Doctoral Dissertation Manuscript

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Descartes
research

Measurement, Modeling, and Emulation of Power Consumption of Distributed Systems

Norbert Schmitt
Department of Computer Science
Julius-Maximilians-Universität Würzburg

Prof. Dr.-Ing. Samuel Kounev (*Reviewer and Advisor*)
Department of Computer Science
Julius-Maximilians-Universität Würzburg

November 29, 2021

www.uni-wuerzburg.de

Norbert Schmitt

Measurement, Modeling, and Emulation of Power Consumption of Distributed Systems

Dissertation, Julius-Maximilians-Universität Würzburg
Fakultät für Mathematik und Informatik, 2021

Gutachter: Prof. Dr.-Ing. Samuel Kounev, Julius-Maximilians-Universität Würzburg
Prof. Dr. Diwakar Krishnamurthy, University of Calgary

Abstract

Today's cloud data centers consume an enormous amount of energy, and energy consumption will rise in the future. An estimate from 2012 found that data centers consume about 30 billion watts of power, resulting in about $263\,TWh$ of energy usage per year. The energy consumption will rise to $1929\,TWh$ until 2030. This projected rise in energy demand is fueled by a growing number of services deployed in the cloud. 50% of enterprise workloads have been migrated to the cloud in the last decade so far. Additionally, an increasing number of devices are using the cloud to provide functionalities and enable data centers to grow. Estimates say more than 75 billion IoT devices will be in use by 2025.

The growing energy demand also increases the amount of CO_2 emissions. Assuming a CO_2-intensity of $200\,g$ CO_2 per kWh in a rather optimistic scenario will get us close to 227 billion tons of CO_2. This emission is more CO_2 than the emissions of all energy-producing power plants in Germany in 2020.

Many technologies exist to increase the energy efficiency of data centers on the hardware level. Efficiency benchmarks were created as an incentive for manufacturers to improve energy efficiency. Through public results and the contribution of the industry, server energy efficiency has increased.

However, data centers consume energy because they respond to service requests that are fulfilled through computing resources like processors, memory, storage, and network. Hence, it is not the users and devices that consume the energy in the data center but the software that controls the hardware. While the hardware is physically consuming energy, it is not always responsible for wasting energy. The software itself plays a vital role in reducing the energy consumption and CO_2 emissions of data centers. The scenario of our thesis is, therefore, focused on software development.

Nevertheless, we must first show developers that software contributes to energy consumption by providing evidence of its influence. The second step is to provide methods to assess an application's power consumption during different phases of the development process and to allow modern DevOps and agile development methods. We, therefore, need to have an automatic selection of system-level energy-consumption models that can accommodate rapid changes in the source code and application-level models allowing developers to locate

power-consuming software parts for constant improvements. Afterward, we need emulation to assess the energy efficiency before the actual deployment. To summarize, we identified four main issues our thesis addresses:

1. Developers lack awareness that they can significantly influence the energy efficiency of software.
2. Fine-granular power models are necessary to help developers identify energy-consuming parts of the application.
3. Constantly and rapidly changing applications necessitate an automated approach to select the best power model.
4. Automated and frequent deployments in a DevOps context need an improved emulation of an application's power consumption.

This thesis contains six contributions to address the problems. The first two contributions focus on achieving a higher awareness while the other contributions are aimed at the remaining problems.

- *Case Study Demonstrating the Impact of Compiler Optimizations*
  This contribution presents a case study on the influence of compiler optimizations on power consumption and energy efficiency. We performed the case study with the SPEC CPU©2017 benchmark suite that provides a wide variety of software artifacts. We ran the benchmarks in two scenarios, a real-world scenario with typical compiler optimizations and an optimized scenario where experts have tuned compiler settings to achieve the best possible benchmark score. The results are evaluated regarding their performance, power consumption, energy efficiency, and whether a C-like or a functional programming language can be optimized better. We show that an increase of about 30% in energy efficiency is possible for most benchmarks, leaving ample room for improvements that need no change in source code.
- *Case Study Demonstrating the Impact of Sorting Algorithms*
  The second contribution shows developers can improve the energy efficiency with their choices. This contribution shows the significant impact a simple choice, such as the correct sorting algorithm can have on the energy efficiency. We evaluate the results from this case study according to their energy efficiency in bytes sorted per watt-second. The results show that algorithms with the same average time complexity of $n * \log(n)$ can vary widely in energy efficiency, and developers should select algorithms

with care. Additionally, an algorithm must be implemented with care. Quick Sort experiences a drop of over $84MB/Ws$ when implemented recursively instead of iteratively. We also compiled a selection of rules for each algorithm with logarithmic runtime complexity that practitioners can use as a rule-of-thumb.

- *Characterizing Software Resource Usage Profiles*
  The third contribution of this thesis is the automatic classification of software according to their resource usage profile extracted by the released performance events during execution. Extracting a resource usage profile allows one to automatically choose an appropriate and more accurate power model. The resource usage profiles have been trained on a regular x86 server with over 100 benchmarks and different configuration options. The same benchmarks train two different power models to show that our approach can select a suitable power model to increase accuracy. Our evaluation shows that CPU-intensive workloads are well represented in a first data exploration step, while memory-, and I/O-intensive workloads are sparse, making the classification difficult. Still, we could achieve an average accuracy of over 80%. We could also show an increase in accuracy from 12.7% to 9.7%, even if we only use two distinct power models. Neither the classification nor the two power models were trained with the application used in our evaluation. This contribution allows developers to monitor unknown applications and even select accurate power models for the current resource usage profile during its execution.

- *Fine-Granular Power Model for Energy-Efficient Development*
  We propose a new fine-grained power model to guide developers in identifying the most power-consuming parts of their application. Our approach uses a stack trace determining the function on top of the stack and attributing triggered performance events to this function for the time it stays on top of the stack. We also introduced a sliding average correctional factor to reduce the amount of falsely attributed performance events. The evaluation of our approach shows that, after training a linear regression model for the application, we can distinguish between the overhead of the software stack and the application and identify the most power-consuming functions. We evaluated our approach under different load levels, which showed similar results, identifying the same functions in the same order for the different load levels. This approach allows developers to determine which parts of their application are responsible for

the highest energy consumption and should be targeted when optimizing energy efficiency.

- *High-Bandwidth Memory Workload Power Model*
  We define a new power model for high-bandwidth memory workloads. The model is built upon performance events recorded in the CPU, and it divides the total system power consumption into three parts: idle, CPU, and memory power consumption. We evaluate the model with a standard memory benchmark and different data types. The results show that we can compare the memory power consumption of different implementations relatively without assuming knowledge of the underlying hardware or give the absolute memory power consumption if technical details of the underlying hardware are known.

- *PET Improvements for Better Power Emulation*
  The last contribution improves our Performance Event Trigger Framework (PET) with new performance event trigger implementations and an automatic configuration approach to compensate for side effects during execution. We evaluated an improved performance event trigger function to allow PET to emulate applications that cause performance events at a high frequency. Additionally, as the synthetic code might cause unwanted performance events, they must be neglected, reducing accuracy, or compensated. In this contribution, we show that an automatic side effect compensation can improve PET's accuracy. In addition, we evaluated PET on the original server and two additional servers of different generations.

The presented contributions of this thesis provide evidence and raise awareness that energy efficiency is achieved through efficient hardware and can considerably be influenced by the application. Improving energy efficiency starts with a selection of the correct algorithms down to the optimal compiler settings. These findings can raise awareness and can also be used to motivate non-functional properties in a product specification. After deciding to include energy efficiency as a goal during development, developers need the right tools and models to improve energy efficiency. Our approach, therefore, can classify applications according to their resource usage profile enabling an automated model selection for more accurate results. By emulating the software before deployment, we can show developers how their application will behave on the production hardware or help operators make an informed choice on which hardware an application should be deployed. Finally, our last contribution

gives developers information on where energy is consumed in their application, and which parts they should focus on when aiming to improve energy efficiency.

Zusammenfassung

Die heutigen Cloud-Rechenzentren verbrauchen eine enorme Menge an Energie, und der Energieverbrauch wird in Zukunft noch steigen. Eine Schätzung aus dem Jahr 2012 ergab, dass Rechenzentren etwa 30 Milliarden Watt Strom verbrauchen, was einem Energieverbrauch von etwa $263\,TWh$ pro Jahr entspricht. Der Energieverbrauch wird bis zum Jahr 2030 auf $1929\,TWh$ ansteigen. Dieser prognostizierte Anstieg des Energiebedarfs wird durch die wachsende Zahl der in der Cloud bereitgestellten Dienste angeheizt. In den letzten zehn Jahren wurden bereits 50% der Arbeitslasten in Unternehmen in die Cloud verlagert. Außerdem nutzen immer mehr Geräte die Cloud, um Funktionen bereitzustellen und das Wachstum von Rechenzentren zu ermöglichen. Schätzungen zufolge werden bis 2025 mehr als 75 Milliarden IoT-Geräte im Einsatz sein.

Der wachsende Energiebedarf erhöht auch die Menge der CO_2-Emissionen. Geht man von einer CO_2-Intensität von $200g$ CO_2 pro kWh in einem eher optimistischen Szenario aus, kommen wir auf fast 227 Milliarden Tonnen CO_2. Dieser Ausstoß ist mehr CO_2 als die Emissionen aller energieerzeugenden Kraftwerke in Deutschland im Jahr 2020.

Es gibt viele Technologien, um die Energieeffizienz von Rechenzentren auf der Hardware-Ebene zu erhöhen. Als Anreiz für die Hersteller, die Energieeffizienz zu verbessern, wurden Effizienz-Benchmarks geschaffen. Durch die öffentlichen Ergebnisse und den Beitrag der Industrie konnte die Energieeffizienz von Servern gesteigert werden.

Rechenzentren verbrauchen jedoch Energie, weil sie auf Serviceanfragen reagieren, die durch Rechenressourcen wie Prozessoren, Arbeitsspeicher, Speicher und Netzwerke erfüllt werden. Es sind also nicht die Benutzer und Geräte, die in einem Rechenzentrum Energie verbrauchen, sondern die Software, die die Hardware steuert. Obwohl die Hardware physisch Energie verbraucht, ist sie nicht immer für die Energieverschwendung verantwortlich. Die Software selbst spielt eine wichtige Rolle bei der Reduzierung des Energieverbrauchs und der CO_2-Emissionen von Rechenzentren. Das Szenario unserer Arbeit konzentriert sich daher auf die Softwareentwicklung.

Dennoch müssen wir die Entwickler zunächst darauf hinweisen, dass die Software zum Energieverbrauch beiträgt, indem wir ihren Einfluss nachwei-

sen. Der zweite Schritt ist die Bereitstellung von Methoden zur Bewertung des Energieverbrauchs einer Anwendung in den verschiedenen Phasen des Entwicklungsprozesses, um moderne DevOps und agile Entwicklungsmethoden zu ermöglichen. Wir brauchen daher eine automatische Auswahl von Energieverbrauchsmodellen auf Systemebene, die schnelle Änderungen im Quellcode berücksichtigen können, und Modelle auf Anwendungsebene, die es den Entwicklern ermöglichen, stromverbrauchende Softwareteile für ständige Verbesserungen zu lokalisieren. Danach benötigen wir eine Emulation, um die Energieeffizienz vor dem eigentlichen Einsatz zu bewerten. Zusammenfassend haben wir vier Hauptthemen identifiziert, mit denen sich unsere Arbeit befasst:

- *Fallstudie zur Demonstration der Auswirkungen von Compiler-Optimierungen*
  In diesem Beitrag wird eine Fallstudie über den Einfluss von Compiler-Optimierungen auf den Stromverbrauch und die Energieeffizienz vorgestellt. Wir haben die Fallstudie mit der SPEC CPU©2017 Benchmark-Suite durchgeführt, die eine Vielzahl von Software-Artefakten bietet. Wir haben die Benchmarks in zwei Szenarien ausgeführt, einem realen Szenario mit typischen Compiler-Optimierungen und einem optimierten Szenario, bei dem Experten die Compiler-Einstellungen so angepasst haben, dass die bestmögliche Benchmark-Punktzahl erreicht wird. Die Ergebnisse werden hinsichtlich ihrer Leistung, ihres Stromverbrauchs und ihrer Energieeffizienz ausgewertet, und es wird untersucht, ob eine C-ähnliche oder eine funktionale Programmiersprache besser optimiert werden kann. Wir zeigen, dass bei den meisten Benchmarks eine Steigerung der Energieeffizienz um etwa 30% möglich ist, was reichlich Spielraum für Verbesserungen lässt, die keine Änderung des Quellcodes erfordern.

- *Fallstudie zum Nachweis der Auswirkungen von Sortieralgorithmen*
  Der zweite Beitrag zeigt, dass Entwickler die Energieeffizienz durch ihre Entscheidungen verbessern können. Er zeigt, welch erheblichen Einfluss eine einfache Entscheidung, wie die Wahl des richtigen Sortieralgorithmus, auf die Energieeffizienz haben kann. Wir bewerten die Ergebnisse aus dieser Fallstudie nach ihrer Energieeffizienz in sortierten Bytes pro Wattsekunde. Die Ergebnisse zeigen, dass Algorithmen mit der gleichen durchschnittlichen Zeitkomplexität von $n * \log(n)$ in Bezug auf die Energieeffizienz sehr unterschiedlich sein können, und Entwickler sollten Algorithmen mit Bedacht auswählen. Außerdem muss ein Algorithmus mit Sorgfalt implementiert werden. Quick Sort verzeichnet eine Reduktion von über $84MB/Ws$, wenn er rekursiv statt iterativ implementiert wird. Wir haben auch eine Auswahl von Regeln für jeden Algorithmus mit

logarithmischer Laufzeitkomplexität zusammengestellt, die Entwickler als Faustregel verwenden können.

- *Charakterisierung von Software-Ressourcen-Nutzungsprofilen*
  Der dritte Beitrag dieser Arbeit ist die automatische Klassifizierung von Software anhand ihres Ressourcennutzungsprofils, das aus den freigegebenen Leistungsereignissen während der Ausführung gewonnen wird. Das Extrahieren eines Ressourcennutzungsprofils ermöglicht die automatische Auswahl eines geeigneten und genaueren Leistungsmodells. Die Ressourcennutzungsprofile wurden auf einem normalen x86-Server mit über 100 Benchmarks und verschiedenen Konfigurationsoptionen trainiert. Mit denselben Benchmarks wurden zwei verschiedene Leistungsmodelle trainiert, um zu zeigen, dass unser Ansatz ein geeignetes Leistungsmodell auswählen kann, um die Genauigkeit zu erhöhen. Unsere Auswertung zeigt, dass CPU-intensive Workloads in einem ersten Datenexplorationsschritt gut repräsentiert sind, während Speicher- und I/O-intensive Workloads spärlich vertreten sind, was die Klassifizierung erschwert. Dennoch konnten wir eine durchschnittliche Genauigkeit von über 80% erreichen. Wir konnten auch einen Anstieg der Genauigkeit von $12,7\%$ auf $9,7\%$ verzeichnen, selbst wenn wir nur zwei verschiedene Leistungsmodelle verwenden. Weder die Klassifizierung noch die beiden Leistungsmodelle wurden mit der in unserer Auswertung verwendeten Anwendung trainiert. Dieser Beitrag ermöglicht es Entwicklern, unbekannte Anwendungen zu überwachen und sogar genaue Stromverbrauchsmodelle für das aktuelle Ressourcennutzungsprofil während ihrer Ausführung auszuwählen.

- *Feingranulares Leistungsmodell für energieeffiziente Entwicklung*
  Wir schlagen ein neues, feinkörniges Energiemodell vor, das Entwicklern hilft, die Teile ihrer Anwendung zu identifizieren, die am meisten Energie verbrauchen. Unser Ansatz verwendet einen Stack-Trace, der die Funktion oben auf dem Stack bestimmt und ausgelöste Leistungsereignisse dieser Funktion für die Zeit, die sie an obester Stelle auf dem Stack bleibt, zuordnet. Wir haben außerdem einen gleitenden durchschnittlichen Korrekturfaktor eingeführt, um die Anzahl der falsch zugeordneten Leistungsereignisse zu reduzieren. Die Bewertung unseres Ansatzes zeigt, dass wir nach dem Training eines linearen Regressionsmodells für die Anwendung zwischen dem Overhead des Software-Stacks und der Anwendung unterscheiden und die Funktionen mit dem höchsten Energieverbrauch identifizieren können. Wir haben unseren Ansatz unter

verschiedenen Lastniveaus evaluiert, die ähnliche Ergebnisse zeigten, wobei die gleichen Funktionen in der gleichen Reihenfolge für die verschiedenen Lastniveaus identifiziert wurden. Mit diesem Ansatz können Entwickler feststellen, welche Teile ihrer Anwendung für den höchsten Energieverbrauch verantwortlich sind und bei der Optimierung der Energieeffizienz gezielt eingesetzt werden sollten.

- *Leistungsmodell für Speicherintensive Anwendungen*
  Wir definieren ein neues Stromverbrauchsmodell für speicherintensive Anwendungen mit hoher Bandbreitenanforderung. Das Modell basiert auf Leistungsereignissen, die in der CPU aufgezeichnet werden, und unterteilt den Gesamtsystemstromverbrauch in drei Teile: Leerlauf, CPU- und Speicherstromverbrauch. Wir bewerten das Modell mit einem Standard-Benchmark und verschiedenen Datentypen. Die Ergebnisse zeigen, dass wir den Stromverbrauch verschiedener Implementierungen relativ vergleichen können, ohne die zugrunde liegende Hardware zu kennen, oder den absoluten Stromverbrauch des Speichers angeben können, wenn die technischen Details der zugrunde liegenden Hardware bekannt sind.

- *PET-Verbesserungen für eine bessere Leistungsemulation*
  Der letzte Beitrag verbessert unser Performance Event Trigger Framework (PET) mit neuen Performance Event Trigger-Implementierungen und einem automatischen Konfigurationsansatz zur Kompensation von Nebeneffekten während der Ausführung. Wir haben eine verbesserte Funktion zur Auslösung von Leistungsereignissen evaluiert, die es PET ermöglicht, Anwendungen zu emulieren, die Leistungsereignisse mit hoher Frequenz verursachen. Da der synthetische Code unerwünschte Leistungsereignisse verursachen kann, müssen diese vernachlässigt werden, was die Genauigkeit verringert, oder sie müssen kompensiert werden. In diesem Beitrag zeigen wir, dass eine automatische Kompensation von Seiteneffekten die Genauigkeit von PET verbessern kann. Darüber hinaus haben wir PET auf dem ursprünglichen Server und zwei weiteren Servern unterschiedlicher Generationen evaluiert.

Die in dieser Arbeit vorgestellten Beiträge können demonstrieren und machen darauf aufmerksam, dass Energieeffizienz durch effiziente Hardware erreicht wird und durch die Anwendung erheblich beeinflusst werden kann. Die Verbesserung der Energieeffizienz beginnt bei der Auswahl der richtigen Algorithmen und reicht bis zu den optimalen Compiler-Einstellungen. Diese Erkenntnisse können auf diese Tatsachen aufmerksam machen und auch

dazu dienen, nicht-funktionale Eigenschaften in einer Produktspezifikation zu motivieren. Nach der Entscheidung, Energieeffizienz als Ziel in die Entwicklung aufzunehmen, benötigen die Entwickler die richtigen Werkzeuge und Modelle zur Verbesserung der Energieeffizienz. Unser Ansatz kann daher Anwendungen nach ihrem Ressourcennutzungsprofil klassifizieren, was eine automatische Modellauswahl für genauere Ergebnisse ermöglicht. Indem wir die Software vor dem Einsatz emulieren, können wir Entwicklern zeigen, wie sich ihre Anwendung auf der Produktionshardware verhalten wird, oder Betreibern helfen, eine fundierte Entscheidung zu treffen, auf welcher Hardware eine Anwendung eingesetzt werden sollte. Schließlich gibt unser letzter Beitrag Entwicklern Informationen darüber, wo in ihrer Anwendung Energie verbraucht wird und auf welche Teile sie sich konzentrieren sollten, wenn sie die Energieeffizienz verbessern wollen.

Contents

Publication List

Peer Reviewed Conference Full Papers

N. Schmitt, S. Kamthania, N. Rawtani, L. Mendoza, K.-D. Lange, and S. Kounev. "Energy-Efficiency Comparison of Common Sorting Algorithms". In: *Proceedings of the 29th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. MASCOTS '21. New York, NY, USA, 2021.

N. Schmitt, J. Bucek, J. Beckett, A. Cragin, K.-D. Lange, and S. Kounev. "Performance, Power, and Energy-Efficiency Impact Analysis of Compiler Optimizations on the SPEC CPU 2017 Benchmark Suite". In: *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. Los Alamitos, CA, USA: IEEE Computer Society, 2020, pp. 292–301. DOI: 10.1109/UCC48980.2020.00047. URL: https://doi.ieeecomputersociety.org/10.1109/UCC48980.2020.00047.

N. Schmitt, L. Iffländer, A. Bauer, and S. Kounev. "Online Power Consumption Estimation for Functions in Cloud Applications". In: *2019 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE. 2019, pp. 63–72.

A. Bauer, M. Züfle, J. Grohmann, **N. Schmitt**, N. Herbst, and S. Kounev. "An Automated Forecasting Framework based on Method Recommendation for Seasonal Time Series". In: *Proceedings of the 11th ACM/SPEC International Conference on Performance Engineering (ICPE 2020)*. Edmonton, Canada: ACM, 2020.

J. von Kistowski, J. Grohmann, **N. Schmitt**, and S. Kounev. "Predicting Server Power Consumption from Standard Rating Results". In: *Proceedings of the 19th ACM/SPEC International Conference on Performance Engineering*. ICPE '19. Full Paper Acceptance Rate: 18.6% (13/70). New York, NY, USA: Association for Computing Machinery (ACM), 2019, pp. 301–312. URL: https://doi.org/10.1145/3297663.3310298.

J. von Kistowski, S. Eismann, **N. Schmitt**, A. Bauer, J. Grohmann, and S. Kounev. "TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research". In: *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE. 2018, pp. 223–236.

Peer Reviewed Journals

E. Van Eyk, J. Grohmann, S. Eismann, A. Bauer, L. Versluis, L. Toader, **N. Schmitt**, N. Herbst, C. Abad, and A. Iosup. "The SPEC-RG Reference Architecture for FaaS: From Microservices and Containers to Serverless Platforms". In: *IEEE Internet Computing* (2019).

Peer Reviewed Conference Short and Work-In-Progress Papers

N. Schmitt, K.-D. Lange, S. Sharma, N. Rawtani, C. Ponder, and S. Kounev. "The SPECpowerNext Benchmark Suite, Its Implementation and New Workloads from a Developers Perspective". In: *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. ICPE '21. Virtual Event, France: Association for Computing Machinery, 2021, pp. 225–232. ISBN: 9781450381949. DOI: 10.1145/3427921.3450239. URL: https://doi.org/10.1145/3427921.3450239.

N. Schmitt, J. Bucek, K.-D. Lange, and S. Kounev. "Energy Efficiency Analysis of Compiler Optimizations on the SPEC CPU 2017 Benchmark Suite". In: *Proceedings of the 11th ACM/SPEC International Conference on Performance Engineering (ICPE 2020)*. New York, NY, USA: ACM, 2020. URL: https://doi.org/10.1145/3375555.3383759.

N. Schmitt, J. von Kistowski, and S. Kounev. "Emulating the Power Consumption Behavior of Server Workloads using CPU Performance Counters". In: *Proceedings of the 25th IEEE International Symposium on the Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*. MASCOTS '17. 2017.

N. Schmitt, J. von Kistowski, and S. Kounev. "Predicting Power Consumption of High-Memory-Bandwidth Workloads". In: *Proceedings of the 8th ACM/SPEC*

International Conference on Performance Engineering. ICPE '17. New York, NY, USA: ACM, 2017, pp. 353–356. URL: http://doi.acm.org/10.1145/3030207.3030241.

Peer Reviewed Workshops, Vision, Tutorial, Poster, and Demonstration Papers

N. Schmitt, R. Vobl, A. Brunnert, and S. Kounev. "Towards a Benchmark for Software Resource Efficiency". In: *Companion of the ACM/SPEC International Conference on Performance Engineering*. ICPE '21. Virtual Event, France: Association for Computing Machinery, 2021, pp. 179–182. ISBN: 9781450383318. DOI: 10.1145/3447545.3451176. URL: https://doi.org/10.1145/3447545.3451176.

N. Schmitt, L. Iffländer, A. Bauer, and S. Kounev. "Online Power Consumption Estimation for Functions in Cloud Applications (Poster)". In: *2019 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE. 2019.

N. Schmitt, J. von Kistowski, and S. Kounev. "Towards a Scalability and Energy Efficiency Benchmark for VNF". in: *Proceedings of the 9th TPC Technology Conference on Performance Engineering and Benchmarking*. TPCTC '17. 2017.

K. Tocze, **N. Schmitt**, I. Brandic, A. Aral, and S. Nadjm-Tehrani. "Towards Edge Benchmarking: A Methodology for Characterizing Edge Workloads". In: *Proceedings of Workshop on Hot Topics in Cloud Computing Performance (HotCloudPerf) as part of FAS*(IEEE ICAC/SASO) conferences companion*. IEEE, 2019.

S. Eismann, J. Kistowski, J. Grohmann, A. Bauer, **N. Schmitt**, and S. Kounev. "TeaStore - A Micro-Service Reference Application (Demo)". In: *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. 2019, pp. 263–264. DOI: 10.1109/FAS-W.2019.00073.

