelas, Y. Georgiou, L. Lefevre, A.-C. Orgerie, J.-M. Pierson, O. Richard, and K. Sharma. The GREEN-NET Framework: Energy Efficiency in Large Scale Distributed Systems. In *Proceedings of the International Symposium on Parallel Distributed Processing (IPDPS)*, pages 1–8. IEEE, May 2009.

[41] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and K.W. Cameron. PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications. *Transactions on Parallel and Distributed Systems*, 21(5):658–671, May 2010.

[42] Adam Wade Lewis, Soumik Ghosh, and Nian-Feng Tzeng. Run-time Energy Consumption Estimation Based on Workload in Server Systems. *HotPower*, 8:17–21, 2008.

[43] Karan Singh, Major Bhadauria, and Sally A McKee. Real time power estimation and thread scheduling via performance counters. *ACM SIGARCH Computer Architecture News*, 37(2):46–55, 2009.

[44] M. Witkowski, A. Oleksiak, T. Piontek, and J. Węglarz. Practical power consumption estimation for real life HPC applications. *Future Generation Computer Systems*, 29(1):208 – 217, 2013.

[45] Rackwise Data Center Software. http://www.rackwise.com/, 2011.

[46] openDCIM - Web Based Data Center Infrastructure Management Application. http://www.opendcim.org, 2012.

[47] Hayk Shoukourian, Torsten Wilde, Axel Auweter, Arndt Bode, and Petra Piochacz. Towards a unified energy efficiency evaluation toolset: an approach and its implementation at Leibniz Supercomputing Centre (LRZ). In *ICT4S 2013: Proceedings of the First International Conference on Information and Communication Technologies for Sustainability*, pages 276–282, 2013.

[48] Hayk Shoukourian, Torsten Wilde, Axel Auweter, and Arndt Bode. Monitoring Power Data: A first step towards a unified energy efficiency evaluation toolset for HPC data centers. *Environmental Modelling & Software*, 56, 2013. Thematic issue on modelling and evaluating the sustainability of smart solutions.

[49] Hayk Shoukourian, Torsten Wilde, Axel Auweter, and Arndt Bode. A Path To Energy Efficient HPC Datacenters. http://www.hpcwire.com/2013/10/29/path-energy-efficient-hpc-datacenters/, 2013.

[50] Torsten Wilde, Axel Auweter, Michael K Patterson, Hayk Shoukourian, Herbert Huber, Arndt Bode, Detlef Labrenz, and Carlo Cavazzoni. DWPE, a new data center energy-efficiency metric bridging the gap between infrastructure and workload. In *Proceedings of the International Conference on High Performance Computing & Simulation (HPCS)*, pages 893–901. IEEE, 2014.

[51] Python. https://www.python.org/, 2015.

[52] MEGWARE Computer Vertrieb und Service GmbH. http://www.megware.com/en/default.aspx, 2015.

[53] SorTech AG. http://www.sortech.de/en/, 2015.

[54] Simple Linux Utility for Resource Management. http://slurm.schedmd.com/, 2015.

[55] Gauss Centre for Supercomputing. http://www.gauss-centre.eu, 2015.

[56] Partnership for Advanced Computing in Europe. http://www.prace-ri.eu/, 2015.

[57] International Business Machines Corporation (IBM). http://www.ibm.com, 2015.

[58] IBM: Tivoli workload scheduler LoadLeveler. http://www.ibm.com/systems/software/loadleveler/, 2015.

[59] Arndt Bode. Energy to Solution: A New Mission for Parallel Computing. In *Euro-Par*, pages 1–2. Springer, 2013.

[60] Pierre-François Lavallée, Guillaume Colin de Verdière, Philippe Wautelet, Dimitri Lecas, and Jean-Michel Dupays. Porting and optimizing HYDRO to new platforms and programming paradigms - lessons learnt. http://www.prace-ri.eu/IMG/pdf/porting_and_optimizing_hydro_to_new_platforms.pdf, December 2012.

[61] Romain Teyssier. The RAMSES Code. http://irfu.cea.fr/Phocea/Vie_des_labos/Ast/ast_sstechnique.php?id_ast=904, 2013.

[62] Sergei Konstantinovich Godunov. A difference method for numerical calculation of discontinuous solutions of the equations of hydrodynamics. *Matematicheskii Sbornik*, 89(3):271–306, 1959.

[63] Philip L Roe. Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of computational physics*, 43(2):357–372, 1981.

[64] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[65] Mosquitto. An Open Source MQTT v3.1/v3.1.1 Broker. http://mosquitto.org/, 2015.

[66] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys (CSUR)*, 35(2):114–131, June 2003.

[67] Poznan Supercomputing and Networking Center. http://www.man.poznan.pl/online/en/, 2015.

[68] Iceotope. http://www.iceotope.com/, 2015.

[69] CINECA. http://www.cineca.it/en, 2015.

[70] Green500. http://www.green500.org/, 2015.

[71] Altair. http://www.altair.com/, 2015.

[72] The SIMOPEK Project. http://simopek.de/, 2015.

[73] Multiphysical Network Simulator MYNTS. http://www.scai.fraunhofer.de/en/business-research-areas/high-performance-analytics-en/products/mynts.html, 2015.

[74] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the Spring Joint Computer Conference*, pages 483–485. ACM, 1967.

[75] John L Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31(5):532–533, 1988.

[76] Hayk Shoukourian, Torsten Wilde, Axel Auweter, and Arndt Bode. Predicting the Energy and Power Consumption of Strong and Weak Scaling HPC Applications. *Supercomputing Frontiers And Innovations, Vol 1, No 2*, pages 20–41, 2014.