S. Eismann, J. v. Kistowski, J. Grohmann, A. Bauer, **N. Schmitt**, N. Herbst, and S. Kounev. "TeaStore: A Micro-Service Reference Application for Cloud Researchers (Poster)". In: *Proceedings of 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. 2018, pp. 11–12.

Book Chapters

N. Schmitt. "Improving the Energy Efficiency of IoT-Systems and its Software". In: *Organic Computing: Doctoral Dissertation Colloquium 2018*. Ed. by S. Tomforde and B. Sick. kassel university press GmbH, 2019.

Peer Reviewed Software Artifacts and Datasets

N. Schmitt, J. Bucek, J. Beckett, A. Cragin, K.-D. Lange, and S. Kounev. *SPEC CPU 2017 Benchmark Suite Results for the HPE ProLiant DL385 Gen10 server*. Zenodo, 2020. DOI: 10.5281/zenodo.3840977. URL: https://doi.org/10.5281/zenodo.3840977.

J. von Kistowski, S. Eismann, **N. Schmitt**, A. Bauer, J. Grohmann, L. Bui, and S. Kounev. *TeaStore*. Standard Performance Evaluation Corporation (SPEC) Research Group. Accepted tool. 2019. URL: https://research.spec.org/tools/overview/teastore.html.

Chapter 1

Introduction

In the introduction of this doctoral thesis, we first set the context of this research and motivate it. We then describe the state-of-the-art on this research topic and assess shortcomings in the problem statement. Through this problem statement the goals are defined. We further divide the goals into research questions that are answered in the following chapters to achieve these goals. The contributions of this work and their associated evaluation results are then summarized. We finally give an outline of the remainder of this thesis.

1.1 Motivation

Today's cloud data centers consume an enormous amount of energy. In 2012 the New York Times and Andrae et al. estimated that data centers consume about 30 billion watts of power [Gla12; AE15], resulting in about $263\,TWh$ of energy usage per year. A more recent study by Masenet et al. found that data centers used about $208\,TWh$ annually but stated that existing efficiency could not compensate for a sharp growth in data center usage [Mas+20]. Andrae et al. estimate that the energy consumption will rise to $1929\,TWh$ until 2030 in the expected scenario [And19], a decrease compared to their earlier estimation of $2967\,TWh$ [AE15]. However, a steep increase compared to Masanet et al. Nevertheless, all studies agree that the energy consumption of data centers will rise in the future.

This projected rise in energy demand is fueled by a growing number of services deployed in the cloud. According to Gartner, public cloud spending has increased by 18.7% in 2021 [Gar20] and Flexera states that 50% of enterprise workloads have been migrated to the cloud in the last decade [Fle20]. Additionally, an increasing number of devices are using the cloud to provide functionalities and enable data centers to grow. Statista estimated that by 2025, more than 75 billion IoT devices would be in use [Sta16]. However, it is not the users and devices that consume the energy in the data center but their

service requests that are fulfilled through the use of computing resources like processor, memory, storage, and network.

The growing energy demand also increases the amount of CO_2 emissions. Assuming a CO_2-intensity of $200g$ CO_2 per kWh in a rather optimistic scenario[1], a consumption of $263\,TWh$ will result in about 53 billion tons of CO_2 produced to operate data centers, whereas a consumption of $1137\,TWh$ will get us close to 227 billion tons of CO_2. According to the Umweltbundesamt, this is more than the CO_2 emissions of all energy-producing power plants in Germany in 2020 [Umw21].

It is, therefore, of paramount importance to reduce the energy consumption of cloud data centers. There are three main factors that drive the energy consumption of data centers and need to be considered in order to achieve better energy efficiency:

- The hardware that physically consumes energy during operation.
- The software that controls how the hardware behaves.
- The amount of work, mostly user-generated in the form of incoming requests, processed by the deployed application.

Many technologies exist to increase the energy efficiency of data centers on the hardware level, such as Dynamic Voltage and Frequency Scaling (DVFS), and benchmarks to create an incentive for manufacturers to improve energy efficiency, such as SPECpower_ssj 2008 benchmark [SPE19]. Through public results and the contribution of the industry, server energy efficiency (operations-per-watt server efficiency) has increased by 113 times for servers with a single CPU (from 190 to $21\,603$ over 13 years), or 19 times on average across all benchmarked servers since the release of the SPECpower_ssj 2008 benchmark [SPE19; Sch+21b]. We argue that the software controls the hardware and, therefore, while physically consuming the energy, the hardware is not always responsible for wasting energy. The software itself plays a vital role in reducing the energy consumption and CO_2 emissions of data centers. We also do not want to alter user behavior or set a maximum to the processable work and, therefore, the varying load on a service.

In this thesis, we aim at improving the energy efficiency of data center applications in general and not just at specific loads or for specific hardware. Hence, it would be highly beneficial to know how much potential energy efficiency improvements developers can achieve through changes in the software, as

[1] https://www.electricitymap.org, accessed on September 30th, 2020

well as which parts of the applications' source code are responsible for most energy consumption and how we can emulate the application behavior before deployment.

1.2 Problem Statement

In this thesis, we focus on approaches for software engineering of energy-efficient distributed systems. Often the term *green* or *sustainable* software engineering or development are used [CP15b]. Johann et al. define this, as the consumption of natural resources for the development, deployment, and utilization, of software [Joh+11]. However, this definition includes natural resources used during development, such as fuel used to attend a meeting for a business process. In contrast to this, we focus strictly on the energy consumption caused by the execution of the software itself. We do not include anything besides the energy an application uses during operation and optimize this energy usage during the development phase.

Nevertheless, we must first show developers that software contributes to energy consumption by providing evidence of its influence. The second step is to provide methods to assess an application's power consumption during different phases of the software lifecycle in modern DevOps and agile development proceses. To this end, methods for automatic extraction of system-level energy-consumption models that can accommodate rapid changes in the source code are needed coupled with application-level models allowing the developer to locate power-consuming software parts and consider them as target for optimization. Finally, approaches to emulate the software energy consumption are needed allowing developers to assess the energy efficiency before the actual deployment.

Reiterating the above, first, we want to raise the awareness among developers that the software design and implementation can have significant impact on the energy efficiency. In a 2016 study by Pang et al. [Pan+16] about software energy consumption, out of 122 programmers, only 18% answered that they take energy consumption into account when programming. More positively, 14% considered minimizing energy consumption as a requirement, and 21% responded they had already changed software to consume less energy [Pan+16]. This survey shows that optimizing energy efficiency is not widely considered and adopted even if it does not mean trading energy consumption for performance as Capra et al. showed by testing two different Enterprise Resource Planning (ERP) systems. While both systems exhibited similar performance differing only by about 5%, their energy efficiency varied up to 50% [CFS12].

Pinto et al. argue that energy consumption is becoming a critical aspect of software engineering. However, there is a lack of knowledge and experience on how to optimize software energy efficiency [PC17]. Instead of making decisions based on instinct and habits, developers should make design and coding choices with care. To make developers aware that software is responsible for its energy consumption by controlling the hardware, we must first show the possible range of energy savings or improvements in energy efficiency that can be achieved in software by providing evidence.

As a next step, developers need tools and methods to improve and optimize the software efficiency. A software's energy consumption can be reduced by intelligently placing or consolidating deployed software components [TPV17; JWC12] or using only the minimal amount of resources to satisfy user demands through autoscaling [Her18]. With new emerging paradigms, such as serverless computing, individual functions of an application are deployed rather than an entire application stack. With the move towards fine-grained deployments, away from classical component-based multi-tier applications, hierarchical energy-saving techniques [LPF10] become less effective. Code offloading (i.e., moving computations to the cloud to conserve battery power) also leads to deploying only parts of an application [Flo+15]. The fine-granular control of the deployment limits existing power models that generally do not capture the power consumption in such detail. To help developers, new more accurate power models for fine-grained monitoring, allowing developers to easily identify energy-efficient code, are necessary.

Identifiying the energy consuming parts of an application can help if the application's source code does not change in quick succession. With the rise and increasing adoption of the DevOps paradigm, code changes due to rapid changes in requirements [Bru+15]. These code changes also impact power consumption and consequentially impact the optimal deployment [Che+20; Kan+14]. Modeling and monitoring a deployed application using application-level metrics under continuous changes can become cumbersome as it would require constant adaptations. Hence, instead of using application-level metrics, we aim to automatically extract a suitable power model for a given application based on system-level metrics. To this end, an automated workload classification selecting the most suitable power model is needed.

Modeling a software system's power consumption is usually done either by training a block-box model through measurements [SMM07; IM03; BJ12] or by defining a workload profile from user input combined with expert knowledge about the hardware [Hao+13]. Such approaches have in common that they are either too coarse-grained to aid developers in guiding towards designing

energy-efficient applications or they rely on hardware information that is often unavailable at development time. The automated deployment in a DevOps environment can also deploy an application on a large variety of hardware platforms such that it becomes infeasible to retrain models for all possible servers. Hence, we investigate the reusability of power models without retraining them for a specific server or application.

In addition to modeling, developers can also emulate the execution of an application to obtain further information about its power consumption. The software developers or an automated deployment process must know how an application would behave in order to select the optimal and most energy-efficient hardware for the application and save energy. This task is problematic as servers are working under highly variable load intensities [KHK14] and are generally not most efficient under full load [Kis+15b]. To select a reasonably efficient application server, Kistowski created an energy efficiency rating for servers [Kis19]. A tool called Server Efficiency Rating Tool (SERT) uses this rating to measure server energy efficiency under different loads stressing CPU, memory, and I/O [SPE13]. Nevertheless, its workloads cannot accurately represent every possible software and do not include network communication. We, therefore, developed a Performance Event Trigger Framework (PET) to emulate network workloads with an accuracy of about 10% [SKK17a; Kis19]. The emulation of an application allows developers to identify the most energy-efficient server before deployment and operation. PET also simplifies the testbed, removing the need to resolve dependencies on additional servers or services. Additionally, developers can use PET to estimate the energy consumption of an application that has not been developed yet under the assumption that the developer knows or can estimate the resource usage profile in advance. One of the main drawbacks of PET is the complicated configuration to account for side effects of the emulation, that is, the indirect triggering of unwanted but unavoidable performance events [SKK17a]. As part of this thesis, we improve PETs emulation and configuration to achieve a better selection of energy-efficient servers for a given application.

To summarize, this thesis addresses four main challenges:

1. Developers lack awareness that they can significantly influence the energy efficiency of software.

2. Fine-granular power models are necessary to help developers identify energy-consuming parts of the application.

3. Constantly and rapidly changing applications necessitate an automated approach to select the best power model.

4. Automated and frequent deployments in a DevOps context need an improved emulation of an application's power consumption.

1.3 Goals and Research Questions

To address the above described challenges, we formulate two main goals of this thesis. Our first goal is to raise awareness among developers about the influence of software on energy efficiency, addressing the first challenge. The second goal focuses on giving developers the knowledge and methods to improve energy efficiency, addressing the remaining three challenges. We further break down the goals into specific research questions that we answer in this thesis to achieve these goals.

Goal A The first goal is to make developers aware of the extend to which the software design and configuration may influence its energy efficiency. In other words, the goal is to investigate the implications on development aspects on the energy efficiency of an application. We conducted two case studies to show the possible ranges in energy efficiency when the compiler optimizations and algorithms are selected with care.

RQ A.1 *How large is the influence of compiler optimizations on the power consumption and energy efficiency of applications?*

RQ A.2 *How large is the influence of sorting algorithms (as an example of commonly used algorithms) on the power consumption and energy efficiency of applications?*

After developers are aware of the possibilities to increase energy efficiency, they must have the knowledge of how and where energy in an application is spent and guide developers to the relevant pieces of code. Hence, more find-grained and accurate power models for predicting and monitoring the energy-efficiency are necessary. Developers also need an automated process to select power models to accommodate rapid code changes in addition to accurate emulation.

Goal B The second goal is to support developers in improving the energy efficiency of applications based on monitoring data both at the system level and the application level. First, an application should be classified according to its resource usage profile on the system level, enabling the generation of a suitable power model based on general system-level metrics. To give more detailed insights, a second power model based on

application-level metrics is also necessary. Fine-granular power models on the application level allow developers to identify the most power consuming parts of their application. The emulation, on the other hand, allows developers to predict how the application will behave on production hardware without having to resolve dependencies.

RQ B.1 *How to classify applications in an automated manner according to their resource usage profile in order to derive suitable and, therefore, more accurate power models?*

RQ B.2 *How to model high-bandwidth memory applications with system-level metrics?*

RQ B.3 *How to determine which function in an application consumes which amount of power and energy? The answer to this question can aid developers in improving the energy-efficiency of the application.*

RQ B.4 *Can power models be reused for different servers and applications? This question is twofold. First, can we transfer a functional power model across different servers; second, can a machine-learning based classification for power models be transferred across applications without retraining?*

RQ B.5 *How can the behavior of an application emulation on production servers be improved to predict its power and energy consumption more accurately and possibly improve the energy efficiency at deployment?*

Achieving *Goal B* allows developers to adapt their code towards higher energy efficiency and select suitable hardware for deployment. This improvement can help counteract the constant growth of cloud data centers and, thus, reduce CO_2 emission.

1.4 Thesis Contributions

This thesis makes six contributions summarized below. The first two contributions focus on achieving *Goal A* while the remaining contributions are aimed at *Goal B*. The contributions address the research questions formulated for the specific goal.

Contribution 1 *Case Study Demonstrating the Impact of Compiler Optimizations*
This contribution addresses *Goal A* and more specifically research question *RQ A.1* by presenting a case study on the influence of compiler optimizations on power consumption and energy efficiency. We performed the case study with the SPEC CPU 2017 benchmark suite from the

Standard Performance Evaluation Corporation (SPEC), which provides time-to-result and throughput metrics for a wide variety of software artifacts covering different application domains, lines of code, and programming languages. We ran the benchmarks in two scenarios, a real-world scenario with common compiler optimizations and an optimized scenario where experts have tuned compiler settings to achieve the best possible benchmark score.

The results are evaluated regarding their performance, power consumption, energy efficiency, and whether a C-like or a functional programming language can be optimized better. We show that an increase of about 30% in energy efficiency is possible for most benchmarks, leaving ample room for improvements that need no change in source code. C-like languages respond better to compiler optimizations to increase energy efficiency. We also investigated the power consumption over time, showing a reduction in variability, which could help ease the predictions and models for power budgeting in the future. We made the measurement results public [Sch+20b] and published the contributions in the Proceedings of the 11th ACM/SPEC International Conference on Performance Engineering (ICPE) 2020 [Sch+20c] and the 13th IEEE/ACM International Conference on Utility and Cloud Computing (UCC) 2020 [Sch+20a].

Contribution 2 *Case Study Demonstrating the Impact of Sorting Algorithms*
The second main contribution also targets *Goal A*, also focussing on answering research question *RQ A.2*. Measurements are taken on four Systems under Test (SUT). For each sorting algorithm, two variants in different programming languages and three problem sizes were used and run on two different SUTs, resulting in a total of 144 measurements. This contribution shows the large impact a simple algorithmic choice, such as choosing a sorting algorithm can have on energy efficiency.

We evaluate the results of the case study according to their energy efficiency in bytes sorted per watt-second. The results show that algorithms with the same average time complexity of $n * \log(n)$ can vary widely in energy efficiency, and developers should select algorithms with care. Additionally, an algorithm must be implemented with care. Quick Sort experiences a drop of over $84MB/Ws$ when implemented recursively instead of iteratively. We also compiled a selection of rules for each algorithm with logarithmic runtime complexity that practitioners can use as a rule-of-thumb. We published this contribution in the Proceedings of the 29th IEEE International Symposium on the Modeling, Analysis, and

Simulation of Computer and Telecommunication Systems (MASCOTS) 2021 [Sch+21a].

Contribution 3 *Classifying Software Resource Usage Profiles*

The third contribution of this thesis is the automatic classification of an application according to its resource usage profile in terms of the released performance events during execution. Extracting a resource usage profile allows one to automatically choose an appropriate and more accurate power model, which addresses research question *RQ B.1* of *Goal B*. In addition, we developed a machine learning power model to test if such a model can be transferred across applications without retraining, partially answering research question *RQ B.4*. The resource usage profiles have been trained on a regular x86 server as SUT with over 100 benchmarks and different configuration options. The same benchmarks train two different power models to show that our approach can select a suitable power model to increase accuracy.

We evaluate XGBoost, Random Forest, and logistic regression with automatic feature selection to compare the workload classification based on different machine learning approaches. Our evaluation shows that CPU-intensive workloads are well represented in a first data exploration step, while memory-, and I/O-intensive workloads are sparse, making the classification difficult. Still, we could achieve an average accuracy of over 80% by XGBoost and Random Forest. Our approach shows an increase in accuracy from 12.7% to 9.7%, even if we only use two distinct power models. Neither the classification nor the two power models were trained with the application used in our evaluation. This contribution allows developers to monitor unknown applications and automatically select accurate power models for the current resource usage profile during its execution.

Contribution 4 *Fine-Granular Power Model for Energy-Efficient Development*

We propose a new fine-grained power model to guide developers in identifying the most power-consuming parts of their application. This contribution is aimed at *Goal B* and addresses research question *RQ B.3*. It also addresses the second part of research question *RQ B.4* by checking if such a fine-grained model can be used on a different server machine without retraining. We use three different sized servers as SUTs and the image provider service of the TeaStore [Kis+18] reference workload as a workload stressing CPU by resizing images, memory via caching, and I/O through reading and writing images to disk. Our approach uses a

stack trace determining the function on top of the stack and attributing triggered performance events to this function for the time it stays on top of the stack. We also introduced a sliding average correctional factor to reduce the amount of falsely attributed performance events.

The evaluation of our approach shows that after training a linear regression model for the application, we can distinguish between the overhead of the software stack and the application, which allows us to identify the most power-consuming functions. We evaluated our approach under different load levels, which showed similar results, identifying the same functions in the same order for the different load levels. This approach allows developers to determine which parts of their application are responsible for the highest energy consumption and should be targeted when optimizing energy efficiency. As developers often do not know or have access to the same physical machine their application will be deployed on, it is essential to test whether our developed model is transferable across different hardware. Our evaluation shows that on different SUTs, without retraining the model, the same functions using the same relative amount of power can be identified with only minor deviations of under 1% to 3%. This contribution has been published in the Proceedings of the 16th IEEE International Conference on Autonomic Computing (ICAC) 2019 [Sch+19a].

Contribution 5 *High-Bandwidth Memory Workload Power Model*

Our next contribution addresses research question *RQ B.2* of *Goal B* by defining a new power model for high-bandwidth memory workloads. The model is built upon performance events recorded in the CPU, and it divides the total system power consumption into three parts: idle, CPU, and memory power consumption. We evaluate the model with the STREAM benchmark [McC95] and different data types. The results show that we can compare the memory power consumption of different implementations relatively without assuming knowledge of the underlying hardware, and we can even estimate the absolute memory power consumption if technical details of the underlying hardware are known. We published this contribution in the Proceedings of the 8th ACM/SPEC International Conference on Performance Engineering (ICPE) 2017 [SKK17b].

Contribution 6 *PET Improvements for Better Power Emulation*

The last contribution improves PET [SKK17a; Kis19] with new performance event trigger implementations and an automatic configuration approach to compensate side effects during execution. The contribution

addresses the research question *RQ B.5* of *Goal B*. PET works by executing synthetic code designed to trigger a defined number of performance events. We evaluated an improved performance event trigger function to allow PET to emulate applications that cause performance events at a high frequency. Additionally, as the synthetic code might cause unwanted performance events, they must be neglected, which reduces accuracy, or compensated. In this contribution, we show that an automatic side effect compensation can improve PET's accuracy. In addition, we evaluated PET on the original server and two additional servers of different generations.

The presented contributions of this thesis can provide evidence and raise awareness that energy efficiency is achieved through efficient hardware and can considerably be influenced by the application. Improving energy efficiency starts with a selection of the correct algorithms down to the optimal compiler settings. These findings can raise awareness and can also be used to motivate non-functional properties in a product specification. After deciding to include energy efficiency as a goal during development, developers need the right tools and models to improve energy efficiency. Our approach, therefore, can classify applications according to their resource usage profile enabling an automated model selection for more accurate results. By emulating the software before deployment, we can show developers how their application will behave on the final production hardware if available or help operators make an informed choice on which hardware an application should be deployed. Finally, our last contribution gives developers information on where energy is consumed in their application, and which parts they should focus on when aiming to improve energy efficiency.

1.5 Thesis Outline

This thesis is structured into seven parts, excluding this introduction. Part II contains the necessary foundations and technical background necessary for this thesis in Chapter 2 and the current state-of-the-art in literature in Chapter 3.

Part II of this thesis describes our two contributions towards raising awareness among developers. The contributions in this part address the research questions of *Goal A*. First, we describe in Chapter 4 that software can influence power consumption. Then, we analyze the influence of compiler optimizations on the power consumption in Chapter 5 and the influence of algorithm selection in Chapter 6.

The Parts III, IV, and V contain our contributions towards achieving *Goal B*. In Part III, we address the classification of applications according to their resource

profile to select more suitable power models. They are followed by Part IV, presenting two novel power models, one for high-bandwidth applications and one for attributing power consumption to functions. This part also addresses the issue of reusing power models without retraining for a specific application or hardware.

Then, in Part V, we identify and propose possible improvements for emulating power consumption. Additionally, we do propose a new configuration approach that makes power consumption emulation less cumbersome to use. We evaluate our power models, transferability, and emulation in Part VI, answering the research questions of our *Goal B*.

Finally, we conclude this thesis in Part VII. This part also presents the benefits of this thesis and possible future work.

Disclaimer

This thesis contains measurement results obtained with the following SPEC benchmarks and tools: SPECpower_ssj® 2008, SPEC SERT® Suite, SPEC SERT® 2.0.4 Suite, Chauffeur® Worklet Development Kit (WDK), and SPEC CPU® 2017. Unless noted otherwise, these results have been obtained according to the SPEC fairuse policy for academic use [SPE21b] and must be considered non-compliant.

Part I

Foundations and Related Work

Chapter 2

Context and Foundations

This chapter describes the context cloud computing, DevOps, and essential foundation for this thesis. We first summarize and define how we understand cloud computing in this thesis as all of our contributions are primarily aimed at software deployed in any kind of cloud scenario. We then describe performance events and performance counters as they are extensively used in our contributions and evaluations. This section is followed by the general methodology on power and energy efficiency measurements under different load levels and the PET for emulation. We then describe the benchmarks and workloads that we used during measurements. Finally, we shortly describe the machine learning approaches applied in the course of this thesis.

2.1 Cloud Computing and DevOps

The contributions of our thesis aim at cloud computing. We, therefore, define what cloud computing is in the context of this thesis. Afterwards we give a short description of DevOps.

2.1.1 Cloud Computing

This section is based on the National Institute of Standards and Technology (NIST) definition of cloud computing [MG11] and the Standard Performance Evaluation Corporation, Research Group (SPEC RG) Function as a Service (FaaS) reference architecture [Van+19]. Given the NIST definition, cloud computing is moving computations, services, and data to data centers. According to NIST, cloud computing has five characteristrics that are extended by energy efficiency by Kounev et al.[Kou+17] to six main characteristics.

1. *Elasticity*: Resources can be added and removed to the customers pool on-demand.

2. *Network Access*: Every device with internet connectivity can access the resources.
3. *Resource Pooling*: Data can be added at any given time.
4. *Measured Services*: Customers pay only for the resources used.
5. *Virtualization*: Infrastructure can be divided and seen as multiple logical components.
6. *Energy Efficiency*: Consumption can be optimized.

While energy efficiency seems the most interesting characteristic, it refers to using software to achieve energy efficiency through optimized placement [TPV17; JWC12], auto-scaling [Her18] rather than optimizing the software itself. The operator must manage a cloud data center to leverage these characteristics. Therefore, the data center operators perform management at different abstraction levels for each service model listed below.

Infrastructure as a Service (IaaS) IaaS provides computational and storage resources to the cloud computing customer. This service model allows the customer to run arbitrary applications and operating systems on the cloud infrastructure. One example for IaaS is Amazon EC2. The provider manages the infrastructure on which virtual machines are running that are added as resources the users pool.

Platform as a Service (PaaS) PaaS provides services, libraries, programming languages, and other tools for developers to design and run their own application. An example for PaaS is Google App Engine. The provider manages the infrastructure and virtualization for the customer.

Function as a Service (FaaS) FaaS provides tools to deploy only parts of an application, the functions, that the developer can combine into an application. Each function is managed by the provider independently. One example is Microsoft Azure Functions.

Software as a Service (SaaS) SaaS provides a ready to use application for customers, for example Google Docs. Management of the software is done completely by the provider.

Next to the service levels, there exist several types of deployment types for cloud computing data centers. We give a short overview of the four general deployment types, according to the NIST definition.

Private Cloud A private cloud is exlusive to the owner. This can be a single corporation where access to the cloud is restricted to its own business units.

Community Cloud A community cloud is exclusive to a closed group of users coming from several corporations with shared interests.

Public Cloud A public cloud is open to the general public.

Hybrid Cloud A hybrid cloud is a combination of two or three clouds of the above deployment types. The two components must not be in the same location but are connected.

We designed the contributions in this thesis to be usable in all deployment types of cloud computing described above.

2.1.2 DevOps

The term DevOps describes the convergence of development (*Dev*) and operations (*Ops*) [Bru+15]. In a study of Jabbari et al. [Jab+16] they define it as follows: "DevOps is a development methodology aimed at bridging the gap between Development (Dev) and Operations, emphasizing communication and collaboration, continuous integration, quality assurance and delivery with automated deployment utilizing a set of development practices."

DevOps, therefore, allows faster development cycles and releases. The developers can release a new version of an application within minutes when they use an automated Continuous Integration / Continuous Deployment (CI/CD) pipeline [Bru+15]. DevOps is one part towards *cloud-native* application development [Red18].

2.2 Performance Monitoring Events and Counters

Most modern processors architectures from AMD, Intel, IBM, and ARM integrate performance counters [Adv21; Int17; IBM18; Arm21]. They are Model Specific Registers (MSRs) in the processors Performance Monitoring Unit (PMU) that measure performance monitoring events, and that can be read by either specialized software like LIKWID [HWT10], through the operating system (for example, the `perf` API in Linux), or manually by directly accessing the performance counter registers. The counted performance events are typically used to identify performance bottlenecks and can be divided into two groups.

Occurrence events An occurrence event is an event happening at a single point in time. A performance counter register will be incremented if an occurrence event appears. Hence, the register holds the number of observed occurrence events of a given type. Examples for occurrence events are cache misses and retired instructions [Int17].

Duration events A duration event is happening over a period of time. A performance counter register will count the duration the processor was in a state that caused the duration event to appear. An example for a duration event is recovery cycles, which is the number of core cycles the resource allocator was stalled due to a branch misprediction or machine clear event [Int17].

Performance counters on Intel platforms can either be architectural or non-architectural events. Architectural events have the benefit of behaving consistently across microarchitectures of the processor's manufacturer [Int16]. However, they are typically not as expressive, and there are not as many non-architectural events to choose from. AMD divides its performance events into core events and northbridge events [Adv21]. Additionally, not all architectures provide identical performance events or the same name for similar performance events. These differences make it difficult to transfer models and methods based on performance events across architectures without expert knowledge. Yet, for x86-based architectures, many similarities exist.

Using the PMU to record performance events has its drawbacks. Performance counter can count more events than actually occurred by incrementing more than once for a single event, or they experience non-determinism, and the count differs even under the identical workload [WTM13]. Since then, processor manufacturers have introduced high accuracy performance counters to remedy these issues (for example Intel Performance Event Based Sampling (PEBS) [Int17]). For this thesis, if not otherwise stated, we refer to the accurate performance counters and assume that the counted events are indeed accurate.

Next to the processor themselves, operating systems also count events. While not technical performance monitoring events, they can be similar and benefit, like architectural events, from consistent behavior across different processors. We consider events counted by the operating system, like context switches, for this thesis as performance events.

We generally measure performance events on two different levels, the *system level*, and the *application level* and define the abstraction levels as follows.

System level We define performance events and other measures at the system level as measures that reflect the state of the overall server or different

hardware subsystems like CPU or memory. We do not take into account any form of virtualization that might be present on the server. Examples for system level measures are the total number of instructions retired on a CPU.

Application level Meaures that we take on the application level are attributable to a software running on the system. These measures can also be performance counters. The software can but must not necessarily be deployed in a virtualized environment. We also count the application stack towards application level metrics, for example the Apache Tomcat[1] that we use for the TeaStore reference application.

2.3 Power and Energy Efficiency

Power and energy efficiency measurements differ from performance measurements. They are often measured under different load levels and not at maximum throughput or time-to-result. Measuring under different load levels is due to the fact that servers in data centers are often working at 30% to 50% load [BH07]. The Standard Performance Evaluation Corporation (SPEC) developed a benchmark and tool to measure and rate the energy efficiency of servers under varying load, namely the SPECpower_ssj 2008 benchmark and the SERT suite. Hence, this thesis follows the SPEC power and performance benchmarking methodology [KLK20; SPE14] where suitable. In this chapter, we first describe the general measurement setup for power and energy efficiency followed by a description of PET to emulate the power consumption of an application.

2.3.1 Measurement

Servers typically are not equipped with power measurement devices but use performance events to model the energy consumption [Rot+12], known as Running Average Power Limit (RAPL). However, RAPL counters are primarily not implemented for power measurements but thermal management, power limiting, and power and performance budgeting [Int16]. Additionally, RAPL counters, while having a high sampling rate of 1 kHz, experience jitter [Häh+12] and have no timestamp assigned to it. The RAPL counters actual implementation can also change between different processors, reducing comparability. We, therefore, use an external power analyzer that measures the wall power

[1] `http://tomcat.apache.org/`

of the SUT as depicted in Figure 2.1 for the SPEC CPU 2017 benchmark suite example. These accurate power measurements act as the ground truth for our evaluations instead of the RAPL counters.

The power analyzers we used in this thesis are connected to the SUTs Power Supply Unit (PSU) and have an internal sampling rate of approximately 10 kHz. All samples are averaged over one second. As the temperature can have a significant impact on the power consumption, an additional temperature sensor can be used to measure the air intake temperature of the SUT to increase the repeatability of a benchmark run or experiment. We do not use the temperature sensor in this thesis because all our experiments were performed in a controlled environment and are not official benchmark results. We also assume that the reader is familiar with basic power and energy measures and units used throughout this thesis in the form of power consumption in *Watt* $[W]$, energy consumption in *Joule* $[J]$ or *Wattsecond* $[Ws]$, voltage in *Volt* $[V]$, and current in *Ampere* $[A]$.

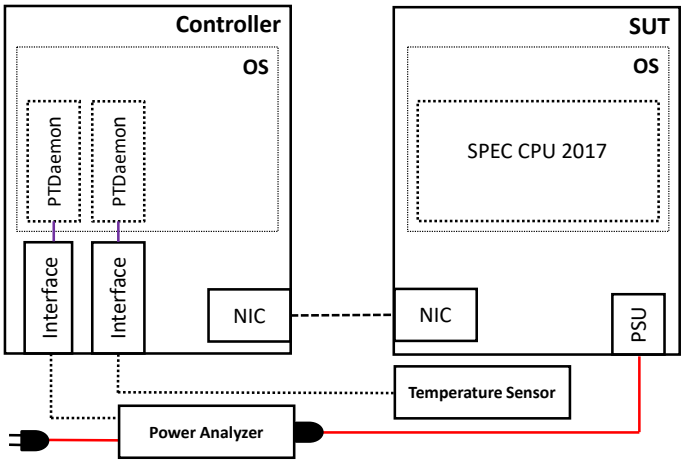


Figure 2.1: General setup for power measurements on the example of the SPEC CPU 2017 benchmark suite.

As mentioned earlier, energy efficiency is mainly measured under different load levels as depicted in Figure 2.2. We define the maximum load level as the amount of work done by a service that the combination of hardware and software can handle without error. This definition of the load must not necessarily but can correlate to CPU, memory, or I/O utilization. The SERT suite and Chauffeur WDK define the load level through the maximum number of transactions a system can handle. They further specify a transaction as a

portion of work blocking computational resources until it is finished. After a transaction is finished, the Controller dispatches the next one. The number of transactions that the Controller can dispatch without delay between them in a defined timeframe is then specified as 100% load. Calibrating the maximum load is typically done once or more, as shown in Figure 2.2 with two calibration runs. The SERT suite and Chauffeur WDK use fractions of the maximum load to calculate the number of transactions or requests for lower load levels. If they measure the SUT under lower load levels, both space the transactions with a delay drawn from a negative exponential distribution. Each measurement also has a pre-measurement and post-measurement phase to allow the system to settle and measure in a steady state. Measuring in a steady-state improves the ability to compute statistical measures from the data [KLK20].

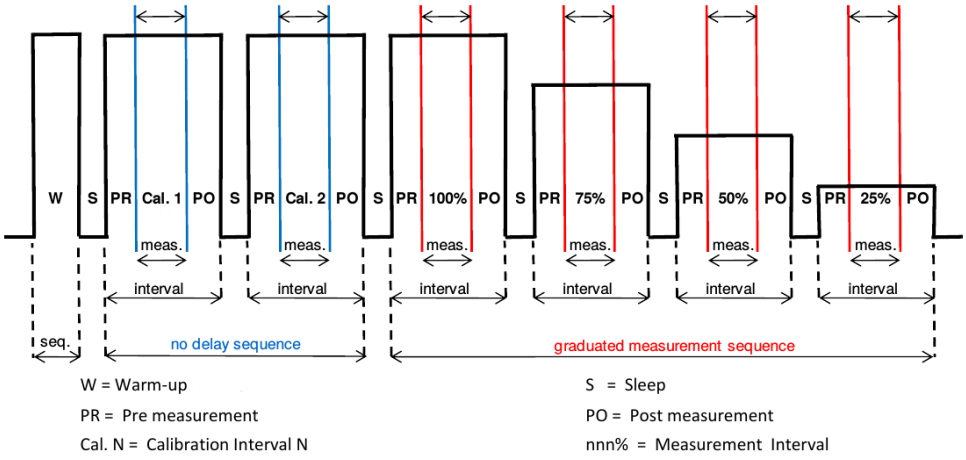


Figure 2.2: SPEC Chauffeur WDK measurement example with calibration and different load levels. [SPE13]

2.3.2 Emulation

PET has been developed to emulate the power consumption of software, specifically networked software with a complicated test environment. This section is based on publications of Schmitt et al. [SKK17a] and Kistowski [Kis19]. To achieve this emulation, PET relies on the assumption that the identical amount of performance events will cause an identical or similar power consumption. This assumption is based on the RAPL counters modeling the energy consumption [Int16]. By removing load drivers and other parts of the environment,

shown in Figure 2.3, that is not under test allows a PET user to simplify their test setup for quicker measurements of complicated networked server software. The only external devices necessary are a control system that governs a power analyzer and a controller that manages the measurement as shown in Figure 2.4. This simplification also brings it closer to the general setup of other benchmarks and workloads used in this thesis, with an example setup shown in Figure 2.1, and for PET specifically in Figure 2.4.

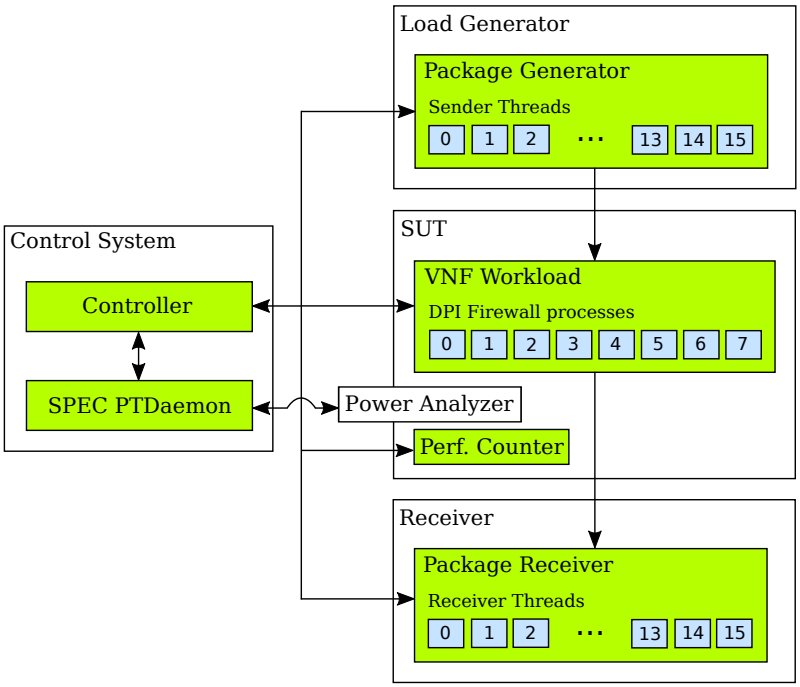


Figure 2.3: PET reference testbed with load driver and receiver.

PET is built as a library to be used by the SPEC Chauffeur WDK [SPE17; Arn13] but can also be used as a standalone tool. While SPEC Chauffeur WDK generates transactions in a set interval to achieve the specified load levels, as explained in Section 2.3.1, PET takes care of triggering a configured number of performance events for each transaction. To validate PET, the performance counters are recorded and send to the Controller. As power analyzers have no standardized interface and vary in usable channels, measurement ranges and measurement accuracy, the SPEC PTDaemon interface [SPE12] is used to communicate with the power meter.

To trigger performance events, PET uses *modules* that provide one or more

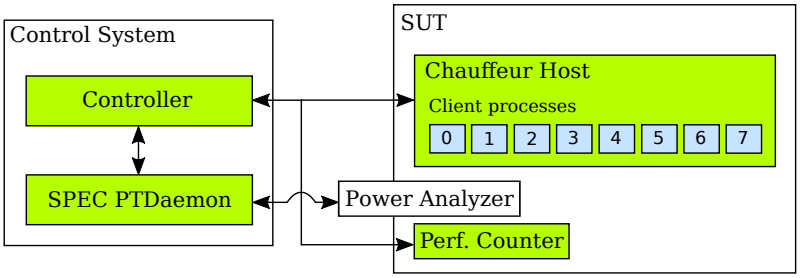


Figure 2.4: PET simplified testbed without additional networked devices.

functions that are designed to cause one type of performance event. An example of a single performance event trigger is shown in Figure 2.5. Starting from a performance counter in state n, a function is repeatedly executed until the configured amount of events i have been generated, bringing the counter state to $n + i$. The original PET implementation comes with four modules providing seven performance event triggers. These performance events that were selected and implemented as triggers showed a high correlation with power consumption.

- Level 3 cache misses
- Level 3 cache hits
- Bytes read from memory
- Bytes written to memory
- Instructions retired
- Hardware interrupts
- Context switches

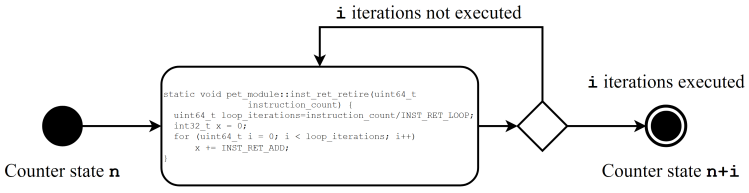


Figure 2.5: An example performance event trigger.

Side Effects Even with reasonable accuracy of the emulation, one issue of PET are side effects. A side effect is triggering a performance event through the synthetic code that is unintentional or sometimes even unavoidable. For example, causing a level 3 cache hit implies that the level 1 and level 2 cache must produce a cache miss event for the respective cache level. Another form of unavoidable side effect is executed code of a performance event trigger that counts towards the retired instructions. PET considers side effects for better accuracy of its emulation. PET can take side effects into account in two different variants, or the user can configure it to ignore them.

The first option to account for unwanted performance events is by accumulating all side effects caused by the other event triggers and subtracting them from the number of events to trigger. PET calculates the total side effects on a performance event s_x according to Equation 2.1. It sums up the product of side effects for a single event s_i and the number of events to trigger v_i. The actual amount of performance events that are triggered $v_{s,x}$ is calculated by subtracting all side effects s_x from the total performance events that need to be triggered v_x or is set to zero.

$$\begin{aligned} s_x &= \sum_{i=1}^{n} s_i * v_i \\ v_{s,x} &= \begin{cases} v_x - s_x & \text{if } s_x \leq v_x \\ 0 & \text{if } s_x > v_x \end{cases} \end{aligned} \tag{2.1}$$

The second option to mitigate side effects uses simulated annealing, as described by Henderson et al. [HJJ03] and Kistowski [Kis19]. Simulated annealing is less prone to get caught in local optima and tries to find the optimal solution $\omega^\star$, balancing performance events that must be triggered and side effects by iteratively generating neighbouring solutions ω' from the current one ω and comparing them over a distance function $f : \Omega \rightarrow R$. The function defines the distance to the optimal solution under the condition $f(\omega) \geq f(\omega^\star)$. $f(\omega)$ is defined as shown in Equation 2.2 as a modified Mean Squared Error (MSE), where $\hat{\omega}_i$ is the target of performance events to trigger, ω_i is the current value for the trigger i, and $\omega_{s,i}$ are the side effects of the current solution.

$$f(\omega) = \frac{1}{n} \sum_{i=1}^{n} (\hat{\omega}_i - \omega_i - \omega_{s,i})^2 \tag{2.2}$$

To create a neighboring solution ω', PET selects a random performance event trigger of the current solution ω_i and increments it by one. A possible solution is accepted with a certain probability shown in Equation 2.3. When a solution

is accepted, the temperature parameter t_k in iteration k is decreased. Simulated annealing can reach a steady state when the reduction in temperature is slow.

$$\Delta_{\omega,\omega'} = f(\omega') - f(\omega)$$
$$P\{\text{Accept } \omega' \text{ as next solution}\} = \begin{cases} e^{\frac{-\Delta_{\omega,\omega'}}{t_k}} & \text{if } \Delta_{\omega,\omega'} > 0 \\ 1 & \text{if } \Delta_{\omega,\omega'} \leq 0 \end{cases} \tag{2.3}$$

2.4 Benchmarks and Workloads

This sections describes the preexisting benchmarks and workloads used in this thesis. First, the SPEC CPU 2017 benchmark suite; and second, the TeaStore with a focus on the image provider service. According to Huppler and Kistowski et al. [Hup09; Kis+15a], a benchmark has five key characteristics it has to balance. We also apply these criteria to our TeaStore benchmarking reference application for the use in our thesis.

- Relevance
- Reproducibility
- Fairness
- Verifiability
- Usability

Common sorting algorithms are used for our second case study, and we argue that they fulfill one or more of the listed benchmark criteria. Namely, the *relevance*, as sorting is a crucial task executed often, and *reproducibility*, because they can be easily distributed and rerun on a large number of different systems. We do not further describe them in much detail and assume they are well known to the reader.

2.4.1 The SPEC CPU 2017 Benchmark Suite

This section describes SPEC CPU 2017, its suites and benchmarks, run and reporting rules, setup, and reasoning for selecting it in more detail. SPEC CPU 2017 is developed and maintained by the SPEC Open Systems Group (OSG) with defined submission, review, and publication procedures [KLK20].

The SPEC CPU 2017 benchmark suite is a compute-intensive benchmark from the SPEC using real-world benchmarks of different code and problem sizes. It is designed to stress a system's processing unit, memory, and compiler [BLK18].

"The purpose of the SPEC CPU 2017 benchmark and its run rules is to further the cause of fair and objective CPU benchmarking. The rules help ensure that published results are meaningful, comparable to other results, and reproducible." [Hen19b]

For our analysis, the important aspects are *relevance* and *reproducibility*. SPEC CPU 2017 clearly provides specified run and reporting rules. The report contains not only the results of a run but also the necessary information to reproduce the results again. The report, therefore, includes the SUT, the SUT's operating system, configuration, and compiler flags for the benchmarks, as well as additional information, increasing reproducibility.

A benchmark also must be relevant to the target audience. For our case, the target audience is energy-efficiency researchers and software developers. As shown in Table 2.1, SPEC CPU 2017 benchmarks come from a wide variety of application domains, ranging from pathfinding algorithms and compression to complex workloads such as artificial intelligence and simulation. Table 2.1 also shows the implementation languages, consisting of compiled machine-independent languages, C, C++, and Fortran. With the variety of languages, different programming paradigms are covered, from functional programming and procedural programming to object-oriented programming. Also presented are the Lines of Code (LOC) in KLOC (lines of code divided by 1000, including whitespaces and comments). The wide range of LOC from only 1000 lines up to over 1.5 million shows a broad spectrum of programs in terms of source code size. This large variety makes the SPEC CPU 2017 relevant for our case study to cover different real-world workloads that can be influenced by compiler optimizations.

The 43 benchmarks are organized in four suites, also shown in Table 2.1. The SPECspeed suites use a time-based metric of a single- or multi-threaded run. Inside the suite, the time required to run one task at a time is measured. The SPECrate suites use a throughput metric, a work per unit of time, in which multithreading is not allowed according to the run rules. Without multithreading, the tester running the benchmark still can decide on the number of copies of each benchmark task that he wants to run.

- SPECrate 2017 Integer, 10 benchmarks
- SPECrate 2017 Floating Point, 13 benchmarks
- SPECspeed 2017 Integer, 10 benchmarks

- SPECspeed 2017 Floating Point, 10 benchmarks

For each suite, a *base* and a *peak* result are reported in which different rules for compiler settings apply. In the *base* run, for each language, C, C++, and Fortran, or a combination of those, a set of optimizations flags must be selected. All benchmarks using this language or combination must use the same settings. In a *peak* run, the compiler settings can be different for each of the benchmarks listed in Table 2.1. The general setup for a SPEC CPU 2017 run, including power measurements, is outlined in Figure 2.1. It consists of the SUT running the benchmark and a controller system. As the temperature can have an influence on the power consumption of the server, a temperature sensor is used to validate the air temperature at the server air intake stays constant over time and does not exceed operational limits of the SUT. It also ensures reliable and accurate power measurements. Both the power analyzer and the temperature sensor are connected to the SPEC PTDaemon interface [SPE12], collecting the measurement data. Measurement devices obtaining the data must be supported by the PTDaemon interface and listed as an accepted measurement device [Hen19b; BLK18].

The large variety in workloads, and the SPEC CPU 2017 power measurements, provide a sound basis for our analysis of compiler optimizations and their influence on energy efficiency and performance. In contrast to the defined energy efficiency measurements detailed in Section 2.3.1, the SPEC CPU 2017 benchmark suite does not have load levels.

Table 2.1: SPEC CPU 2017 benchmarks with programming language and lines of code. [Hen19a]

SPECrate 2017 Integer	SPECspeed 2017 Integer	Language	KLOC	Application
500.perlbench_r	600.perlbench_s	C	362	Perl interpreter
502.gcc_r	602.gcc_s	C	1304	GNU C compiler
505.mcf_r	605.mcf_s	C	3	Route planning
520.omnetpp_r	620.omnetpp_s	C++	134	Discrete Event simulation - computer network
523.xalancbmk_r	623.xalancbmk_s	C++	520	XML to HTML conversion via XSLT
525.x264_r	625.x264_s	C	96	Video compression
531.deepsjeng_r	631.deepsjeng_s	C++	10	Artificial Intelligence: alpha-beta tree search (Chess)
541.leela_r	641.leela_s	C++	21	Artificial Intelligence: Monte Carlo tree search (Go)
548.exchange2_r	648.exchange2_s	Fortran	1	Artificial Intelligence: recursive solution generator (Sudoku)
557.xz_r	657.xz_s	C	33	General data compression
SPECrate 2017 Floating Point	SPECspeed 2017 Floating Point	Language	KLOC	Application
503.bwaves_r	603.bwaves_s	Fortran	1	Explosion modeling
507.cactuBSSN_r	607.cactuBSSN_s	C++, C, Fortran	257	Physics: relativity
508.namd_r		C++	8	Molecular dynamics
510.parest_r		C++	427	Biomedical imaging: optical tomography with finite element
511.povray_r		C++, C	170	Ray tracing
519.lbm_r	619.lbm_s	C	1	Fluid dynamics
521.wrf_r	621.wrf_s	Fortran, C	991	Weather forecasting
526.blender_r		C++, C	1577	3D rendering and animation
527.cam4_r	627.cam4_s	Fortran, C	407	Atmosphere modeling
	628.pop2_s	Fortran, C	338	Wide-scale ocean modeling (climate level)
538.imagick_r	638.imagick_s	C	259	Image manipulation
544.nab_r	644.nab_s	C	24	Molecular dynamics
549.fotonik3d_r	649.fotonik3d_s	Fortran	14	Computational Electromagnetics
554.roms_r	654.roms_s	Fortran	210	Regional ocean modeling

2.4.2 The TeaStore

We use the TeaStore image provider service in this thesis. We, therefore, give a short overview of what the TeaStore is and how it operates. The image provider service is then described in detail. This section is based on our TeaStore publications [Eis+18; Kis+18; Eis+19].

2.4.2.1 Overview

The TeaStore was developed as a benchmarking reference application that is peer-reviewed by members of the SPEC RG as a quantitative evaluation and analysis tool [Kis+19a]. It represents a basic web store that offers randomly generated tea and tea supplies. The user can browse, modify their shopping cart, and check out the shopping cart. The TeaStore consists of six microservices, five services for the webshop, and a registry, shown in Figure 2.6. Each service can be scaled as needed, and the services do not need to be on the same physical machine. The services communicate through Representational State Transfer (REST) interfaces and use the Netflix Ribbon [Net12] client-side load balancer. The TeaStore services come instrumented with Kieker [HWH12; Hoo+09] for performance monitoring. The TeaStore is built for three main purposes:

- Performance modeling, model extraction, and model learning.
- Performance management techniques.
- Energy efficiency and power modeling and optimization.

For energy efficiency measurements, as the TeaStore is request driven by an external load driver, load levels are not calibrated transactions but the number of requests that can be answered without problems. For our fine-grained power model in Contribution 4, we only use the image provider service and requests are issued directly to it.

WebUI The WebUI provides the user interface in the form of Java Server Pages (JSPs). Data about the products, images, and recommendations are fetched from the persistence, image provider, and recommender services. The WebUI also communicates with the authentication service for user and session data verification. Next to the user interface, the WebUI also performs user input validity checks before forwarding requests to other services. The performance of this service is dependent on the page that needs to be compiled and rendered.