[77] Yuan Shi. Reevaluating Amdahl's law and Gustafson's law. *Computer Sciences Department, Temple University (MS: 38-24)*, 1996.

[78] Dong Hyuk Woo and Hsien-Hsin S Lee. Extending Amdahl's Law for Energy-Efficient Computing in the Many-Core Era. *Computer*, 41(12):24–31, 2008.

[79] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*, volume 2. MIT press Cambridge, 2001.

[80] Michael McCool, James Reinders, and Arch Robison. *Structured parallel programming: patterns for efficient computation*. Elsevier, 2012.

[81] Watts S. Humphrey. *A Discipline for Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[82] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, Cambridge, MA, 2012.

[83] Richard Hamming. *Numerical methods for scientists and engineers*. Courier Corporation, 2012.

[84] Leonid Yavits, Amir Morad, and Ran Ginosar. The effect of communication and synchronization on Amdahl's law in multicore systems. *Parallel Computing*, 40(1):1–16, 2014.

[85] T. Arber and et al. EPOCH: Extendable PIC Open Collaboration. http://ccpforge.cse.rl.ac.uk/gf/project/epoch/, 2015.

[86] Hartmut Ruhl. Classical Particle Simulations with the PSC code. https://www.physik.uni-muenchen.de/lehre/vorlesungen/wise_09_10/tvi_mas_compphys/vorlesung/Lecturescript.pdf, 2005.

[87] E. Oran Brigham. *The Fast Fourier Transform and Its Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[88] Great Internet Mersenne Prime Search. http://www.mersenne.org/freesoft/, 2015.

[89] W. Shockley. Problems related to p-n junctions in silicon. *Uspekhi Fizicheskikh Nauk*, 77(1):161–196, 1962.

[90] Franck Cappello, Al Geist, Bill Gropp, Laxmikant Kale, Bill Kramer, and Mark Snir. Toward exascale resilience. *International Journal of High Performance Computing Applications*, 2009.

[91] Rong Ge, Xizhou Feng, and Kirk W Cameron. Modeling and evaluating energy--performance efficiency of parallel processing on multicore based power aware systems. In *Proceedings of the International Symposium on Parallel & Distributed Processing, (IPDPS)*, pages 1–8. IEEE, 2009.

[92] Hayk Shoukourian, Torsten Wilde, Axel Auweter, Arndt Bode, and Daniele Tafani. *Predicting Energy Consumption Relevant Indicators of Strong Scaling HPC Applications for Different Compute Resource Configurations*. Proceedings of the 23rd High Performance Computing Symposium, Society for Modeling and Simulation International (SCS), 2015.

[93] HP Staff. HP power capping and dynamic power capping for ProLiant servers. Technical report, HP, Tech. Rep. TC090303TB, 2009.

[94] Harry J.M. Veendrick. Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits. *Journal of Solid-State Circuits*, 19(4):468–473, 1984.

[95] Thomas Rauber and Gudula Rünger. *Parallel programming: For multicore and cluster systems*. Springer Science & Business, 2013.

[96] David Blaauw, Steve Martin, Trevor Mudge, and Krisztián Flautner. Leakage current reduction in VLSI systems. *Journal of Circuits, Systems, and Computers*, 11(06):621–635, 2002.

[97] M Gasca. Multivariate polynomial interpolation. In *Computation of curves and surfaces*, pages 215–236. Springer, 1990.

[98] The R Project for Statistical Computing. http://www.r-project.org/, 2015.

[99] The ARM Cortex-A15 processor. http://www.arm.com/products/processors/cortex-a/cortex-a15.php, 2014.

[100] The Mont-Blanc Project. http://www.montblanc-project.eu, 2015.

[101] Ricardo Gonzalez, Benjamin M. Gordon, and Mark A. Horowitz. Supply and Threshold Voltage Scaling for Low Power CMOS. *Journal of Solid-State Circuits*, 32(8):1210–1216, 1997.

[102] Chong-Min Kyung and Sungjoo Yoo. *Energy-Aware System Design: Algorithms and Architectures*. Springer Science & Business Media, 2011.

[103] Hayk Shoukourian, Torsten Wilde, Axel Auweter, and Arndt Bode. *Power Variation Aware Configuration Adviser for Scalable HPC Schedulers*. Proceedings of the 13 International Conference on High Performance Computing & Simulation, HPCS, 2015.

[104] Yong Fu, Nicholas Kottenstette, Chenyang Lu, and Xenofon D Koutsoukos. Feedback thermal control of real-time systems on multicore processors. In *Proceedings of the tenth ACM international conference on Embedded software*, pages 113–122. ACM, 2012.

[105] D. Hackenberg, R. Oldenburg, D. Molka, and R. Schone. Introducing FIRESTARTER: A processor stress test utility. In *Proceedings of the International Green Computing Conference (IGCC)*, pages 1–9, June 2013.

[106] Rod Mahdavi. Liquid Cooling v. Air Cooling Evaluation in the Maui High Performance Computing Center. U.S. Department of Energy, 2014.

[107] Jack Dongarra and Michael A Heroux. Toward a New Metric for Ranking High Performance Computing Systems. *Sandia Report, SAND2013-4744*, 312, 2013.

[108] Torsten Wilde, Axel Auweter, Hayk Shoukourian, and Arndt Bode. Taking advantage of node power variation in homogenous HPC systems to save energy. In *Supercomputing*. Springer, 2015.