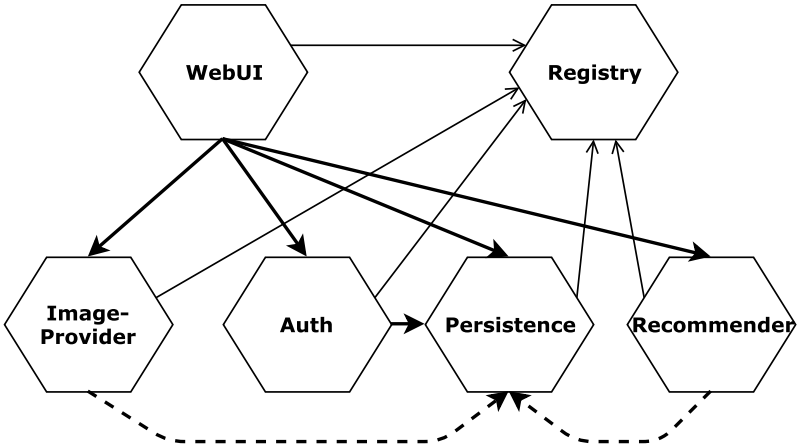


Figure 2.6: The TeaStore microservice architecture. [Kis+18]

Authentication The authentication service uses SHA-512 hashes to validate session data and BCrypt for user validation as the session data includes the shopping cart contents and past orders from a logged-in user. The performance of this service is, therefore, dependent on the size of the session data.

Recommender The recommender uses rating algorithms to offer customers products based on the current shopping cart content, the purchase history of other users, and the actual product the user is currently viewing. Different rating algorithms can be used to modify the performance behavior to be either CPU-intensive or memory-intensive.

Persistence The persistence service stores all necessary information of the TeaStore in a relational Structured Query Language (SQL) database, including products, product categories, past purchases, and registered users. The initial content of the database is generated on service startup but will be modified during benchmark runs.

2.4.2.2 Image Provider

The image provider serves images of different image sizes to the WebUI when being queried. It resizes an image when a size is requested that is not found in its dataset. The image provider also uses an internal cache and returns the image with the target size from the cache if available. Suppose the image is not available for this size. In that case, the image provider uses the largest available

image for the category or product, scales it to the target size, enters it into the cache, and stores it on the disc. It uses a least frequently used cache, reducing resource demand on frequently accessed data. Through caching, the response time for an image depends on whether this image is in the cache or not. This service queries the persistence service once on start-up to generate all product images with a fixed random seed. The performance of the image provider depends on the request types. The image provider will behave CPU-intensive if many images of the same product are queried in random sizes not encountered and memory-intensive when different product images in available sizes are requested. It will also stress the I/O system due to loading and storing images on disc.

2.4.3 Chauffeur WDK

All measurements with Chauffeur WDK in this thesis follow the methodology introduced earlier in Section 2.3.1. The remainder of this section is based on [SPE13; Arn13]. The primary design goal for the Chauffeur WDK is developing new benchmarks and tools like the SERT. The transactions to stress the SUT are defined in a worklet. A single worklet can have one transaction or multiple different transactions like the Server Side Java (SSJ) worklet. These worklets are grouped in workloads. The workloads are designed to stress certain aspects or components of the SUT. The Chauffeur WDK runs the contained worklets within a workload in a defined sequential order. All workloads together comprise the test suite and are also run sequentially. The setup of Chauffeur WDK is similar to Figure 2.1. We use three worklets from the Chauffeur WDK in our contributions, the Pi worklet, the XMLValidate worklet, and the SSJ worklet, which we describe in the following.

2.4.3.1 Pi Worklet

The Pi worklet comes from the ChauffeurTest package shipped together with Chauffeur WDK. It calculates π with an iterative approximation using the Gregory-Leibniz series shown in 2.4. The upper limit for n is set randomly for each transaction with values between $[1000; 100000]$. This worklet is especially stressful for the CPU but not other hardware components of the SUT. It is, therefore, a good reference worklet for CPU heavy loads without setting up a full benchmark suite like the SPEC CPU 2017.

$$\pi = 4 \cdot \sum_{k=1}^{n} \frac{(-1)^{k+1}}{2k-1} \tag{2.4}$$

2.4.3.2 XMLValidate Worklet

The XMLValidate worklet validates Extensible Markup Language (XML) files against a XML schemata as shown in Figure 2.7. It randomizes the input file by swapping commented regions within the XML file. Despite being used as a CPU benchmark, it also stresses the memory subsystem to a certain degree. Hence, it is also a more complex worklet than the Pi worklet.

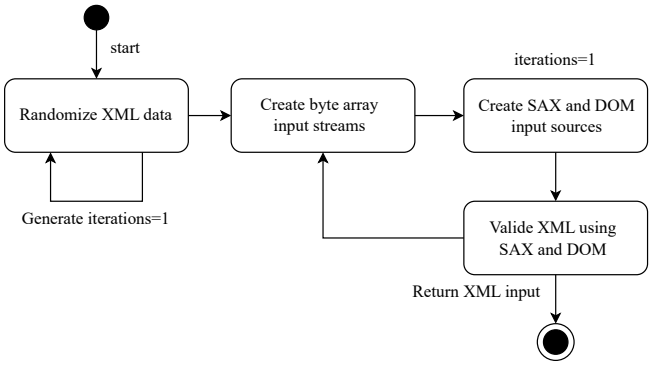


Figure 2.7: XMLValidate transaction. [SPE13]

2.4.3.3 SSJ Worklet

SSJ from the SERT test suite simulates Online Transaction Processing (OLTP) as a Server Side Java (SSJ) application. The SSJ worklet is a hybrid, which means it is stressing multiple components of the SUT, the CPU, caches, and memory. This worklet is closer to a real-world application than the Pi or XMLValidate worklets. It is, therefore, raising the complexity of our evaluation. The SSJ worklet includes six transactions as a worklet sequence and that it executes with certain frequencies. The mentioned frequencies are approximations of how often a transaction occurs.

New Order (30.3 %)	A new order is inserted into the system.
Payment (30.3 %)	Records a customer payment.
Order Status (3.0 %)	Requests the status of an existing order.
Delivery (3.0 %)	Processes orders for delivery.
Stock Level (3.0 %)	Finds recently ordered items with low stock levels.
Customer Report (30.3 %)	Creates a report of recent activity for a customer.

2.5 Machine Learning

For the classification of workloads to a resource usage profile in our *Contribution 3*, we give a short overview of the applied machine learning algorithms. Machine learning algorithms are split into two categories, *supervised* and *unsupervised*. For supervised algorithms, the training data contains the correct response for each input. In the case of unsupervised learning, this is not the case, and the response is unknown [Jam13]. We only apply supervised learning in this thesis: two decision tree models, a neural network, and one logistic regression. To improve the prediction accuracy of our machine learning models for the decision trees, we use two different techniques, bagging and boosting.

2.5.1 Bagging and Boosting for Decision Trees

Bagging, or bootstrap aggregation, does not create single but multiple decision trees, which can be up to hundreds or thousands of decision trees. The algorithm samples the original training data set to create varying subsets, which is called bootstrapping. The algorithm then builds a decision tree from each sampled subset. The outcome of each decision tree is averaged (aggregation) to predict the response variable. In case of categorization, the aggregation step could also be a majority vote [Jam+13]. A commonly used method is Random Forest that we use in this thesis. Random Forest not only samples the training data set but also generates new feature sets through sampling [Bre01].

Boosting is the second approach to improve the prediction of a decision tree. New decision trees can be added at the terminal nodes (leaves) given an initial decision tree for the training data. Instead of training the newly added decision tree to the response variable, it is trained for the residual, thus reducing the residual and improving the response. The algorithm adds new decision trees where the model is not performing well. New decision trees can be added iteratively to the original and sub-trees to further refine the prediction accuracy [Jam+13]. The decision tree implementation that leverages boosting to increase prediction accuracy and that we use in this thesis is XGBoost [CG16].

2.5.2 Neural Networks

We also apply a Feed-Forward Neural Network (FFNN) next to decision tree-based machine learning algorithms. A simple FFNN consists of layers of neurons that are interconnected. The first layer in a FFNN is called the input layer, and the last layer is the output layer. Each layer in between is called a hidden layer. A small FFNN with input, two hidden, and an output layer is shown

in Figure 2.8. While the number of neurons in the input and output layer is often determined by the problem the FFNN is applied to, the number of hidden layers and the number of neurons in that hidden layer can be freely selected. Information is only flowing in one direction from input to output layer without loops, hence, a feed-forward network. The output of a neuron is connected to all neurons in the following layer, and the connections are weighted. Changing the weight on a connection is how a FFNN is trained. The inputs of a neuron are inserted into an activation function that determines the output of that neuron. The activation function can be freely selected. Typical examples are the sign and sigmoid functions [Agg18].

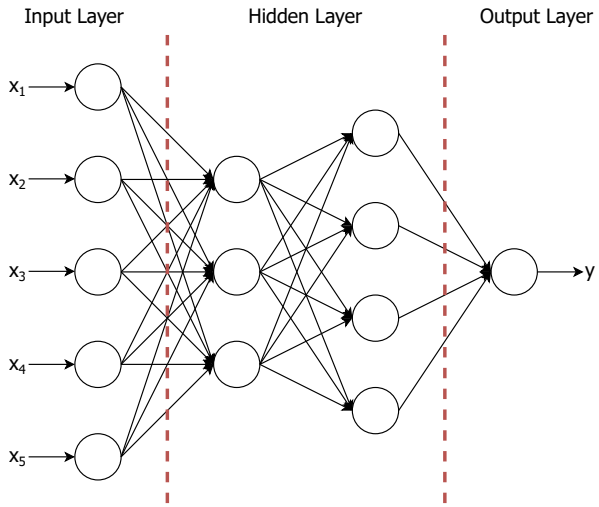


Figure 2.8: Basic FFNN with input, hidden, and output layers. [Agg18]

2.5.3 Logistic Regression

Logistic regression is mainly used in the field of medicine, economy, social, and physical science but is not limited to those fields. The basic logistic regression model is used to model its input onto a binary (or dichotomous) response variable but can be expanded to multiple categories, multinomial (or polytomous) logistic regression. If the categories are ordered, it is called an ordinal logistic regression [Hil09]. The logistic regression is based on the logistics function shown in Equation 2.5 [KKP02]. The output of $f(z)$ stays between zero and one, and in a binary logistic regression is the probability for a condition z.

Given a set of independent variables $\mathbf{X}$, z is replaced with the term $\alpha + \sum \beta_i X_i$, similar to the linear regression model, and describe a conditional probability given the input $\mathbf{X}$. The conditional probability is shown in Equation 2.6 [KKP02]. While Equation 2.6 is still binary, it can be extended to handle multiple categories and even ordinal response variables. The unknown variable α and the unknown coefficients β_i can be determined with a maximum likelihood technique [Hil09].

$$f(z) = \frac{1}{1 + e^{-z}} \tag{2.5}$$

$$P(\mathbf{X}) = \frac{1}{1 + e^{-\left(\alpha + \sum \beta_i X_i\right)}} \tag{2.6}$$

Chapter 3

Related Work

In this section, we will give an overview of the existing work. This section also shows where our identified problems are located in the state-of-the-art and from which shortcomings in literature they arise. We divided the related work into three categories, the modeling of power and energy consumption based on monitoring a systems state in Section 3.1, improving the power and energy consumption in Section 3.2, and workload classification approaches in the context of power and energy research in Section 3.3.

- Power and energy consumption models in Section 3.1 are relevant as we propose two new power models. A high-bandwidth memory model in Chapter 9 and a power model that works on the programmatic function-level in Chapter 10. Our focus in Section 3.1 is on the performance event models as our newly developed models are based on them as well. Some power models also provide insights if performance event-based models are transferable across systems, relevant to our contribution in Chapter 11.

- We describe the current state in improving the energy efficiency as it is relevant throughout this thesis, for our modeling contributions, but also our case studies in Chapter 5 and Chapter 6. These works give insights into the current state of identifying potential for energy efficiency increases.

- Investigating workload characterization in Section 3.3 is relevant as we designed a new approach to improve power modeling by selecting a suitable model based on characterizing performance events into workload profiles in Part III. Our focus is, therefore, on classification approaches that leverage performance events.

3.1 Power and Energy Consumption Models

Multiple works [IM03; SBM09; Che+10; LPF10; BJ12; Rod+13; Liv+12; Tsa+14] use performance events to create power consumption models for various applications. What most of them have in common is that they model the total system power. Some also model different subsystems of the hardware that can be memory or CPU internal subsystems like caches. Modeling the power and energy consumption is particular relevant for our thesis for our two new power models in *Contribution 4* and *Contribution 5*. Additionally, performance events are extensively used throughout this thesis.

Isci and Martonosi [IM03] model the power consumption of a system with performance events. Rather than a complete power model, they break down the power consumption model into different component models. This split allows them to estimate the power consumption of the CPU, like cache and memory bus. While achieving a reasonable accuracy, their approach only allows attributing the power consumption to CPU parts rather than the application itself. Modeling the power consumption of the entire application is possible with Isci and Martonosi's approach is focused on the rather old Pentium 4 hardware.

Singh et al. [SBM09] are modeling the power consumption of the hardware with performance events and achieve reasonable accuracy. Nevertheless, their work focuses on system and CPU power consumption to allow the operating system scheduler to operate more energy efficiently. Focusing on the system and CPU power does not allow to differentiate between the many currently running applications and is, therefore, too coarse to help developers write energy-efficient code.

Chen et al. [Che+10] is more accurate than the work from Singh et al. by being able to attribute performance events to specific processes running on the system. This approach allows them to build a power model with reasonable accuracy, reaching an error of 7.7%. However, their approach does, as many others, only include the CPU and no other components of the system that consume and draw power. Additionally, it is too coarse to identify pieces of an application that consume the most power and is only limited help to application developers.

Lim et al. [LPF10] developed *SoftPower*, that trains a linear regression model with a mean accuracy of 7% to 14%. They take CPU and memory power consumption into consideration by selecting performance events for these two subsystems. However, their model only estimates full system power. This full system power approach makes it, as the other models, too coarse.

Bircher and John [BJ12] use the trickle-down effect to model a system's power consumption. The trickle-down effect, described in the author's earlier work [BJ07], states that power consumption of a subsystem like the memory that is correlated to certain performance events is observable at the CPU as it propagates from the subsystem to the CPU. Most power models based on performance events that model subsystems do rely on this trickle-down effect. Still, both models proposed in Bircher and John's work estimate power consumption on the whole system level rather than the application level.

Rodrigues et al. [Rod+13] identified the possibility to use microarchitecture-independent performance events across different CPUs, low-performance, and high-performance CPUs. Microarchitecture-independent events, described in Section 2.2, behave similarly across a range of different CPUs of one vendor. They identified several performance events that are usable for power modeling, and they model and evaluate full system power.

Lively et al. [Liv+12] do model the power consumption of the memory and CPU for certain applications. However, they restrict themselves to scientific applications with high computational demand for High Performance Computing (HPC) systems. Tsafack et al. [Tsa+14] use a similar approach modeling HPC applications with performance events. While this is similar to our work and can be of value for software developers of HPC operators, our work focuses on data center workloads with high variability in load and allow more insight into the application itself.

Dong et al. [DZ11] presented a self-modeling approach for a mobile system. This automatic generation for a power model uses the smart battery interface. Thus, it allows the system to generate its energy model without external assistance and expert knowledge. While this approach presents a good precision, the results have been achieved only on mobile hardware, especially relatively old hardware that does not yet employ modern power-saving technologies.

For Java-based software systems, Seo et al. [SMM07] present the first iteration of a framework for estimating power consumption. This approach focuses on the interaction among distributed components. Thus, it allows developers to estimate their systems' power consumption at design time. However, the component-based approach is coarse, making informed decisions for developers of single components complicated due to the high abstraction.

Stier et al. [Sti+15] modeled the power consumption of software systems with the Palladio Component Model on an architectural level, achieving an error of less than 5.5%. Nevertheless, modeling the power consumption, an abstraction itself, on a high level can lead to a better architecture but is too coarse to make informed decisions for developers of single components of a

software system, similar to Seo et al. [SMM07]. Hence, it can achieve only an improvement on the architecture level but not on an algorithmic level.

Mammeri et al. [Mam+19] is using a FFNN, as described in the Foundations 2.5.2, with 24 input neurons and two hidden layers, each with 50 neurons. The 24 input neurons are the performance event values. Mammeri et al. did use performance events from both the CPU and the GPU. Mammeri et al. used OpenGL and OpenCL benchmarks to stress both components. They claim that they outperform statistical linear regression models by 3.3 times. They also showed that a FFNN does not need to be excessively large and that a FFNN with over 3000 neurons did not yield better results. As their approach focuses on System on Chips (SoCs) for mobile devices, it is of limited applicability in our approach. However, it shows that machine learning approaches can model the power consumption equally or better than statistical linear regression. This result is expected as a chip's power consumption characteristic is non-linear.

Existing power models are either too generic modeling full system power or too specific, for example, for HPC or SoC applications, for use in our context. Other simple full-system power models, such as [FWB07; RRK08] do not consider other components of a system, like memory access, sufficiently. In stark contrast, models intended for hardware design [RJ00; MD04; Li+09] are too specific, requiring detailed information about the target hardware. Similarly, some more general-purpose models which consider other components like memory accesses, such as [Gur+02] also require detailed hardware information. More generic models are mostly intended to be used for system management and thus still require some concrete hardware descriptions. [BD12] features a processor power model, whereas [BC10] and [VAN08] focus on management of virtual machines.

Those models are suitable to determine how much power and energy a system or application consumes and how the energy efficiency can be calculated. However, they lack the ability to guide developers on improving the energy efficiency of their applications without expert knowledge. Additionally, the MSR counting the performance events are known to be inaccurate in certain situations, and the authors of [ZJH09] name multiple problems when using performance counters: complex configuration, missing programmability, the requirement of root privileges, and the lack of discrimination between the triggering threads. The authors then perform a comparative study of the accuracy of three commonly used measurement infrastructures for performance counters on three different processors. While this is correct for older architectures, CPU vendors now provide higher accuracy for most counter values. Thus, performance event-based power and energy modeling can still be accurate.

3.2 Improving Energy Efficiency

Next to modeling the power and energy consumption of applications to know that the amount of energy an application actually consumes, developers must know where in the application an improvement is sensible. Improving the energy efficiency is specifically important to our thesis as we raise awareness by showing possibilities to developers in our two case studies in *Contribution 1* and *Contribution 2*. Therefore, identifying these code pieces where an improvement is sensible is necessary. How the identified section of the code can be improved needs additional information and knowledge. A gap that has already been identified by Pinto and Castor [PC17]. Hence, we divide this section into two items, identifying inefficient code in Section 3.2.1, and improving the energy efficiency in Section 3.2.2. The improvements are focusing on compiler optimizations due to the stated knowledge gap.

3.2.1 Identifying Energy Inefficient Code

For a developer to produce energy-efficient software, it is often not known which changes to the source code will have a beneficial impact on energy efficiency. Hence, developers could rely on additional information in the form of runtime monitoring of the deployed software or models based on the available source code and architecture of the software. Another option is not to explicitly develop the software for a better energy efficiency but let the compiler introduce optimizations similar to the performance optimizations. These two approaches divide the topic of improving the energy efficiency of software into two categories, first improvements during design time by the developers and second, optimizations during compile time.

For mobile devices power and energy consumption are pressing issues, primarily if the devices are powered by batteries. In [Li+13] Li et al. present an approach to measure power consumption on a source line level. The authors combine hardware-based power measurements with program analysis and statistical modeling. While executing an application, the approach measures the energy and derives the executed parts of the application using path profiling. The per-line consumption is then evaluated using static and regression analysis as presented in the authors' previous works [Hao+13]. However, this approach requires a significant amount of resources to perform the profiling.

The work presented in [DZ11] studies a self-modeling approach for a mobile system. Thus, it allows the system to generate its energy model without external assistance. This generation uses the smart battery interface. While this approach presents a good precision, the results have been achieved only

on mobile hardware, and especially, relatively old hardware that does not yet employ modern power-saving technologies.

Pathak et al. introduce *eprof*, a fine-grained energy profiler for smartphone apps [PHZ12]. When applied to several apps, eprof exposes energy drainers like third-party advertisements or pinpoints wake lock bugs in the code. Next, *bundles* are introduced to help developers optimize their app's energy drain by presenting the app's I/O energy consumption.

Another tool-assisted approach by Zhang et al. [Zha+10] introduces two tools. *PowerBooter* constructs a power model without using a power meter. *PowerTutor* is a tool that, using online analysis, shows developers the implications of their design choices on power consumption. Unlike [Li+13], this approach is not based on the code line level but instead on the component level. Thus, the model is built using the consumption of the CPU, LCD, GPS, Wi-Fi, 3G, and audio components.

Banerjee et al. [Ban+14] generate an event flow graph of a mobile application and trace the execution to find pieces of code related to high energy consumption. They distinguish between energy hotspots, shorter peaks, and energy bugs that result in increased energy consumption over extended periods of time. This approach aids the developers in showing them that energy hotspots and bugs are present and locates them in the code.

These works on mobile devices have in common that they either use the sensors of a mobile device, tracing, profiling, or Java's bytecode. While this might work for the specific device or language and gives insight into where energy is wasted, it is complicated to derive accurate, practical guidance without intensive measurement setups [PHZ12; Ban+14] for developers to decrease energy wastage and increase energy efficiency.

Other works shift their focus away from modeling the power and energy consumption of devices and relate to the energy-efficiency of software without regard to what type of hardware is executing the application. As a good example, Capra et al. measured and analyzed complete software systems: two ERP systems, two Customer Relationship Managements (CRMs) systems, and four Database Management Systems (DBMSs). The authors could show that software can differ considerably (50%) in energy efficiency without significant differences in performance (5%) in extreme cases. Further, they found that this discrepancy stems from using hibernation during external requests and that algorithmically more efficient software also can be more energy-efficient [CFS12].

Consequently, Aggarwal et al. [AHS15] developed *GreenAdvisor* based on their previous work [Agg+14] about system-calls as their primary source to predict power consumption. GreenAdvisor is a tool that analyzes the appearance

of system calls in an application and maps them to the corresponding function call inside the application. The change of system call behavior across multiple versions is tracked to determine if a significant change in energy consumption has occurred.

Bunse et al., as in our work, used measurements of six sorting algorithms, with some in both iterative and recursive implementation, to define a trend function, allowing the developers to choose between performance and energy consumption. They continued by using this trend function to increase the battery lifetime of mobile devices [Bun+09]. Rashid et al. compared sorting algorithms on an ARM platform, popular for mobile devices, showing that the algorithm and language do affect the energy consumption [RAT15]. While an interesting and promising approach, in our opinion, mobile devices often are connected to the cloud to fulfill heavy computational tasks, that while conserving energy for a longer battery lifetime, they shift the energy consumption into data centers.

Chandra et al. conducted a basic study on the power consumption, energy consumption, and runtime of sorting algorithms for their work. They found that the energy consumed is related directly to its time complexity and that integer sorting takes less energy than sorting floating-point data [CPK13]. Again, their focus is not on server systems but desktop machines and a small problem size of just $10,000$ integer or floating-point numbers.

It can be seen that the focus of energy-efficiency research is on mobile devices, and there is a gap of knowledge on data center hardware. Therefore, we selected state-of-the-art server systems for our measurements. Additionally, most works are concerned with finding the location inside an application where energy is wasted. The actual improvements in energy efficiency often are left to the compiler leveraging compiler performance optimizations.

3.2.2 Leveraging the Compiler to Improve Energy Efficiency

Compilers usually are not targeting energy efficiency with the available optimizations; their main target is performance. Hence, to optimize for energy efficiency, the correct optimization settings must be known to adapt a compiler to improve energy efficiency [PHB13], or there must be adaptation points added during compilation.

Nobre et al. [NRC18] are changing the order in which performance optimizations are performed during compilation. They have shown that while an increase in energy efficiency is possible, the sequence of optimization is dependent on the application and not necessarily transferable to other applications. Kandemir et al. [Kan+00] focused on six loop optimizations of a compiler and

tested them on five applications and simulations with mixed results. Most loop optimizations increased the power consumption in the embedded systems.

The tool *Socrates* from Gadioli et al. [Gad+18] not only tries to find suitable compiler optimizations but also weaves in code that can be used for tuning the software to different targets, like power consumption or performance. The additional instructions or parameters are used, for example, to adapt the accuracy of numerical approximations or the number of OpenMP threads. Socrates creates several binary versions that are compiled and profiled to optimize at runtime. Hsu et al. [HK03] inserted instructions to control the Dynamic Voltage Scaling (DVS). The approach selects program parts suitable for running with lower voltage and frequency without degrading performance above a user-selectable value resulting in energy savings of up to 28% and performance degradation of 5%, increasing energy efficiency.

Gheorghita et al. [GCB05] argue that iterative compilation, generating different versions of source code, can be used to optimize energy efficiency for mobile embedded systems. Trying to find the Pareto optimal set of compiler flags for performance, Hoste and Eeckhout used an evolutionary algorithm to increase energy efficiency [HE08]. Martins et al. [Mar+16] used clustering to find the best order of optimization settings for performance on a soft microprocessor inside an FPGA.

Our work extends the related work by aiming not at mobile devices but cloud servers. Additionally, our work is intended as a guideline or help for practitioners to select a better choice of sorting algorithm for energy efficiency before compilation and to leave the fine-tuning to approaches that leverage compiler optimizations. By focusing on the developers selecting better algorithms and not relying on a black-box in the form of the compiler, allows us to increase a developer's awareness and proficiency in writing more energy-efficient software from the start.

3.3 Workload Classification Based on Performance Events

Workload classification is an essential aspect of our thesis and, therefore, relevant, specifically in our *Contribution 3* where we use classification to assign applications to workload profiles based on their resource usage in the form of performance events.

Jia et al. [Jia+13] use k-means clustering to classify micro-architectural performance event characteristics of data analysis workloads. Their data analysis workloads are run on three common cloud workloads, which are, according

to Jia et al., electronic commerce, social networks, and media streaming. They found that data analytics workloads are diverse in their performance event characteristics. This diversity confirms that our approach using performance events for characterization is feasible and can result in meaningful insights. However, Jia et al. do not discuss the accuracy of their characterization that leads us to other machine learning approaches.

Panda and John [PJ14] also use k-means clustering for classification, very similar to Jia et al. Panda and Johns classified applications, on the other hand, come from benchmarking suites, namely SPEC CPU2006 [SPE21a], SPECjbb 2013 [SPE21c], and TPC-H [TPC21]. They compare the diversity of the TPC-H benchmarks against the SPEC benchmarks. They conclude that TPC-H has a wider variety of workloads. The classification approach works with Jia et al., further confirming that performance events are the correct measure to define our workload profiles.

Choi et al. [CPN18] classify HPC applications. Their goal is to optimize resource usage through workload scheduling and better resource management, similar to Kandalintsev et al. They used profiling tools to divide their used benchmarks, eleven benchmarks from the NAS Parallel Benchmark (NPB), into three resource categories. The applications are then clustered by expectation-maximization clustering on low, medium, and high resource usage. They were able to confirm that the resource profile classified by their approach matched the category of the respective benchmark. At the same time, this approach is closest to our approach but focuses purely on HPC applications and not cloud computing. This HPC focus also leads to a low number of benchmarks as representative applications.

Kandalintsev et al. [Kan+14] classify virtual machines according to their performance. They are specifically interested in predicting the performance. The performance is affected in the case many virtual machines are located on the same physical server, and the virtual machines interfere with each other and cause performance degradations. Kandalintsev et al. use a mix of real-world applications, like nginx, and synthetic benchmarks, like large matrix multiplications. Predicting the performance degradation is helpful for resource allocation and scheduling in cloud data centers. Our approach does not aim to predict performance degradations to improve scheduling but to predict power consumption more accurately. Although avoiding performance bottlenecks can improve energy efficiency, it is not their primary target.

Zaman et al. [ZAM15] is closer to our approach as it forecasts power consumption and temperature of a system. They also leverage performance events to classify workloads into profiles and train an Support Vector Machines (SVMs)

to predict the power consumption and temperature. The predicted information is to improve the workload scheduler. Their main goal is to extend the system's lifetime by reducing temperature hotspots created by excessive power draw. Compared to our approach, one of the significant drawbacks is that the authors do not use power analyzers but rely on power and a temperature simulation tool. Our approach does not directly predict the power consumption but instead selects a power model that is more accurate for a given workload profile or execution phase. This two-step approach allows us to use simpler power models instead of one model that needs to be trained to all possible applications that could be encountered in a cloud data center.

Wei et al. [Wei+19] classify applications according to their power consumption. They defined a power model based on 14 discriminant functions, linear combinations of the predictor variable, the performance events. Based on this power model, applications are grouped into 15 groups, each with different power consumption. The distance between the groups is $5\,\mathrm{W}$. Their work aims to improve scheduling and power consumption and, therefore, the energy efficiency of the application. Although it is unclear how the classification of the entire application aids developers in writing better code as the power-consuming parts are not identified in their approach.

The presented related work shows that workload classification is an ongoing topic, but the main focus is not cloud data center applications but HPC. In HPC, scheduling the different tasks that need to be processed is critical, which is why many authors focus less on improving the application itself but on the scheduling. In contrast to our work, we want to improve the power prediction for cloud applications, a goal that is also mentioned by Wei et al. [Wei+19].

Part II

Energy Efficiency Improvement Opportunities

Chapter 4

Software Influence on Power Consumption

A growing number of services are deployed in the cloud. Additionally, an increasing number of devices are using the cloud to provide functionalities and enabling data centers to grow. This growth, in turn, results in a rise in energy demand with an estimated total energy consumption of $1929\ \mathrm{TW\,h}$ until 2030 in the expected scenario by 2030 [And19]. Moreover, while advances in hardware increase their energy efficiency for sustainable growth, they cannot fully compensate for the increase in energy demand in data centers if not all opportunities to conserve energy are exhausted [Mas+20]. Therefore, it is essential to include software as a significant part of conserving energy and increasing energy efficiency. Nevertheless, there is a lack of awareness and proficiency on achieving better energy efficiency in software [PC17].

Cloud data centers can be made more efficient. For instance, it is possible to conserve energy by intelligently placing or consolidating the workload [JWC12] or using only the minimal amount of resources to satisfy user demands through auto-scaling. The server hardware used in data centers also has become more efficient with the introduction of DVFS and demand-based CPU sleep states, known as C-states. Benchmarks, on the other hand, create an incentive for manufacturers to improve energy efficiency of their hardware, such as the SPECpower_ssj 2008 benchmark [Sch+21b; SPE19]. Figure 4.1 shows an increase in efficiency score from 190 to 21603 for single cpu and node systems. Nonetheless, the executed software itself, has not gotten much attention in terms of energy efficiency.

While the server hardware consumes the energy, it is the software that, for the most part, controls how the server behaves and, therefore, indirectly controls how much energy the hardware uses. Initial measurements on storage servers show that the power consumed by a server is influenced by the workload it is running [Lan09b]. Another example is Capra et al. shows that two different enterprise resource planning software systems have little performance differ-

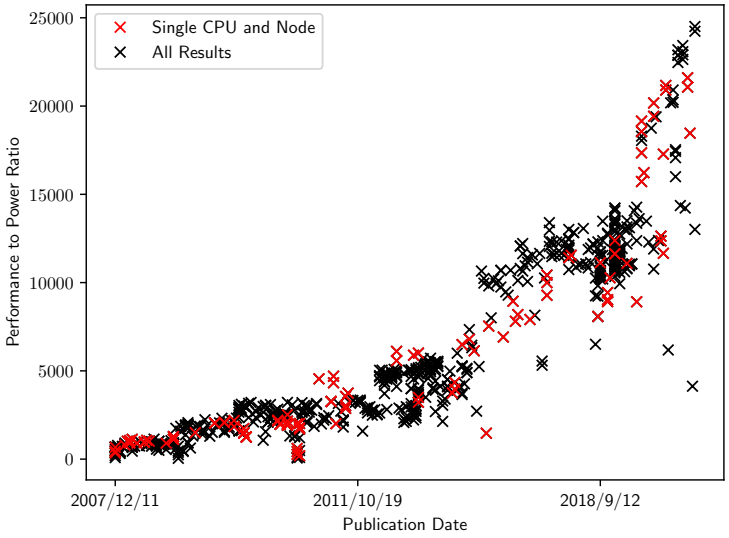


Figure 4.1: Reported performance to power ratio in ssj_ops / sum of power of the SPECpower_ssj 2008 official benchmark results. [Sch+21b]

ences (+5%) but deviate widely in energy consumption (+50%) [CFS12]. The two system are showing a better energy efficiency for the tested system over the baseline. For a fair comparison of software, it must perform the same task but with a different code or a code compiled with different options. Hebbar et al. already showed that different compilers influence the energy efficiency through better utilization of the hardware [HM19]. Another option to achieve better energy efficiency is to use compiler optimizations that modify the source code for higher performance or optimize for size while the functionality is left untouched.

We first confirm that energy efficiency is not only a matter of efficient hardware but is also influenced by the software running on said hardware. Following the principle that applications indirectly control the hardware they are running on [CP15a], we base our approaches on improving the power consumption of the software rather than the hardware. To investigate the validity of this principle and in turn our approaches, we measured the energy consumption (power over time) of three different variants of a REST service for 240 s. This service performs caching and resizing image files stored on a physical drive. While the resizing needs processing by all three variants, the implementation of the data structure for the cache varies. Variants include a Random Replacement (RR) strategy as a linked list, a Least Frequently Used

Table 4.1: Energy consumption of different cache implementations per request over a 240 s measurement period.

Implementation	Energy in Ws
Direct Drive Access (DDA)	3.78
Least Frequently Used (LFU)	3.68
Random Replacement (RR)	3.43

(LFU) strategy with a red-black tree, and no cache with Direct Drive Access (DDA). The results in Table 4.1 show that the DDA uses the most amount of energy per request. It seems that the constant drive access is responsible for the higher energy consumption but even LFU and RR exhibit differences in energy consumption supporting our assumption. With a constant request rate, the RR caching strategy will consume the least amount of energy in this scenario. The lower energy consumption for RR most likely occurs due to the random selection of images. Singh et al. also confirmed the impact of software on efficiency by showing that different software designs, implementations, and configuration parameters for the same logic result in different energy consumption [SNM15; CFS12].

Chapter 5

Measuring the Impact of Compiler Optimizations

After we confirmed that software does influence the power and energy consumption of a typical server, we continue by analyzing the possible ranges of energy efficiency achievable with different compiler options. We base our analysis on the SPEC CPU 2017 benchmark suite [KLK20; BLK18]. With 43 benchmarks from different domains organized in four suites with integer and floating-point heavy computations, the benchmark suite has a large collection of workloads that we consider to be representative of a variety of real-world workloads, shown in Table 2.1 in Section 2.4.1. The analysis allows insight into which type of workload is influenced by changing the code through compiler optimizations without changing its functionality. We analyze the differences of the compiler optimizations on the performance, power utilization, and energy efficiency between two different scenarios: First, a typical real-world scenario that consists of compiler flags not chosen for the highest benchmark score, and second, the expert, highly optimized scenario used for the SPEC CPU 2017 suite.

The main contribution of this work is an analysis of impact of compiler optimizations on how the different benchmarks behave in terms of performance, power consumption and energy efficiency and not a single optimization goal as is done by most work in our related work Chapter 3. This contribution, therefore, allows us to address *RQ A.1* of *Goal A*. This section is based on our publication in the Proceedings of the 11th ACM/SPEC International Conference on Performance Engineering (ICPE) 2020 [Sch+20c] and the 13th IEEE/ACM International Conference on Utility and Cloud Computing (UCC) 2020 [Sch+20a]. We also made the raw data for the compiler case study publicly available [Sch+20b].

The remainder of this chapter is structured as follows. First, we explain the scenarios for our measurement together with the definition of energy efficiency for this use-case and evaluation approach in Section 5.1. In the next Section 5.2

we give a short description of the SUT used in the measurement setup already described in our Foundations 2.3.1. We then analyse our results in Section 5.3 with the concluding remarks in Section 5.4.

5.1 Defining the Real-World and Optimized Scenario

We define our two scenarios under which we measured, called the real-world scenario, and the optimized scenario. We then describe how the energy efficiency is defined for our evaluation and a short overview of evaluation approach.

5.1.1 Real-World versus Optimized Scenario

We reduce the aggressive settings, refined by the server manufacturer to achieve the best outcome, to a more common real-world scenario. We have done so by removing highly specific values for all four benchmark suites, for example, `inline-threshold=1000` or `loop-unswitch-threshold=20000`. Such specific values might lead to a performance increase for one CPU or workload but not necessarily for another. In our opinion, overly specific values are also ordinarily unknown to the developers. This means, they are highly specialized to the SUT as well as to the corresponding benchmark. We also reduce the general optimization setting `-O` from 3 to 2 as well as remove optimizations that would not be included, even in `O3`. Another reason to change from `-O3` to `-O2` is that the AOCC compiler uses even more optimizations than the LLVM on which it is based [Nag19a; Nag19b] in `O3`. This way, we are able to achieve a set of compiler flags shown in Table 5.1 that is closer to real-world applications. We did not remove vectorization level AVX2 that might have an impact on performance as they are included in `march=znver2`, a typical setting for compilation. Hence, our setup does not allow a comparison of vectorization techniques, like SSE4.2 versus AVX2. While we expect that the real-world scenario can reduce performance, a higher performance or better benchmark score may not necessarily lead to better energy efficiency.

Table 5.1: Compiler optimization flags for comparison real-world scenario.

Language	SPECrate 2017 Integer
C	`-flto -Wl, -mllvm -O2 -march=znver2 -Wl, -vector-library=LIBMVEC -mllvm -vector-library=LIBMVEC -z muldefs -lmvec -ljemalloc -lflang`
C++	`-flto -Wl, -mllvm -O2 -march=znver2 -Wl, -vector-library=LIBMVEC -mllvm -vector-library=LIBMVEC -z muldefs -lmvec -ljemalloc -lflang`
Fortran	`-flto -Wl, -mllvm -O2 -march=znver2 -Mrecursive -Wl, -vector-library=LIBMVEC -mllvm -vector-library=LIBMVEC -z muldefs -lmvec -ljemalloc -lflang`
Language	**SPECrate 2017 Floating Point**
C	`-flto -Wl, -mllvm -O2 -march=znver2 -Wl, -vector-library=LIBMVEC -mllvm -vector-library=LIBMVEC -z muldefs -lmvec -ljemalloc -lflang`
C++	`-flto -Wl, -mllvm -O2 -march=znver2 -Wl, -vector-library=LIBMVEC -mllvm -vector-library=LIBMVEC -z muldefs -lmvec -ljemalloc -lflang`
Fortran	`-flto -Wl, -mllvm -O2 -march=znver2 -Mrecursive -Wl, -vector-library=LIBMVEC -mllvm -vector-library=LIBMVEC -z muldefs -Kieee -lmvec -ljemalloc -lflang`
Fortran / C	`-flto -Wl, -mllvm -O2 -march=znver2 -Mrecursive -Wl, -vector-library=LIBMVEC -mllvm -vector-library=LIBMVEC -z muldefs -Kieee -lmvec -ljemalloc -lflang`
C / C++	`-flto -Wl, -mllvm -O2 -march=znver2 -Wl, -vector-library=LIBMVEC -mllvm -vector-library=LIBMVEC -z muldefs -lmvec -ljemalloc -lflang`
Fortran / C / C++	`-flto -Wl, -mllvm -O2 -march=znver2 -Mrecursive -Wl, -vector-library=LIBMVEC -mllvm -vector-library=LIBMVEC -z muldefs -Kieee -lmvec -ljemalloc -lflang`

5.1.2 Energy Efficiency for the SPEC CPU 2017

Next to performance, the SPEC CPU 2017 benchmark suite also reports the energy consumption of the SUT as well as the power consumption over time. Each system's energy consumption E consists of two parts, the static energy E_s and the dynamic energy E_d, as shown in Equation 5.1. The dynamic energy is due to the running software. The static energy for our case is equal to the idle power times the time-to-result of the benchmark. For energy efficiency, only the dynamic energy is considered. Hence, we subtract the idle energy from the total energy consumption E reported in the results. We consider only dynamic energy because only E_d should be influenced by the compiler optimizations in

an optimal setting while the static energy is influenced mainly by the hardware.

$$E = E_s + E_d \tag{5.1}$$

We define energy efficiency as the ratio of work to the energy consumed in Equation 5.2. For the throughput metric of the SPECrate suites, the amount of work is the number of copies for a benchmark run, (usually the number of logical cores, 256) for our SUT. In the SPECspeed suites, the amount of work is always one independent of its runtime. Even with multiple threads, all threads are computing one single unit of work. We average all three runs to compute the energy efficiency. It expresses the number of copies executed per $1000\,\mathrm{W\,s}$. As the energy efficiency is not normalized and reports units of work per $1\,\mathrm{kW\,s}$, it can reach values above one.

$$\mathit{eff} = \begin{cases} \frac{work}{E} & \text{if SPECrate} \\ \frac{1}{E} & \text{if SPECspeed} \end{cases} \tag{5.2}$$

5.1.3 Approach

We first take a look at how the performance, in the form of throughput (SPECrate) or time-to-result (SPECspeed), and energy-efficiency changes. We use our two different scenarios for this comparison. While the SPEC CPU 2017 is a well-respected and accurate benchmark, the averages are over only three iterations, so we do not report confidence intervals so as not to give the impression of a high number of consecutive runs. The comparison ought to show that the influence of drastic compiler optimization changes the performance, energy efficiency, and power consumption behavior. For the real-world scenario, we use the adapted compiler flags from Table 5.1 while we run the optimized benchmark suites with compiler flags from the official results.

After taking a look at the performance and energy efficiency, we focus on the power consumption of the SPECrate suites over time. To do so, we plot the power consumption of each of the SPECrate suites benchmarks in a box plot for a first insight on the behavior of the different benchmarks. We assume that the power consumption is not normally distributed. We also consider outliers as warmup and cooldown phases at the beginning and end of a benchmark that cannot be distinguished from a drop in power consumption during a run. Warmup and cooldown phases are usually caused, but are not limited to, filling caches, loading and storing data on drives, or frequency increases until reaching the thermal limit. We selected a box plot to make the variability of power

consumption visible across both scenarios. The scenarios are comparable and independent due to the compiler optimizations producing different binaries.

For our analysis, we also group the benchmarks into four application domains shown in Table 5.2. The four categories are *language transformation*, *artificial intelligence*, *modeling and simulation*, and *others* that do not fit in the mentioned categories. The grouping allows us to analyze if a particular application domain is susceptible to compiler optimizations. Each group consists of at least three benchmarks.

Table 5.2: SPEC CPU 2017 benchmarks with application domain. [Hen19a]

SPECrate	SPECspeed	Application	Domain
500.perlbench_r	600.perlbench_s	Perl interpreter	Language Transformation
502.gcc_r	602.gcc_s	GNU C compiler	Language Transformation
505.mcf_r	605.mcf_s	Route planning	Other
520.omnetpp_r	620.omnetpp_s	Discrete Event simulation - computer network	Modeling and Simulation
523.xalancbmk_r	623.xalancbmk_s	XML to HTML conversion via XSLT	Language Transformation
525.x264_r	625.x264_s	Video compression	
531.deepsjeng_r	631.deepsjeng_s	Artificial Intelligence: alpha-beta tree search (Chess)	Artificial Intelligence
541.leela_r	641.leela_s	Artificial Intelligence: Monte Carlo tree search (Go)	Artificial Intelligence
548.exchange2_r	648.exchange2_s	Artificial Intelligence: recursive solution generator (Sudoku)	Artificial Intelligence
557.xz_r	657.xz_s	General data compression	Other
503.bwaves_r	603.bwaves_s	Explosion modeling	Modeling and Simulation
507.cactuBSSN_r	607.cactuBSSN_s	Physics: relativity	Modeling and Simulation
508.namd_r		Molecular dynamics	Modeling and Simulation
510.parest_r		Biomedical imaging: optical tomography with finite element	Other
511.povray_r		Ray tracing	Other
519.lbm_r	619.lbm_s	Fluid dynamics	Modeling and Simulation
521.wrf_r	621.wrf_s	Weather forecasting	Modeling and Simulation
526.blender_r		3D rendering and animation	Other
527.cam4_r	627.cam4_s	Atmosphere modeling	Modeling and Simulation
	628.pop2_s	Wide-scale ocean modeling (climate level)	Modeling and Simulation
538.imagick_r	638.imagick_s	Image manipulation	Other
544.nab_r	644.nab_s	Molecular dynamics	Modeling and Simulation
549.fotonik3d_r	649.fotonik3d_s	Computational Electromagnetics	Modeling and Simulation
554.roms_r	654.roms_s	Regional ocean modeling	Modeling and Simulation

5.2 Measurement Setup

We chose the HPE ProLiant DL385 Gen10 server as representative for cloud computing for this experiment. The SUT is equipped with two AMD EPYC 7702 CPUs. Each is configured with a 2.00 GHz base and 3.35 GHz maximum frequency, 64 cores, and 128 threads, thus resulting in 128 cores and 256 threads. The memory setup is 1 TB, organized in 16 modules with 64 GB each.

The operating system is a SUSE Linux Enterprise Server 15 SP1 with kernel version 4.12.14-195-default. Benchmark binaries are generated on an AMD-optimized compiler. The AOCC 2.0.0 compiler, based on LLVM, is used throughout our analysis. The full benchmark results including optimization flags that we are using are available online [Sch+20b], together with additional information on the SUT and test environment. We adapt the aggressive optimizations selected by a performance expert for an official benchmark run to be closer to real-world scenarios. The compiler flags we used for our comparison of the SPECrate suite runs are shown in Table 5.1. We skip the compiler flags for the SPECspeed suites for brevity. They differ only by using OpenMP with the following flags added: `-DSPEC_OPENMP -fopenmp -fopenmp=libomp -lomp -lpthread -ldl`. According to the SPEC CPU 2017 run and reporting rules, each benchmark was executed three times, a number decided on to further the comparability and fairness of the benchmark, in *base* configuration for an optimized and real-world scenario.

All measurements were taken according to the run and reporting rules of the SPEC CPU 2017 benchmark suite [Hen19b] under controlled conditions at a constant 21°C intake air temperature with 56% humidity that is within operational limits of the SUT. The power analyzer samples with approximately 10 kHz and reports the average to the controller that verifies less than 5% of all samples have an uncertainty greater than 1% and less than 1% of all samples have an unknown uncertainty.

5.3 Case Study Results

To find the impact of compiler optimizations on the energy efficiency and performance of applications, we compare two different scenarios, one with optimized compiler settings, selected by a performance expert, and a second one with settings that resemble a real-world scenario. In our analysis, we first take a look at the performance and energy-efficiency impact of compiler optimizations, followed by an examination of the power consumption over time.

As a last step we check if certain application domains are more susceptible to energy efficiency improvements.

5.3.1 Performance and Energy Efficiency

We compare the performance and energy efficiency results of the *base* runs from the SPECrate and SPECspeed benchmark suites for our SUT, described in Section 5.2. First, we compare the energy efficiency against the performance to show the difference between them, using the *base* runs of the SUT. The time-to-result is plotted as it is the performance metric for the SPECspeed suites. For the SPECrate suites, the runtime and the number of copies to execute, 256, gives us the throughput for the suites benchmarks.

Figures 5.1, 5.2, 5.3, and 5.4 show the resulting performance and energy-efficiency metrics. Tables 5.3 and 5.4 show the difference (optimized minus real-world scenario) in performance and energy-efficiency metrics. Keep in mind that for the SPECrate suites, a higher throughput value is better, while for the SPECspeed suites, a lower value is better.

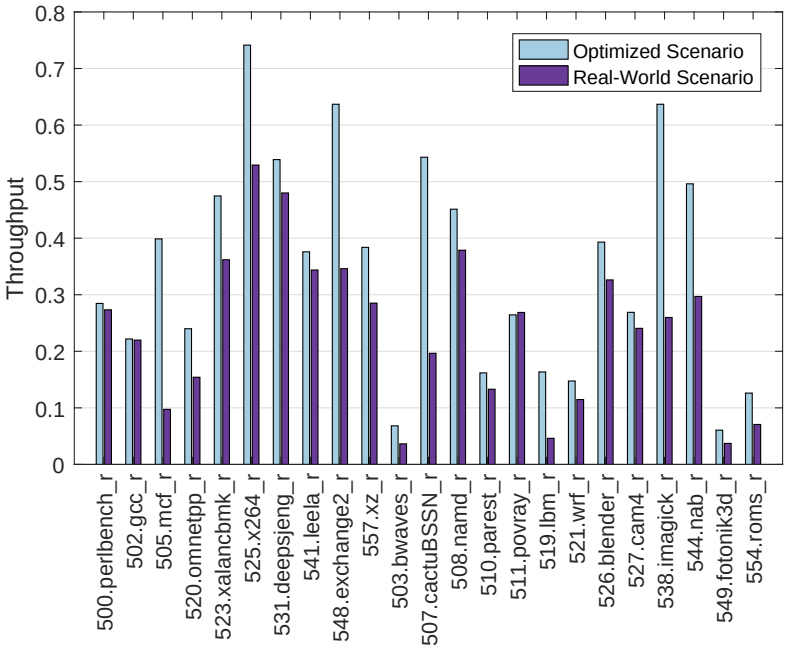


Figure 5.1: Time-to-result for the SPECrate suites base runs for both scenarios on the ProLiant DL385 Gen10 server.

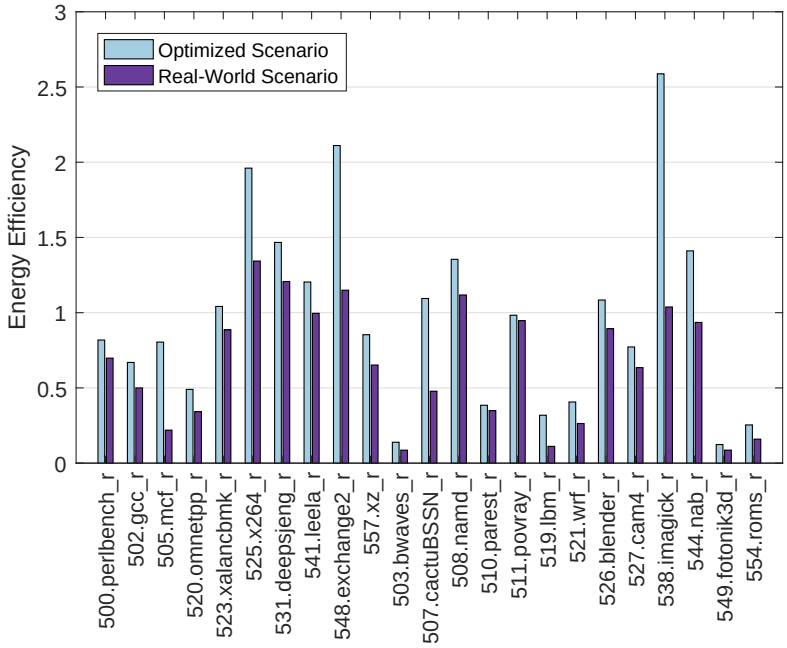


Figure 5.2: Energy efficiency for the SPECrate suites base runs for both scenarios on the ProLiant DL385 Gen10 server.

As visible in Figure 5.1 and Table 5.3, throughput is consistently higher in the optimized scenario. The only exception is the 511.pov-ray_r benchmark which shows a small decrease in throughput for the optimized scenario while the energy efficiency increases slightly. With the 511.povray_r benchmark, the only one showing a decrease in performance, is what we would assume an outlier for the SPECrate benchmarks. The mean performance increased by 30.7% for the SPECrate suite benchmarks while the energy efficiency increased by 31.1% when compiling for the optimized scenario instead of the real-world scenario. These values should be taken with care as not all benchmarks tend to increase performance and energy efficiency at the same time. For example the 502.gcc_r increased performance by only 0.9% but energy efficiency by 25.3%.

While the SPECspeed suites show a similar picture in Figure 5.4 and Table 5.4, more benchmarks seem to deviate from the general rule that the optimized scenario increases performance. Namely the 602.gcc_s, 623.xalancbmk_s, 631.deepsjeng_s, and 644.nab_s benchmarks. While the 602, 623 and 631 benchmarks from the SPECspeed Integer suite decrease performance by only a small margin in the real-world scenario, the 644.nab_s has a reduced performance of 39.2%. At the same time, a large amount of benchmarks from the SPECspeed

suite increased performance by a large margin, ranging from a reduction in time-to-result from -2.8% up to -2847.5%. While the SPECrate suites show that the optimized scenario leads to better throughput and energy efficiency, the SPECspeed suites show a more diverse picture and a reduction in time-to-result (increase in performance) does not always increase energy efficiency and if it increases energy efficiency, it does so at a different rate. For example, the 627.cam4_s tremendously increase performance but energy efficiency goes down by -26.2%. The 603.bwaves_s benchmark also increases performance (-1466.1% in time-to-result), however likewise manages to raise energy efficiency by 42.3%. Small differences, around 3%, are still viable as the SPEC CPU 2017 benchmark suite is designed for low run to run variation.

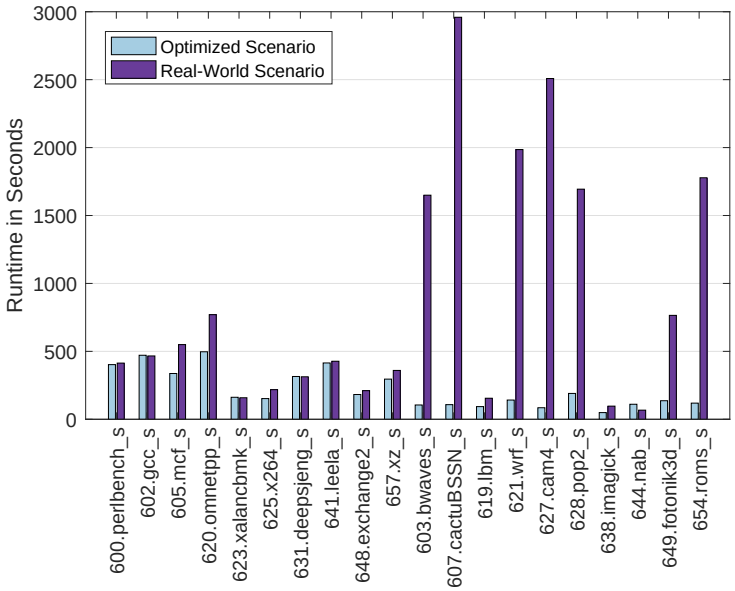


Figure 5.3: Time-to-result for the SPECspeed suites base runs for both scenarios on the ProLiant DL385 Gen10 server.

In conclusion, we can say that energy efficiency can be different from performance, if we look at throughput as well as time-to-result. The SPEC CPU 2017 contains some benchmarks in its SPECspeed Floating-Point suite that react very dynamically to changes in compiler optimizations. Yet, not all improvements in performance influence energy efficiency in the same manner but they can overlap. This is consistent with the findings of Capra et al. [CFS12], where two functionally identical but different ERP software products had a performance difference of 5%, but a difference in energy efficiency of 50%.

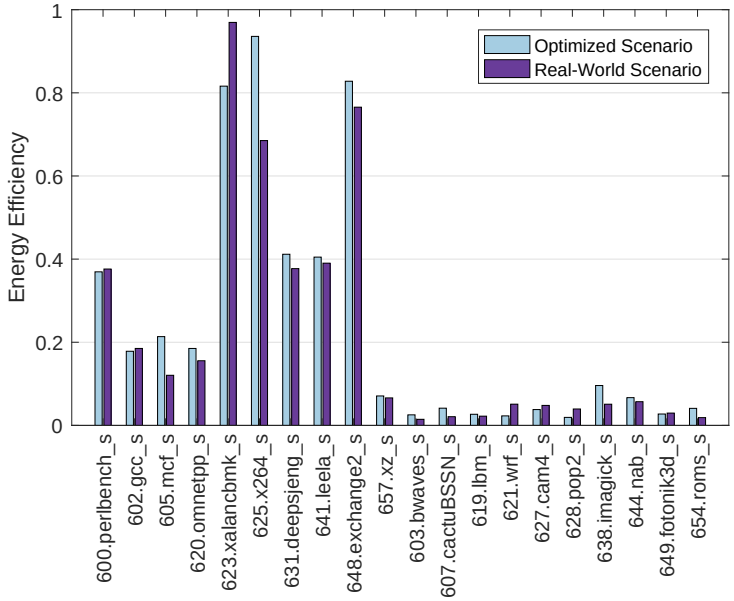


Figure 5.4: Energy efficiency for the SPECspeed suites base runs for both scenarios on the ProLiant DL385 Gen10 server.

Table 5.3: Differences in throughput and energy efficiency for the SPECrate suites.

Benchmark	Difference			
	Throughput		Energy Efficiency	
500.perlbench_r	0.011	4.0%	0.120	14.7%
502.gcc_r	0.002	0.9%	0.169	25.3%
505.mcf_r	0.301	75.6%	0.585	72.7%
520.omnetpp_r	0.086	35.8%	0.148	30.3%
523.xalancbmk_r	0.113	23.8%	0.155	14.9%
525.x264_r	0.212	28.6%	0.617	31.5%
531.deepsjeng_r	0.059	10.9%	0.260	17.7%
541.leela_r	0.032	8.5%	0.208	17.3%
548.exchange2_r	0.291	45.7%	0.961	45.5%
557.xz_r	0.099	25.7%	0.201	23.6%
503.bwaves_r	0.032	46.7%	0.053	37.9%
507.cactuBSSN_r	0.347	63.8%	0.616	56.3%
508.namd_r	0.073	16.1%	0.237	17.5%
510.parest_r	0.029	17.9%	0.036	9.4%
511.povray_r	−0.004	−1.6%	0.036	3.6%
519.lbm_r	0.117	71.7%	0.207	65.0%
521.wrf_r	0.033	22.2%	0.143	35.2%
526.blender_r	0.067	17.0%	0.191	17.6%
527.cam4_r	0.028	10.5%	0.138	17.8%
538.imagick_r	0.377	59.2%	1.549	59.9%
544.nab_r	0.199	40.1%	0.475	33.7%
549.fotonik3d_r	0.023	38.6%	0.037	30.0%
554.roms_r	0.055	44.0%	0.094	37.2%
Mean	0.112	30.7%	0.315	31.1%

Table 5.4: Differences in time-to-result and energy efficiency for the SPECspeed suites.

Benchmark	Difference			
	Time-to-Result		Energy Efficiency	
600.perlbench_s	−11.3	−2.8%	−0.007	−1.8%
602.gcc_s	5.0	1.1%	−0.007	−3.8%
605.mcf_s	−212.7	−63.1%	0.093	43.6%
620.omnetpp_s	−273.0	−54.9%	0.030	15.9%
623.xalancbmk_s	3.3	2.1%	−0.153	−18.8%
625.x264_s	−65.7	−43.1%	0.251	26.8%
631.deepsjeng_s	2.0	0.6%	0.035	8.4%
641.leela_s	−12.7	−3.1%	0.015	3.6%
648.exchange2_s	−28.3	−15.5%	0.062	7.5%
657.xz_s	−64.0	−21.6%	0.005	6.6%
603.bwaves_s	−1544.3	−1466.1%	0.011	42.3%
607.cactuBSSN_s	−2851.7	−2648.6%	0.020	49.3%
619.lbm_s	−61.7	−66.2%	0.005	16.9%
621.wrf_s	−1843.7	−1301.4%	−0.028	−122.8%
627.cam4_s	−2423.2	−2847.5%	−0.010	−26.2%
628.pop2_s	−1503.7	−790.0%	−0.020	−104.9%
638.imagick_s	−47.7	−97.0%	0.045	46.9%
644.nab_s	43.4	39.2%	0.010	14.5%
649.fotonik3d_s	−628.3	−459.8%	−0.002	−8.2%
654.roms_s	−1659.0	−1394.1%	0.022	54.2%
Mean	−658.9	−561.6%	0.019	2.5%

5.3.2 Power Consumption

In Figure 5.5 we can see the power consumption of the SPECrate Integer suite during all three benchmark runs. The SPECrate Floating Point produced similar results. A general tendency shown in the figure is that the real-world scenario (white boxes) seems to reduce variability in power consumption compared to the optimized scenario (gray boxes). For example, the 502.gcc_r and 521.wrf_r show a more focused behavior. There are benchmarks for which this general rule does not apply, such as 523.xalancbmk_r. Outliers, shown as gray crosses, are most likely from the state change during startup and teardown of the benchmark until a more steady state is reached. Our assumption is supported by the fact that most outliers are below the mean value indicating a ramp-up and ramp-down behavior. Yet, some benchmarks seem to have a higher dispersion in terms of consumed power during a benchmark run.

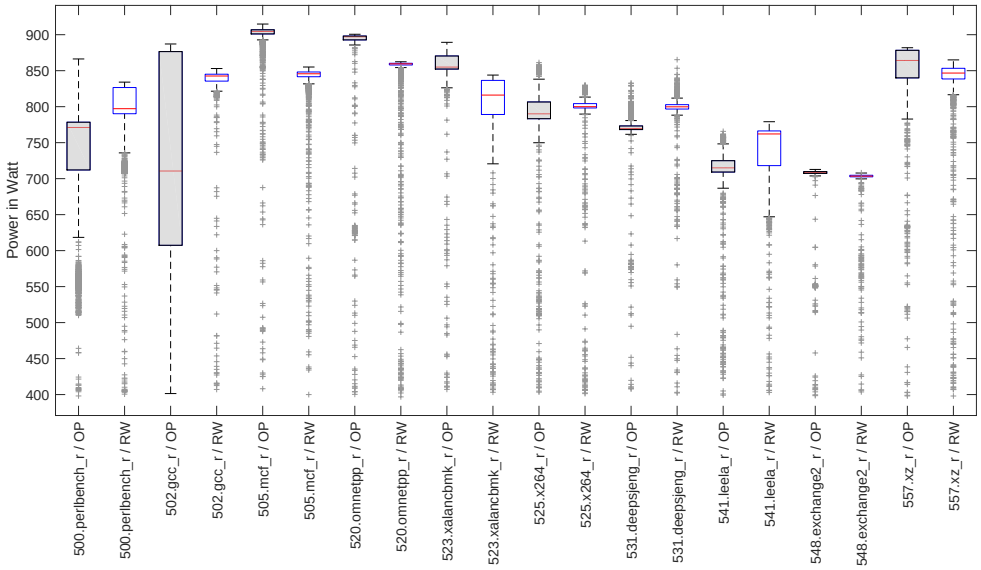


Figure 5.5: Power consumption of the SPECrate Integer suite base run for the optimized (left, gray boxes) and real-world scenario (right, white boxes) of the ProLiant DL385 Gen10 server.

The variance for each benchmark of the SPECrate suites runs is shown in Table 5.5. For the SPECrate Integer suite, we can see a trend towards a variance reduction in power consumption for the real-world scenario while the SPECrate Floating Point suite is more diverse. Such a behavior would lead us to suspect that integer workloads are more susceptible to smoothing or stabilizing the

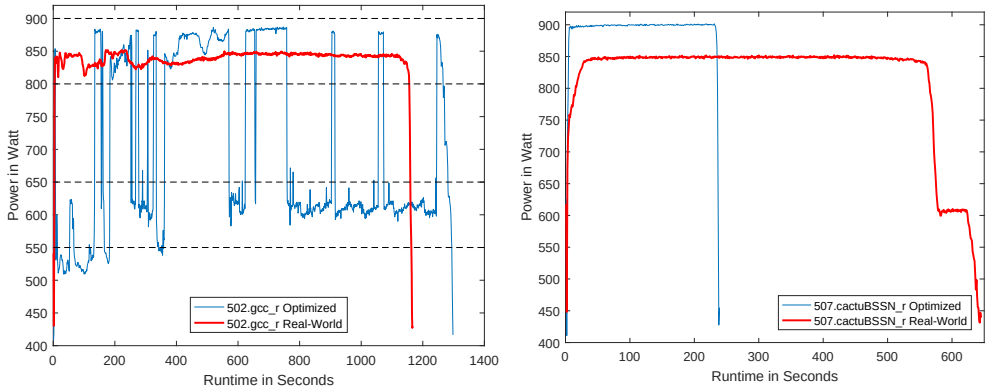
power consumption at a certain level. Another reason could be that the compiler in the optimized scenario does make a trade-off in favor of IPC over memory.

Table 5.5: Variance comparison of power consumption of SPECrate suites base runs for the optimized and real-world scenario on the DL385 Gen10.

Benchmark	Variance		Trend
	Optimized	Real-World	
500.perlbench_r	$9.30 \cdot 10^3$	$2.81 \cdot 10^3$	-
502.gcc_r	$1.97 \cdot 10^4$	$1.11 \cdot 10^3$	-
505.mcf_r	$2.78 \cdot 10^3$	$0.64 \cdot 10^3$	-
520.omnetpp_r	$2.43 \cdot 10^3$	$2.45 \cdot 10^3$	+
523.xalancbmk_r	$2.96 \cdot 10^3$	$3.42 \cdot 10^3$	+
525.x264_r	$7.43 \cdot 10^3$	$3.98 \cdot 10^3$	-
531.deepsjeng_r	$2.08 \cdot 10^3$	$1.64 \cdot 10^3$	-
541.leela_r	$2.15 \cdot 10^3$	$1.91 \cdot 10^3$	-
548.exchange2_r	$1.76 \cdot 10^3$	$0.86 \cdot 10^3$	-
557.xz_r	$3.42 \cdot 10^3$	$3.41 \cdot 10^3$	-
503.bwaves_r	$0.92 \cdot 10^3$	$0.92 \cdot 10^3$	/
507.cactuBSSN_r	$2.39 \cdot 10^3$	$1.06 \cdot 10^4$	+
508.namd_r	$6.21 \cdot 10^3$	$7.01 \cdot 10^3$	+
510.parest_r	$2.90 \cdot 10^3$	$7.26 \cdot 10^3$	+
511.povray_r	$5.92 \cdot 10^3$	$2.29 \cdot 10^3$	-
519.lbm_r	$1.20 \cdot 10^3$	$1.06 \cdot 10^3$	-
521.wrf_r	$1.95 \cdot 10^4$	$6.09 \cdot 10^3$	-
526.blender_r	$3.25 \cdot 10^3$	$4.64 \cdot 10^3$	+
527.cam4_r	$1.74 \cdot 10^4$	$1.01 \cdot 10^4$	-
538.imagick_r	$9.07 \cdot 10^3$	$7.00 \cdot 10^3$	-
544.nab_r	$3.00 \cdot 10^3$	$1.00 \cdot 10^4$	+
549.fotonik3d_r	$0.78 \cdot 10^3$	$1.16 \cdot 10^3$	+
554.roms_r	$1.60 \cdot 10^3$	$2.18 \cdot 10^3$	+

To support this claim, we take a closer look at the two benchmarks with the highest difference in variance, the 502.gcc_r and 507.cactuBSSN_r benchmarks. Their power consumption over time can be seen in Figure 5.6a and Figure 5.6b. It is visible that 502.gcc_r in the optimized scenario has drastic changes at specific points in time. The benchmark seems to have two states, a low power state around 550-650 Watts and a high power state between 800 and 900 Watts, depicted by the black dashed lines. The less optimized real-world scenario has

even higher performance with a throughput for this specific run compared to the optimized scenario. On average though, we would assume that the optimized and real-world scenarios are close in terms of performance as shown in Table 5.3. For comparison, 507.cactuBSSN_r shows a very stable power consumption pattern with a distinct drop for the real-world scenario towards the end of the measurement. Also visible is that in the real-world scenario, the power does not reach the same level, around 900 Watt for the optimized, and 850 Watts for the real-world scenario. This behavior reinforces our claim that the compiler optimizations meant for performance improvements can influence the power consumption. Yet, despite the large discrepancy in variance, the 507.cactuBSSN_r benchmark exhibits less fluctuation in power consumption. The difference in variance most likely stems from the longer runtime and lower peak power consumption in the real-world scenario.



(**a**) 502.gcc_r (**b**) 507.cactuBSSN_r

Figure 5.6: Power consumption of the 502.gcc_r and 507.cactuBSSN_r benchmark from the SPECrate Integer and SPECrate Floating Point suite base run on the DL385 Gen10 server. The first run out of three is plotted.

5.3.3 Application Domain and Implementation Language

Finally, we take a look at the benchmarks that display an improvement in energy efficiency and their implementation language. From 23 benchmarks of the SPECrate suite that ran, 7 are excluded due to reporting *peak* as *base* values. From the remaining 16 benchmarks, 12 achieved a higher energy efficiency in the *peak* runs, listed in Table 5.6. Of those 12, only one is written in Fortran. We, therefore, suspect that C or C-like languages can be optimized better. We ran

Fisher's exact test (see Table 5.7) with the null hypothesis H_0 that both C-like and functional languages are equally likely to show better energy efficiency. We selected Fisher's exact test in favor of the χ^2 test due to the small number of observations. We must reject H_0 in favor of the alternative hypothesis at the 5% significance level with $p = 1.57 \cdot 10^{-2}$. H_0 cannot be rejected at the 1% level.

This outcome could have three reasons: 1) it is due to the compiler that allows fewer optimizations for Fortran programs, 2) it is the functional programming paradigm that provides an already energy-efficient programming style, or 3) the results are outliers. In all cases, additional observations should be made in the future to verify or reject our findings that the C-like languages are more susceptible to compiler optimizations.

Table 5.6: Implementation language and percentage of efficient benchmarks. Three benchmarks are implemented in two languages and are counting towards each implementation language.

Implementation Language	Benchmarks More Efficient	Total	Percentage
C	8	8	100%
C++	6	7	85.7%
Fortran	1	4	25.0%

Table 5.7: Fisher's exact test contingency table.

	Improved *eff*		
Language	Yes	No	Sum
C-like	14	1	15
Functional	1	3	4
Sum	15	4	19

The last step is the application area from which the benchmarks are taken. As mentioned in Section 5.1, we group each benchmark in one of four categories. Table 5.8 lists the percentage of how many benchmarks that reported a higher energy efficiency in *peak* came from which application area. The results suggest that modeling and simulation workloads cannot be optimized well by

a compiler. Yet, it must be taken into consideration that the application areas are not equally distributed.

Table 5.8: Application domain and percentage of efficient benchmarks.

Application Domain	Benchmarks		
	More Efficient	Total	Percentage
Language Transformation	2	2	100%
Modeling and Simulation	3	7	42.8%
Artificial Intelligence	1	1	100%
Other	6	6	100%

5.4 Concluding Remarks

Conclusively, it can be seen that compiler optimizations have a significant impact on energy efficiency and power consumption, thereby answering our research question *RQ A.1*. However, not all benchmarks in the SPEC CPU 2017 benchmark suite react similarly. For example, the 502.gcc_r benchmark can only achieve a 0.9% increase in throughput while its energy efficiency can be increased by 25.3% and the 523.xalancbmk_r reaches a 23.8% better throughput but only 14.9% higher energy efficiency. The majority of benchmarks, though, behave synchronously to performance. This result gives a good incentive to developers to select their compiler optimizations with care and test out different configurations if their application does not respond in an expected way to performance increases by compiler optimizations. These results could also incentivize additional tooling that, given the source code of an application, can choose the best optimizations options for the developer. Additionally, a general decrease in variability can be observed in the power consumption of the SPECrate suite, with some outliers that might help with power budgeting.

Given that the results show a general rule that higher performance correlates to better energy efficiency with some outliers, we look at the application domain and programming language. The impact of the application domain needs additional measurements to be able to reach a clear verdict. The relation of implementation language to energy efficiency already shows promising results towards improving the energy efficiency for C-like languages, but we also recommend additional measurements.

Chapter 6

Measuring the Impact of Common Sorting Algorithms

Compiler optimizations are not the only way to increase the energy efficiency of software but there is, as mentioned in Chapter 5, a knowledge gap. Closing this knowledge gap is essential. One important aspect is selecting a good or even the best algorithm for a given task. One task that is part of most programs is sorting data. Sorting algorithms are well known, and some research exists focusing on conserving energy by looking at those algorithms [RAT15; Bun+09; CPK13]. However, they put their focus either on battery-powered mobile devices [RAT15; Bun+09] or on comparing different data types (floating-point versus integer) of a small problem size on desktop computers [CPK13].

The contribution of this work is the analysis of the energy efficiency of six common sorting algorithms. We use server systems representative for cloud computing, two different programming languages, and two different implementation variants for Python and C to analyze which or if an approach is more energy-consuming than another. We also compiled a list of guidelines for energy efficient sorting. This helps industry practitioners in software development to utilize our results to make informed decisions about the most suitable sorting algorithms for their problem at hand. At the same time, this contribution raises awareness that algorithm selection can have a large impact on energy efficiency, addressing *RQ A.2* of *Goal A*. This section is based on our publication in the Proceedings of the 29th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS) 2021 [Sch+21a].

We first describe our SUT and setup, followed by the measurement analysis in which we compare the sorting algorithms. In a second step we take a look at two different representative server systems to analyze if the energy efficiency is transferable or is an attribute of the server.

6.1 Measurement Setup

Measurements are taken on four SUTs described in Section 6.1.1. For each sorting algorithm, two variants in two different programming languages and three problem sizes are used and run on two SUTs, resulting in a total of 144 measurements that are outlined in Section 6.1.2 and 6.1.3 respectively. We measured each experiment five times. The aim is to compare the energy efficiency against each other to determine which implementation approach is preferable under certain circumstances, different servers, implementation variants, and problem sizes.

The energy consumption is calculated by integrating the measured power consumption. For our sampling interval of one second, this is the sum of the power consumption P in Equation 6.1. The energy efficiency, shown in Equation 6.2, is calculated by dividing the problem size p (total number of sorted integers) by the total energy consumption E, resulting in the number of integers sorted by Joule or Watt second. For the number of bytes sorted by Joule, we multiply p by four, the size of an integer on the SUTs. We count a variant as preferable if the 95% confidence intervals do not overlap.

$$E = \sum P \tag{6.1}$$

$$Eff = \frac{p * 4}{E} \tag{6.2}$$

6.1.1 System Under Test and Setup

As our SUTs, we selected four different representative HPE ProLiant servers listed in Table 6.1, each with a different CPU size. We do not list the CPU manufacturer to avoid giving the impression of bias towards a manufacturer. All servers are equipped with the same $480\,\mathrm{GB}$ SSD storage drive. CentOS 8 is used as the operating system. These servers are, in our opinion, a good representation of standard cloud servers that are not purpose built and do not contain any additional acceleration units. Each server's power supply is connected to a Yokogawa WT210 power analyzer for power measurements. The power analyzer samples internally at about 10kHz and aggregates each sample over one second. The setup is in accordance with the power and performance benchmarking methodology described by Kounev et al. [KLK20]. For our analysis, we do not use the internal sampling rate but the aggregated sampling rate of one second.

Table 6.1: Systems under test.

Name	Cores/Threads	Clock	Memory
Server A	128/256	2.25GHz	16x16GB
Server B	56/112	2.20GHz	12x16GB
Server C	32/64	3.00GHz	16x16GB
Server D	36/72	2.60GHz	12x16GB

6.1.2 Selected Sorting Algorithms

We have selected six commonly known sorting algorithms, listed in Table 6.2. The algorithms were chosen as they have different runtime behaviors, with a range of n^2, n, and $n \log(n)$ for their best, average, and worst case respectively. While Merge and Heap Sort have identical time complexity of $n \log(n)$, they differ in space complexity. Stability of an algorithm has not been considered. We also did not consider hybrid sorting algorithms like Tim Sort or Intro Sort that combine techniques from algorithms already in our list.

Table 6.2: Selected common sorting algorithms.

Name	Time Complexity			Space
	Best	Average	Worst	Complexity
Merge Sort	$n \log(n)$	$n \log(n)$	$n \log(n)$	n
Heap Sort	$n \log(n)$	$n \log(n)$	$n \log(n)$	1
Quick Sort	$n \log(n)$	$n \log(n)$	n^2	$\log(n)$
Insertion Sort	$n \log(n)$	n^2	n^2	1
Bubble Sort	n	n^2	n^2	1
Selection Sort	n^2	n^2	n^2	1

We self-implemented each algorithm in two variants with their differences shown in Table 6.3 to avoid bias of a specific implementation variant, and two programming languages, Python and C. We selected Python and C as they are well known representatives for interpreted and compiled languages[1,2]. For Python, the servers A and B have been used, and for C, the servers C and D.

[1]https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2020. Accessed: May 3rd, 2021

[2]https://www.tiobe.com/tiobe-index/. Accessed: June 18th, 2021

Table 6.3: Implementation variants for Python and C.

	Python	
Algorithm	Variant 1	Variant 2
Merge Sort	Recursive and code inlined	Recursive and code distributed across multiple functions
Heap Sort	Efficient iterative loop to build max heap	Inefficient iterative loop to build max heap
Quick Sort	Iterative	Recursive
Insertion Sort	Iterative	Recursive
Bubble Sort	Iterative	Recursive
Selection Sort	Explicitly type checking in return function to ensure the consistency between the element types going in and out of the sorting function	Dynamic return type (no explicit type checking)
	C	
Algorithm	Variant 1	Variant 2
Merge Sort	Recursive and without dynamic memory allocation	Recursive and with dynamic memory allocation
Heap Sort	Memory based swapping	Pointer based swapping
Quick Sort	Memory based swapping	Pointer based swapping
Insertion Sort	Memory based swapping and no separate function for sorting	Memory based swapping but additional function to perform sort operation
Bubble Sort	Memory based swapping	Pointer based swapping
Selection Sort	Memory based swapping	Pointer based swapping

6.1.3 Problem Size

To stress a SUT, each algorithm had to sort a number of integers, which we refer to as the problem size. For each algorithm, we selected three problem sizes shown in Table 6.4. The problem size is split across the available threads of the SUT listed in Table 6.1. The problem sizes shown in these tables for each algorithm were calibrated separately based on the average time complexity characteristics. Setting common problems sizes across algorithms, servers, and implementation variants proved unfeasible because setting small problem sizes for algorithms with linear time complexity, like Heap Sort, reduces run times and not giving sufficient time for data collection from the power analyzer. Similarly, setting large problem sizes for algorithms with quadratic time complexity, like Bubble Sort, increases the runtime to impractical lengths. The problem sizes used for C and Python are different as we would expect a performance contrast between them. To allow for stable measurements of the C algorithms that are roughly equal in length than the Python measurements, the problem size would have to be larger than the memory and would distort the measurement due to disc I/O. We, therefore, took actions, such as reducing the problem size, to achieve a stable measurement while at the same time keeping disc I/O to a minimum.

The total problem sizes used for the SUTs are kept constant. For example, if we calibrated the Python implementations of Insertion Sort to a total of $10,240,000$ (see Table 6.4), all integers that must be sorted are distributed across the cores equally. If we want to sort $10,240,000$ integers on server A with 256 threads and server B with 112 threads, each thread of server A is given $40,000$ $(10,240,000/256)$ integers and each thread of server B is given $91,428$ $(10,240,000/112)$ integers. Pseudo-random numbers were generated by reading from the `urandom` file for the C implementations and using the `random` library for the Python implementations. All problem sizes fit in the memory of the SUTs.

Table 6.4: Calibrated problem sizes for Python and C.

	Algorithm	Problem Size (# of integers) Small	Medium	Large
Python	Merge Sort	$1,024,000,000$	$1,152,000,000$	$1,280,000,000$
	Heap Sort	$1,280,000,000$	$1,408,000,000$	$1,536,000,000$
	Quick Sort	$1,536,000,000$	$1,664,000,000$	$1,792,000,000$
	Insertion Sort	$10,240,000$	$11,520,000$	$12,800,000$
	Bubble Sort	$10,240,000$	$11,520,000$	$12,800,000$
	Selection Sort	$10,240,000$	$11,520,000$	$12,800,000$
C	Merge Sort	$5,760,000,000$	$6,120,000,000$	$6,480,000,000$
	Heap Sort	$5,760,000,000$	$6,120,000,000$	$6,480,000,000$
	Quick Sort	$5,760,000,000$	$6,120,000,000$	$6,480,000,000$
	Insertion Sort	$3,600,000$	$7,200,000$	$10,800,000$
	Bubble Sort	$3,600,000$	$7,200,000$	$10,800,000$
	Selection Sort	$3,600,000$	$7,200,000$	$10,800,000$

6.2 Impact of Algorithms on Energy Efficiency

In this Section, we first analyze the results of our measurements. Our focus in the analysis is on the three sorting algorithms with a time complexity of $n * \log(n)$ but present the results for the n^2 algorithms as well. We then derive recommendations or guidelines for developers and discuss threats to validity.

6.2.1 Analysis

First, we will take a look at the algorithms that do not have a logarithmic runtime behavior. It is clear from the results in Table 6.5 and 6.6 that the three sorting algorithms that have an average time complexity of n^2, Bubble Sort, Insertion Sort, and Selection Sort, are much less energy efficient than the algorithms with $n * \log(n)$ as expected no matter the implementation variant.

Regarding the three algorithms with $n * \log(n)$ time complexity, Heap Sort, Merge Sort, and Quick Sort, the Heap Sort algorithm is outperformed across almost all implementation variants, problem sizes, and SUTs (see Table 6.5). The only exception is Quick Sort on server A, variant 2 on a small problem size with $62\,\mathrm{MB\,J^{-1}}$ compared to $64\,\mathrm{MB\,J^{-1}}$ for Heap Sort. We can also determine that even an inefficient implementation to generate the max heap for Heap Sort is not the main reason for this behavior as variant 1 and 2 achieve similar energy

efficiency values on all problem sizes and both SUTs. Merge Sort maintains a stable energy efficiency across the problem sizes for each server. One exception is variant 2 on server A with a medium problem size close to $62\,\mathrm{MB\,J^{-1}}$ but still in the same range of Quick Sort and outperforming Heap Sort given this configuration. For Heap Sort on server A with a higher thread count, inlined code (variant 1) has higher efficiency. At the same time, not inlining and calling additional functions seems beneficial for servers with smaller core counts. Quick Sort scales well on server A and becomes more energy-efficient the larger the problem size grows. Although, on server B, it seems to plateau as the configurations for the problem sizes are similar by the 95% confidence interval. In general, Quick Sort also has a higher energy efficiency than Merge Sort for server B.

Essentially, better energy efficiency for a larger thread count can be observed between both SUTs for Python. These results are expected. We assume that this stems from the measurement setup, measuring wall power at the power supply for the complete server. As server A has a higher thread count but, apart from memory, similar hardware, the base power consumption is similar, resulting in a lower energy efficiency score. This lower score is neither attributable to the CPU or the algorithm.

For the C implementations, shown in Table 6.6, we take a look at only the three algorithms with a time complexity of $n * \log(n)$. It can be observed that Merge Sort outperforms all other sorting algorithms by a large margin if the memory is not dynamically allocated in variant 1. Nevertheless, if the memory is dynamically allocated with variant 2, its energy efficiency is reduced by a factor ranging from 7.3 on the medium problem size and server C up to 13.4 on the large problem size and server D.

Quick Sort is the next most energy-efficient implementation in C and should be preferred over a Merge Sort with dynamically allocated memory. Both variants of Quick Sort are more energy efficient than variant 2 of Merge Sort. Yet, in all but one case (medium problem size on server C), Quick Sort becomes less energy efficient when it is implemented with pointer based swapping. This single case for Quick Sort is probably due to the large confidence intervals and we assume that additional measurement runs would resolve this outlier.

Regarding Heap Sort, the difference between pointer and memory based swapping is not as clear. There are two measurements on server C, the small and large problem size, that do overlap on the 95% confidence interval. Given that Quick Sort has a better space complexity of $\log(n)$ compared to Heap Sort with n, it is expected that Heap Sort is less susceptible to changes on how the swapping in memory is handled.

Table 6.5: Energy efficiency for the Python implementations in sorted $\mathrm{kB\,J^{-1}}$.

	Server	Problem Size	Variant 1 Mean	Variant 1 95% CI	Variant 2 Mean	Variant 2 95% CI	V1 and V2 Overlap
Merge Sort	A	Small	95 827.81	2219.27	89 306.57	320.12	✗
		Medium	93 869.7	702.76	61 848.02	1050.81	✗
		Large	93 251.27	531.87	86 783.91	305.66	✗
	B	Small	42 599.25	255.33	45 074.79	177.47	✗
		Medium	42 401.23	147.29	44 907.01	178.77	✗
		Large	42 149.87	162.97	45 016.84	200.9	✗
Heap Sort	A	Small	64 238.06	1888.36	64 072.1	2055.87	✓
		Medium	49 276.52	3061.87	49 179.08	3273.6	✓
		Large	50 795.02	3537.18	49 664.16	3817.94	✓
	B	Small	31 546.17	70.99	31 561.86	46.96	✓
		Medium	31 514.58	77.78	31 447.73	86.82	✓
		Large	31 350.52	59.84	31 331.65	52.29	✓
Quick Sort	A	Small	85 200.41	500.45	61 514.99	833.26	✗
		Medium	87 686.18	395.88	62 400.97	1038.0	✗
		Large	119 615.0	670.43	77 710.85	255.19	✗
	B	Small	72 838.23	514.72	49 790.54	361.29	✗
		Medium	72 994.33	715.83	49 872.73	338.38	✗
		Large	73 519.13	707.49	49 995.9	310.38	✗
Insertion Sort	A	Small	585.35	2.82	42.03	0.12	✗
		Medium	400.73	3.4	35.41	1.6	✗
		Large	447.42	3.99	31.76	0.06	✗
	B	Small	102.05	0.32	2120.12	75.03	✗
		Medium	90.15	0.17	7.64	0.92	✗
		Large	81.16	0.06	6.63	0.95	✗
Bubble Sort	A	Small	263.49	17.22	274.77	10.95	✓
		Medium	186.99	22.5	184.75	21.4	✓
		Large	173.83	15.49	175.83	15.11	✓
	B	Small	67.61	58.0	192.27	0.64	✗
		Medium	41.34	0.1	42.13	0.17	✗
		Large	37.09	0.08	32.51	14.55	✓
Selection Sort	A	Small	414.76	48.77	429.13	45.98	✓
		Medium	394.48	2.23	390.1	4.08	✓
		Large	410.4	17.12	414.7	11.83	✓
	B	Small	100.39	0.26	102.08	0.36	✗
		Medium	88.82	0.36	90.46	0.22	✗
		Large	79.44	0.38	81.35	0.12	✗

Table 6.6: Energy efficiency for the C implementations in sorted $\mathrm{kB\,J^{-1}}$.

	Server	Problem Size	Variant 1 Mean	Variant 1 95% CI	Variant 2 Mean	Variant 2 95% CI	V1 and V2 Overlap
Merge Sort	C	Small	3 057 140.43	68 234.82	401 219.46	1867.08	✗
		Medium	2 874 856.35	607 613.69	393 527.45	20 993.1	✗
		Large	3 406 575.21	109 261.19	376 680.4	41 104.92	✗
	D	Small	3 554 848.68	702 507.71	322 805.22	1190.23	✗
		Medium	3 946 797.65	572 690.29	318 950.61	3342.17	✗
		Large	4 263 224.3	261 687.77	317 969.18	3109.83	✗
Heap Sort	C	Small	156 507.34	9793.82	156 538.77	2647.3	✓
		Medium	159 005.12	491.7	156 300.54	1164.83	✗
		Large	156 701.09	2314.56	153 358.12	4113.29	✓
	D	Small	112 808.83	165.06	107 845.36	110.66	✗
		Medium	112 207.69	350.81	107 072.05	135.36	✗
		Large	111 542.68	403.48	106 421.56	474.03	✗
Quick Sort	C	Small	712 085.62	2818.19	581 176.35	49 584.84	✗
		Medium	687 025.63	80 615.43	583 890.48	44 280.08	✓
		Large	689 021.31	74 343.39	604 550.75	6370.04	✗
	D	Small	539 591.67	6980.36	424 654.5	5686.78	✗
		Medium	544 565.18	7796.15	420 779.36	3926.93	✗
		Large	545 010.71	5673.11	424 777.04	3415.2	✗
Insertion Sort	C	Small	1571.49	23.95	1299.39	13.67	✗
		Medium	1136.63	11.74	851.6	5.85	✗
		Large	819.89	5.01	600.31	2.2	✗
	D	Small	1565.2	108.45	1328.51	29.46	✗
		Medium	1017.02	24.45	829.03	11.5	✗
		Large	711.04	11.77	578.79	6.98	✗
Bubble Sort	C	Small	464.87	2.79	447.34	2.23	✗
		Medium	249.44	1.82	237.7	0.37	✗
		Large	165.24	1.01	157.21	0.72	✗
	D	Small	368.83	2.2	349.39	2.87	✗
		Medium	192.95	0.88	181.93	1.16	✗
		Large	127.86	0.52	121.8	1.31	✗
Selection Sort	C	Small	571.07	3.53	901.45	5.29	✗
		Medium	314.75	1.36	537.04	3.14	✗
		Large	211.29	1.31	369.8	0.79	✗
	D	Small	497.21	2.6	837.31	23.62	✗
		Medium	264.64	1.39	477.16	4.57	✗
		Large	176.83	1.09	320.51	1.09	✗

6.2.2 Guidelines for Developers

We took the findings from our measurements in Section 6.2.1 before and provide some basic guidelines for practitioners for each sorting algorithm. We left out Bubble Sort, Insertion Sort, and Selection Sort on purpose as a much higher energy efficiency can be achieved and we would recommend, given the choice, implementing a different algorithm with a better time complexity. We also found that there are major differences even for algorithms of similar time complexity. We marked in front of each guideline whether it is applicable to both implementation languages (*Both*), only for *Python*, or only for *C*.

Merge Sort

Both Merge Sort is preferable over a Heap or Quick Sort algorithm when memory is no constraint.

Python Use Merge Sort for up to 1.2 billion integers (medium problem size).

Python In case of a very large amount of sorting data, a Quick Sort algorithm might be a better choice.

C Preallocate memory if possible. If the memory must be allocated dynamically, a Quick Sort implementation should be preferred.

Heap Sort

Both Implement a Quick Sort algorithm instead when possible and the space complexity is no hard constraint.

Python Max heap generation can be neglected in terms of energy efficiency.

Python Does not scale as good with the growing size of the sorting problem.

Quick Sort

Python Implement iterative variant if possible to achieve a better energy efficiency.

Python It scales better with the growing size of the sorting problem.

C Pre-allocate memory if possible.

6.2.3 Threats to Validity

We will discuss the threats to validity in this section. The threats are discussed in the order of severity rated by the authors.

Lower number of repetitions All measurements were repeated five times. However, a higher number of repetitions might reduce the variance and, therefore, some confidence intervals might not overlap, which will negate the conclusions drawn from the measurements. As this work is intended as a guideline, a selection of a good algorithm instead of the optimal still can be of value.

Selection of sorting algorithms The selection of algorithms to choose for this work has been based on the time and space complexity of the algorithms. We are aware that alternatives with similar time and space complexity might exist and argue that, given the assumption that they were implemented with care and perform as intended, they should be interchangeable. Stability, on the other hand, was not considered and the results might not be generalizable to algorithms with different stability properties.

Programming language selection We selected the programming languages Python and C as representatives for an interpreted and compiled language, respectively. Both languages are publicly available for extended periods of time, well-known, and widely used. We are aware that the compilation can introduce optimizations but argue that this is not a threat as compiler optimizations, ahead-of-time or just-in-time, will be used and can even introduce further improvements in energy efficiency.

Problem size selection We took care to select three different problem sizes for each algorithm that allowed for a stable measurement while keeping the measurement feasible timewise. However, due to the nature of the time complexity of slow algorithms, not all algorithms have been calibrated to identical problem sizes. This threat might invalidate our conclusions when absolute energy consumption results are compared but not for the normalized energy efficiency.

Limited number of server configurations As measuring takes time and the number of possible server configurations is vast, considering all configurations becomes practically infeasible. We, therefore, opted for four server configurations representative of a cloud data center scenario. As the four options consider both major x86 CPU manufacturers, we see this as a good starting point for this analysis, although, it is possible that servers with a lower computational power might yield different results. Other instruction set architectures have not been considered as, in our opinion, cloud hardware predominantly uses x86 servers, so our generalizability is limited to x86 only.

6.3 Concluding Remarks

We answered our research question *RQ A.2* of *Goal A* by analyzing and comparing six common sorting algorithms. We were able to show that, even with algorithms of the same time complexity, there can be large differences in energy efficiency. For Python, for example, the energy efficiency of commonly used sorting algorithms ranges from $49\,\mathrm{MB\,J^{-1}}$ to $120\,\mathrm{MB\,J^{-1}}$ on server A. In the compiled language C, Merge Sort in our implementation variant one outperforms all other implementations by a large margin with over $4\,\mathrm{GB\,J^{-1}}$. We further provide short guidelines that help industry practitioners in writing and propagate awareness for energy-efficient software.

Part III

Resource Profile Classification

Chapter 7

Overview

Applications can change rapidly in a DevOps environment due to rapidly changing requirements [Bru+15]. This problem of rapid code changes, identified in Section 1.2, leads to inaccurate power models for an application as they are trained for that application. Application-level monitoring approaches also need constant adaptations to deliver accurate information. Additionally, cloud data centers are, to our knowledge, not equipped with specialized power measuring equipment that can measure the power consumption of a single application. We, therefore, propose an automated classification approach for power modeling. Our approach automatically characterizes an application's workload profile based on system-level performance events to choose a suitable and more accurate power model for the current version of the application. Yet, a workload profile of an application can change during runtime as different tasks are performed. To accommodate this behavior in the approach, it can select power models not only per application, but also for execution phases of an application. The advantages of our contribution, addressing *RQ B.1* of *Goal B*, are twofold:

1. No instrumentation of the application or operating system is necessary.
2. No specialized equipment in the form of power analyzers are necessary.

Our approach, shown in Figure 7.1, depicts how our approach works. First, we run a large variety of benchmarks from which we can gather the power consumption, resource utilization, and performance events in the data acquisition phase, described in Section 8.1 and Section 8.2. As benchmarks, we use the open-source Phoronix Test Suite with 200 benchmarks [Sui21] and stress-ng with 220 benchmarks [Kin19], representing different applications with their individual workload profile. We also test our approach on a real-world application without retraining, namely Elasticsearch [GT15]. This real-world application allows us to show that our approach is transferable, as we expect for machine learning classifiers, to answer part of research question *RQ B.4* or

our *Goal B*. It also shows that our approach can classify execution phases of an application and increase the accuracy of the power consumption prediction.

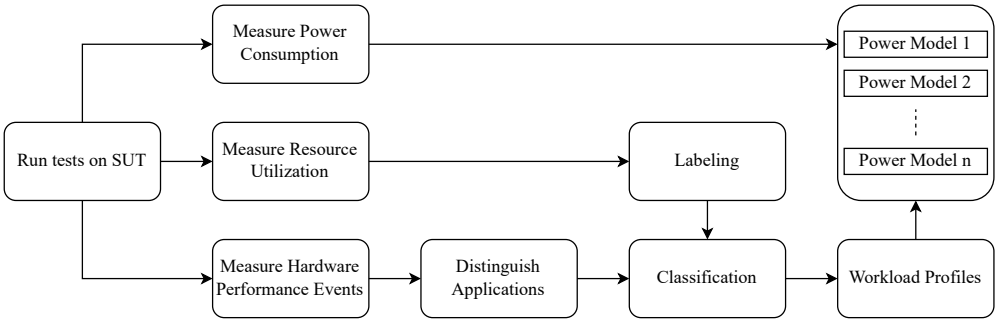


Figure 7.1: Overview of the application classification approach.

After measuring the benchmarks, we analyze the data to determine the workload profiles in Section 8.3. A workload profile is represented as a vector of CPU, memory, and I/O states. The performance events are attributed to the specified benchmark and are used together with the power measurements to create a set of n power models that correspond to a workload profile.

The resource utilization of the benchmarks is automatically labeled into the workload profiles according to the data exploration phase. The labeled resource utilization and performance events train the classifier. We use three different machine-learning techniques, specifically XGBoost, Random Forest, and LogitNet, as a comparison to determine which of them results in the best classification.

We also use two very different systems, a Logical Partition (LPAR) on an IBM zEnterprise EC12 and a standard x86 server. We use the LPAR for the classification task and for the power consumption prediction, we use the x86 server as representative for a typical data center server. The case study in our evaluation is performed on the x86 server.

Chapter 8

Data Acquisition and Exploration

In this chapter, we will first give a short overview of the used testbed and benchmarks. We follow up with the description of the data acquisition phase in Section 8.2 together the selection of performance events for both systems, the EC12, and x86 server. In Section 8.3, we describe our definition of the workload profiles based on the recorded performance events during the benchmark runs. Finally, in Section 8.4, we shortly describe the machine learning approaches that we used for classification.

8.1 Testbed and Benchmarks

Our approach is tested with a typical x86 cloud data center server and the testbed setup for the SUT is as described in Section 2.3.1. The SUT is equipped with 32 GB of memory, 10 cores with 20 threads, and runs an Ubuntu 18.04.5 with kernel version 4.15.0-147-generic. For the classification task, we additionally use an LPAR on an IBM z12 with 2 cores and threads, 8 GB of memory, running the same Ubuntu version as the x86 server but with kernel version 4.15.0-126-generic.

The LPAR on the EC12 is not directly measurable. Hence, we read the power consumption information of the EC12 from the Hardware Management Console (HMC). The HMC provides us the total power consumption of the system at a sampling rate of 15 s. This approach has some drawbacks to a connected power analyzer. First, the power measurements reported will have noise included due to other users with LPARs on the same machine. The small size of our LPAR compared to the total machine size [Dob+13] makes it more challenging to differentiate noise from the power consumption caused by our benchmark execution. Second, the sampling rate is low compared to our power analyzer that reports the average power consumption every 1 s with a sampling rate of about 10 kHz, which degrades the results. We, therefore, opted to evaluate only our classification approach on the EC12 while the power prediction phase and case study is done on the x86 server.

Machine learning approaches typically need large amounts of training data to be accurate. We, therefore, decided to use benchmark suites, including a multitude of benchmarks stressing different hardware components instead of a single benchmark. The Phoronix Test Suite that we selected comes with over 200 benchmarks [Sui21]. Some benchmarks have configuration options. To further increase the variety of our data, we also ran the benchmarks in different configuration settings. Table 1 shows all benchmarks and their different configurations that we have run. It also denotes which benchmarks we used for training and which we used for our evaluation. We also used stress-ng that comes with over 220 benchmarks [Kin19] for our approach to have additional benchmark runs acting as application workload profiles. The used stress-ng benchmarks are listed in Table 2.

The decision for the Phoronix Test Suite and stress-ng is that they can stress a wide variety of system components, including but not limited to CPU, caches, memory, and I/O. Both benchmark suites are open source and come with the Linux distribution on our SUTs. Although both suites are portable and runnable on both SUTs, a few benchmarks are not executable on both. Those benchmarks were not executed on either of the two SUTs. We also did not use the networking benchmarks for our approach.

For our use case study to show that our trained machine learning model for diffferent workload profiles works, we use Elasticsearch, a search and analytics engine [GT15]. Yet, the model is not trained for this specific application but from the benchmark runs from the Phoronix Test Suite and stress-ng.

8.2 Acquisition Phase

In the data acquisition phase, we gather the necessary data from which we build our workload profiles of different applications. We define a workload profile as a vector of *CPU*, *memory*, *I/O read*, and *I/O write* utilization, as shown in Equation 8.1. Equation 8.1 also shows that each of the three elements of the workload profile vector can have one of four states ranging from *low* to *very_high*. These states are the classes in our approach. Our classification, therefore, consists of four classification processes, one for each subsystem. The goal for this step is to gather the data from the benchmarks that allow us to define the four states of each of the three workload profile components together with the performance events that caused on the four states in the SUT's subsystem as well as the power consumption that we use to train four linear regression power models for the evaluation.

$$\begin{aligned} profile &= [cpu, memory, io\_read, io\_write] \\ cpu, memory, io &\in \{low, medium, high, very\_high\} \end{aligned} \tag{8.1}$$

For this step, as shown in Figure 7.1, we need the following data recorded on the SUT.

- Power consumption
- Resource utilization
- Performance events

The measurement of the power consumption is described in the previous Section 8.1, and we monitor the resource utilization for an application with tooling available on the operating system. The resource utilization monitored consists of the CPU utilization in percentage, memory utilization in the percentage of allocated memory, and I/O utilization in kilobytes read from disk and kilobytes written to disk.

Recording the performance events presented a challenge for both SUTs as the systems differ widely in their architecture and available performance events. On the EC12, a large set of performance events can be monitored without a noticeable overhead. The recording of performance events on the x86 system is also limited to a few events without significant overhead by reconfiguring the PMU to monitor a different set of events.

For the x86 server, the set of five performance events shown in Table 8.1 are used for our experiment. The performance events for the EC12 are shown in Table 3. We record all performance events on the EC12 except events known to be zero and that are only triggered when the machine is in a certain state not applicable to our benchmark runs or when the applications would explicitly use an EC12 exclusive feature. The performance events of the x86 server are selected based on the work of Bircher et al. [BJ07], and either has a known correlation to certain performance characteristics, like Instruction Per Cycle (IPC), or we assume that they can represent the resource usage for a certain subsystem, such as memory.

The events *INSTRUCTIONS_RETIRED*, *UNHALTED_CORE_CYCLES*, and *MISPREDICTED_BRANCH_RETIRED* represent the resource usage of the CPU. While the first two events give a good representation, we also selected the branch misprediction as they can cause a pipeline flush resulting in performance penalties. The *LLC_MISSES* are representing the memory resource as missing the last-level-cache will result in memory access. In the case of the

DTLB_MISSES, they also represent memory but also I/O activity. The Translation Lookaside Buffer (TLB) holds mappings from virtual to physical memory addresses that are assumed by the CPU to be needed in the near future to avoid a lookup in the page table in memory. A miss on the TLB can, therefore, cause a pipeline stall due to a page table lookup or, in some cases, even I/O activity as a memory page could be swapped out [BJ07].

Table 8.1: Performance events selected on the x86 server.

Event	Description
INSTRUCTIONS_RETIRED	Counts the number of completely executed instructions
UNHALTED_CORE_CYCLES	Number of cycles, in which the core was not in HALT state
MISPREDICTED_BRANCH_RETIRED	Number of fully executed, mispredicted branch instructions
LLC_MISSES	Misses in the last-level-cache
DTLB_MISSES	Misses in the data TLB

8.3 Exploration Phase

In our data exploration phase, we first define the four states *low*, *medium*, *high*, and *very_high* for our resource profiles shown in Equation 8.1 to automatically label our data. Afterward, we use an ANOVA analysis on the performance events of the EC12 to select the most suitable events for our classification approach.

8.3.1 Defining Resource Profile Classes

To define our resource profiles, we take the average utilization values of the benchmark runs from which we derive the absolute boundaries for the four levels, the class definitions. As our SUTs differ primarily in the size of CPU and memory, the absolute values for these levels will also differ.

First, we take a look at the CPU utilization of both SUTs. Figure 8.1 and Figure 8.2 show the CPU utilization. We can see that the majority of benchmark runs caused a CPU utilization below 20% or above 80% for the x86 server. The benchmarks causing a lower CPU utilization are mostly I/O-heavy applications. We observed during the measurement that the benchmark runs that cause a

lower CPU utilization is causing the CPU to be mainly in the I/O-Wait state. The CPU utilization for the EC12 shows similar results. We, therefore, define both SUTs the same classes.

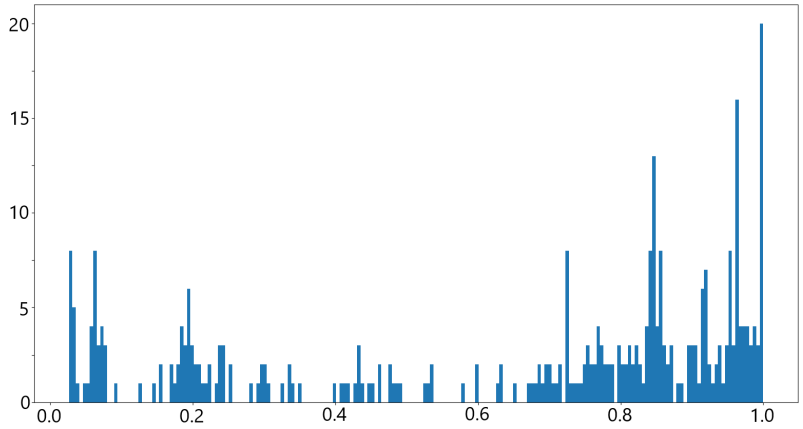


Figure 8.1: Average CPU utilization histogram for the x86 server.

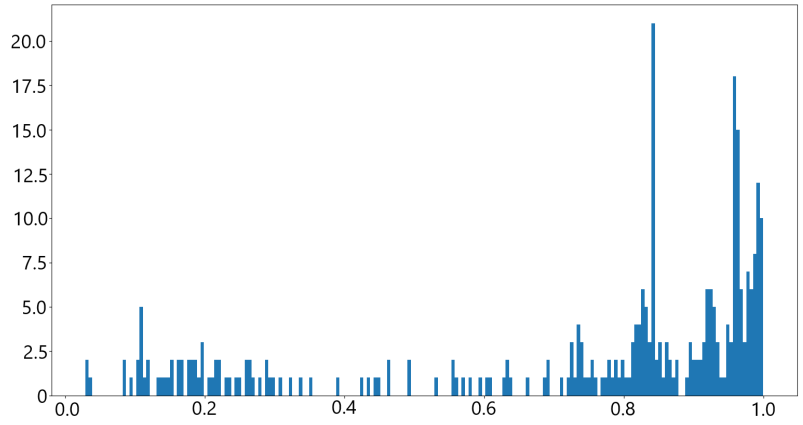


Figure 8.2: Average CPU utilization histogram for the EC12.

In terms of memory utilization, both SUTs behave very similar again, as can be seen in Figure 8.3 for the x86 server and in Figure 8.4 for the EC12. Most benchmarks cause a very low memory utilization. 205 of the 334 benchmark runs cause an average memory utilization of below 0.5% for the x86 server, which is less than 0.16 GB. As the LPAR that we used has a smaller amount of memory, fewer benchmark runs show utilization of below 0.5%. However, 182 runs are still in the first bin. As the utilization is low, we dropped the *very_high*

class for the memory, resulting in only three classes where every utilization level above 20% for the x86 server and above 30% for the EC12 is classified as *high*.

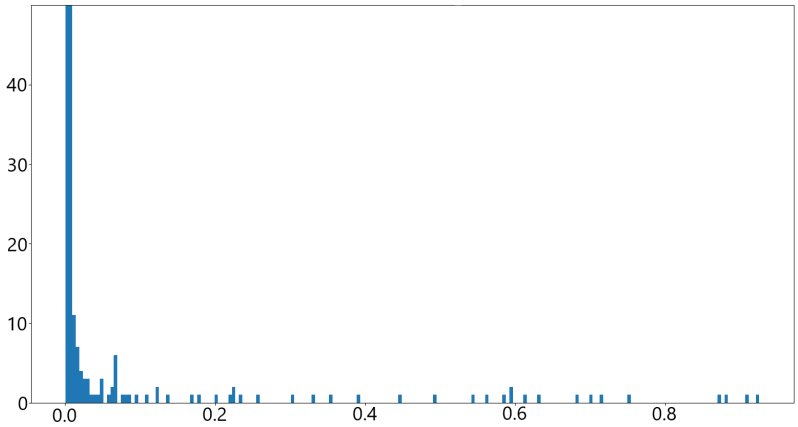


Figure 8.3: Average memory usage histogram for the x86 server. Truncated view with 205 benchmark runs in the first bin.

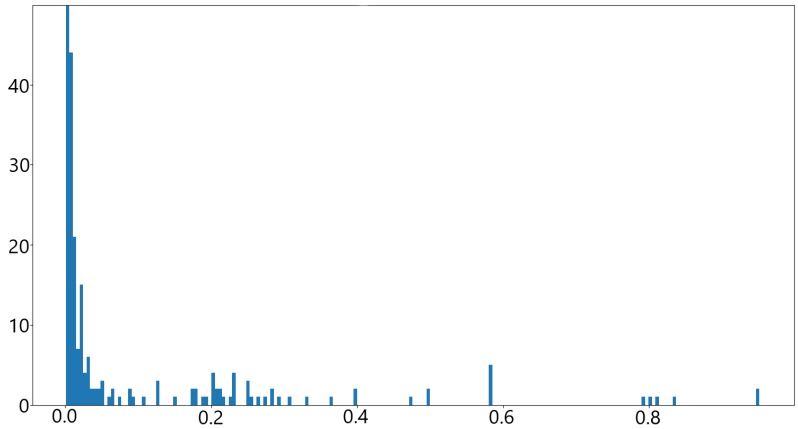


Figure 8.4: Average memory usage histogram for the EC12. Truncated view with 182 benchmark runs in the first bin.

Figure 8.5 and Figure 8.6 show the bytes read from the disk for the x86 server and EC12 respectively. It is clear from both SUTs that the benchmark runs cause only a low amount of I/O activity. Most of the benchmark runs are in the first bin. As with memory utilization, we decided to reduce the number of classifications. For the x86 server, only the two first classes, *low* and *medium*,

are left. For the EC12, three classes are chosen, and only the *very_high* class is dropped because less benchmark runs can be found in the first bin with 262 benchmark runs for the x86 server compared to 311 benchmark runs for the EC12.

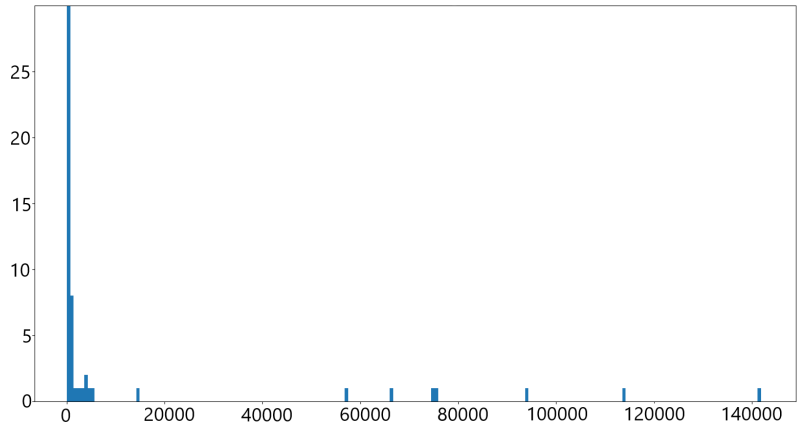


Figure 8.5: Average kilobytes read from disk histogram for the x86 server. Truncated view with 262 benchmark runs in the first bin.

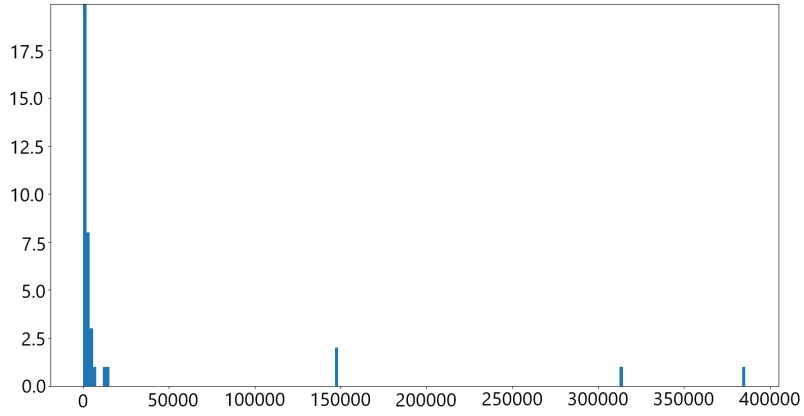


Figure 8.6: Average kilobytes read from disk histogram for the EC12. Truncated view with 311 benchmark runs in the first bin.

The last part of the resource profile, the bytes written to disk, shows a comparable characteristic as the bytes read from disk in Figure 8.7 and Figure 8.8. The majority of the benchmark runs have little I/O activity writing to disk. We drop

the highest class from the resource profile, similar to the memory utilization and the bytes read from disk.

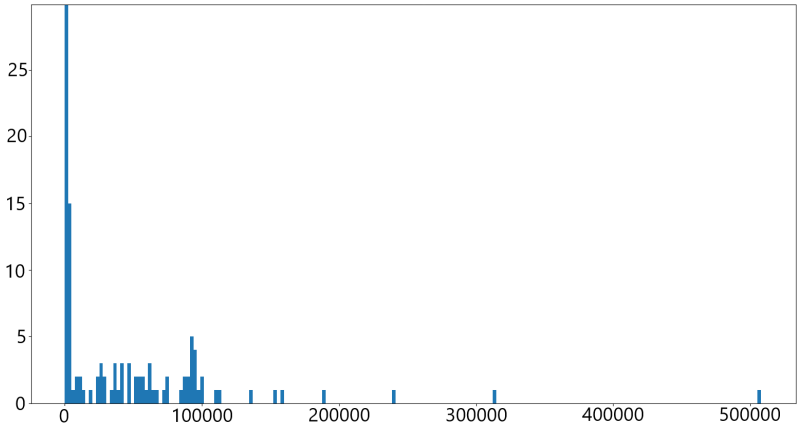


Figure 8.7: Average kilobytes written to disk histogram for the x86 server. Truncated view with 267 benchmark runs in the first bin.

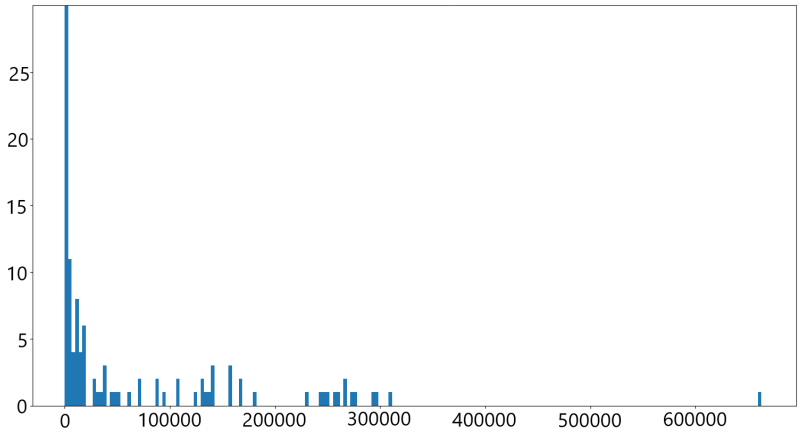


Figure 8.8: Average kilobytes written to disk histogram for the EC12. Truncated view with 268 benchmark runs in the first bin.

The defined thresholds of the classes for the resource profile are shown in Table 8.2. Due to the different sizes of the SUTs, the absolute values are not comparable. While the x86 server had 32 GB of memory, the LPAR on the EC12 was equipped with only 2 GB. Therefore, the definition of what a *high* memory

utilization differs from system to system. Still, the defined thresholds can now be used for the automated labeling of our recorded resource utilizations.

Table 8.2: Class definitions for both SUTs.

SUT	State	CPU	Memory	kB Written	kB Read
x86	low	0.000 - 0.375	0.00 - 0.04	0 - 10000	0 - 3000
	medium	0.375 - 0.625	0.04 - 0.20	10000 - 50000	>3000
	high	0.625 - 0.875	>0.20	>50000	
	very_high	0.875 - 1.000			
EC12	low	0.000 - 0.375	0.00 - 0.05	0 - 9000	0 - 1000
	medium	0.375 - 0.625	0.05 - 0.30	9000 - 50000	1000 - 5000
	high	0.625 - 0.875	>0.30	>50000	>5000
	very_high	0.875 - 1.000			

8.3.2 Distinguish Classes

The next step to our classification approach is to determine if we can distinguish our classes in Table 8.2 with the available performance events on both SUTs. We use an ANOVA to select the performance events, features in the machine learning context that are most suitable to distinguish our classes. The ANOVA identifies a subset of available performance events that have the highest relevance to the output variable, in our case, the resource profile. The results of the ANOVA for the x86 server and EC12 are listed in Table 8.3 and Table 8.4. The first line for a performance event provides the F-values, and the second line shows the p-value. The significance level for both ANOVAs is 0.05.

Distinguish CPU The performance event most suitable to differentiate between the CPU classes on the x86 server is UNHALTED_CORE_CYCLES with an F-value of close to 37.5. This result is expected as the number of cycles the CPU is not in a HALT state is a direct measure of its activity. The lowest F-value for the CPU relevant performance events has been calculated for MISPREDICTED_BRANCH_RETIRED with 6.7. However, all selected performance events have high F-values and low p-values close to zero. These values mean that all five performance events selected on the x86 server are suitable as features for the CPU classification.

For the EC12 LPAR, most performance events also exceed the $F_{critical}$ value and have p-values close to zero. This behavior is expected for performance

events such as CPU_CYCLES and INSTRUCTIONS as they, similar to the x86 server's UNHALTED_CORE_CYCLES, directly capture the CPU activity. Not expected to be relevant for classifying CPU activity are the performance events that occur with a comparatively low frequency listed below. According to our ANOVA, these performance events have F-values of greater than 26 and are even more relevant to CPU activity than the CPU_CYCLES event with an F-value of about 23.

- L1D_OFFCHIP_L3_SOURCED_WRITES
- L1D_ONCHIP_L3_SOURCED_WRITES_IV
- L1I_ONCHIP_L3_SOURCED_WRITES
- L1I_ONBOOK_L4_SOURCED_WRITES

The performance events SHA_FUNCTIONS and SHA_CYCLES have the lowest relevance to distinguish the CPU activity and are below $F_{critical}$. These performance events typically capture activity in the EC12's cryptographic co-processor. They, therefore, have the lowest relevance as features for the CPU classification.

Distinguish Memory For the x86 server, the performance events LLC_MISSES and UNHALTED_CORE_CYCLES have the highest F-values. While the high score of the LLC_MISSES with about 22 is expected due to memory access after a last-level-cache miss, the UNHALTED_CORE_CYCLES with s slightly higher F-value of 23 is unexpected. Also unexpected is the low F-value for the DTLB_MISSES performance event. While we assumed that this performance event could represent memory activity through the page table access, the opposite is the case.

On the EC12, the CPU-related INSTRUCTIONS and CPU_CYCLES performance events are also informative for the memory classification. The INSTRUCTIONS performance event even has the highest F-value of over 48 of all recorded performance events. An expected outcome is also that performance events related to the level three, and level four cache have are relevant to distinguish memory activity where L1D_OFFCHIP_L3_SOURCED_WRITES has the highest F-value, with 45, of all cache related events. Performance events directly related to memory activity, the L1D_LMEM_SOURCED_WRITES and L1I_LMEM_SOURCED_WRITES have a comparatively moderate capacity to distinguish memory activity in our ANOVA. Similar to the CPU classification, the performance events SHA_FUNCTIONS and SHA_CYCLES have the lowest relevance.

Distinguish I/O The UNHALTED_CORE_CYCLES is the most significant feature for writing data to disk on the x86 server. As the INSTRUCTIONS_RETIRED performance event also has a higher F-value of 3.7 compared to the other events, it seems that the performance events for the CPU activity also can describe the I/O writing activity. The DTLB_MISSES event did not have the anticipated information on the I/O and memory activity. However, with an F-value of 2.3 it still contains more information about the I/O activity than the LLC_MISSES and MISPREDICTED_BRANCH_RETIRED performance events. Regarding the I/O reading activity, the MISPREDICTED_BRANCH_RETIRED event has the best capacity to distinguish the bytes read from disk classes. Nevertheless, the relevance compared to the CPU and memory F-values is lower. We assume from our ANOVA that the performance events on the x86 server do not carry as much information about the subsystem the further away from the CPU the subsystem is.

Comparing the F-values of the bytes read from disk and bytes written to disk to the memory and CPU F-values, we can see that the performance events provide less information to differentiate the I/O classes. This behavior is similar to the x86 server described before. Two outliers in the I/O write activity to our observation are the L1D_ONCHIP_L3_SOURCED_WRITES_IV, with an F-value of 46.5, and L1I_ONCHIP_L3_SOURCED_WRITES, with an F-value of 10.1.

We can observe similar behavior on both SUTs that the I/O activity is harder to distinguish through performance events. We assume that this behavior can have two reasons. First, the I/O subsystem is further away from the CPU; or second, an imbalance in the dataset of our benchmark runs that favor CPU intensive workloads over I/O heavy workloads, also observable in I/O histograms (Figure 8.5, 8.6, 8.7, and 8.8).

Table 8.3: ANOVA F-values and p-values on the x86 server performance events.

Performance Event	CPU	Memory	kB Written	kB Read
INSTRUCTIONS_RETIRED	15.4424	11.4189	3.6735	1.7582
	0.0000	0.0000	0.0264	0.1858
UNHALTED_CORE_CYCLES	37.4999	23.3584	8.3557	3.6993
	0.0000	0.0000	0.0003	0.0553
MISPREDICTED_BRANCH_RETIRED	6.7124	10.0897	0.4271	5.4940
	0.0002	0.0001	0.6528	0.0197
LLC_MISSES	27.0772	22.1438	1.6880	0.9817
	0.0000	0.0000	0.1865	0.3225
DTLB_MISSES	21.6118	6.9911	2.2500	1.1898
	0.0000	0.0011	0.1070	0.2762
$F_{critical}$	2.6321	3.0232	3.0232	3.8698

Table 8.4: ANOVA F-values and p-values on the IBM EC12 LPAR performance events.

Performance Event	CPU	Memory	kB Written	kB Read
CPU_CYCLES	23.0408	25.3164	4.2823	0.0151
	0.0000	0.0000	0.0146	0.9850
INSTRUCTIONS	27.7921	48.2583	4.1012	0.2286
	0.0000	0.0000	0.0174	0.7958
L1I_DIR_WRITES	21.4178	13.1137	6.8471	0.2672
	0.0000	0.0000	0.0012	0.7657
L1I_PENALTY_CYCLES	18.4836	12.4090	4.3773	0.2348
	0.0000	0.0000	0.0133	0.7909
L1D_DIR_WRITES	15.0791	10.0534	3.3007	0.1287
	0.0000	0.0001	0.0381	0.8793
L1D_PENALTY_CYCLES	14.6715	14.3303	2.3339	0.0678
	0.0000	0.0000	0.0985	0.9345
SHA_FUNCTIONS	0.7964	0.0993	0.1106	0.0281
	0.4966	0.9055	0.8953	0.9722
SHA_CYCLES	0.7968	0.0996	0.1107	0.0281
	0.4964	0.9052	0.8953	0.9722
DTLB1_MISSES	9.8780	14.2701	2.1981	0.0067
	0.0000	0.0000	0.1126	0.9934
ITLB1_MISSES	15.4846	10.7347	2.9669	0.2343
	0.0000	0.0000	0.0528	0.7913
L1D_L2I_SOURCED_WRITES	8.7732	12.6054	2.9761	0.0142
	0.0000	0.0000	0.0523	0.9859
L1I_L2I_SOURCED_WRITES	21.3684	13.0088	6.8422	0.2704
	0.0000	0.0000	0.0012	0.7633
L1D_L2D_SOURCED_WRITES	12.3970	3.1158	6.0348	0.3121
	0.0000	0.0456	0.0027	0.7321
DTLB1_WRITES	9.1156	13.8928	1.7233	0.0506
	0.0000	0.0000	0.1801	0.9507
L1D_LMEM_SOURCED_WRITES	16.3325	17.8460	2.4731	0.0251
	0.0000	0.0000	0.0859	0.9752
L1I_LMEM_SOURCED_WRITES	19.5266	23.1845	2.6663	0.0344
	0.0000	0.0000	0.0710	0.9662
L1D_RO_EXCL_WRITES	11.0440	15.8284	1.6750	0.0049
	0.0000	0.0000	0.1889	0.9951
DTLB1_HPAGE_WRITES	9.2228	8.7466	1.2141	0.1175
	0.0000	0.0002	0.2983	0.8892
ITLB1_WRITES	7.0040	4.5938	2.0630	0.3456
	0.0001	0.0108	0.1287	0.7080
TLB2_PTE_WRITES	9.3026	15.9498	2.0584	0.1230
	0.0000	0.0000	0.1293	0.8843
TLB2_CRSTE_HPAGE_WRITES	15.4736	17.4209	2.1402	0.0988
	0.0000	0.0000	0.1192	0.9059
TLB2_CRSTE_WRITES	14.8018	17.3395	1.4182	0.0356
	0.0000	0.0000	0.2436	0.9650
L1D_ONCHIP_L3_SOURCED_WRITES	9.0393	10.9797	2.8806	0.0301
	0.0000	0.0000	0.0575	0.9704
L1D_OFFCHIP_L3_SOURCED_WRITES	26.9282	45.3858	2.3536	0.1324
	0.0000	0.0000	0.0966	0.8760
L1D_ONBOOK_L4_SOURCED_WRITES	16.6099	14.8228	2.8230	0.0879
	0.0000	0.0000	0.0608	0.9158
L1D_ONCHIP_L3_SOURCED_WRITES_IV	26.4027	11.0062	46.5028	0.6512
	0.0000	0.0000	0.0000	0.5221
L1D_OFFCHIP_L3_SOURCED_WRITES_IV	18.8898	29.2215	1.4369	0.0398
	0.0000	0.0000	0.2391	0.9610
L1I_ONCHIP_L3_SOURCED_WRITES	26.9254	31.5368	10.1472	0.0045
	0.0000	0.0000	0.0001	0.9955
L1I_OFFCHIP_L3_SOURCED_WRITES	9.3533	14.3683	2.6601	0.0783
	0.0000	0.0000	0.0714	0.9247
L1I_ONBOOK_L4_SOURCED_WRITES	29.8413	29.6591	5.4591	0.0313
	0.0000	0.0000	0.0046	0.9692
L1I_ONCHIP_L3_SOURCED_WRITES_IV	9.0717	13.5836	2.4864	0.0346
	0.0000	0.0000	0.0848	0.9660
$F_{critical}$	2.6318	3.0228	3.0228	3.0228

8.4 Labeling and Training

After we have collected the data from our benchmark runs in Section 8.2 and defined our resource profile classes in Section 8.3.1, we used ANOVA to determine the influence of certain performance events on the classification in Section 8.3.2. We now can train our machine learning models for classification. We use three different machine learning techniques for comparison. Each of the three parts of the resource profile, CPU, memory, and I/O write. We decided to drop the I/O read component from our resource profile due to the high imbalance in the recorded values. The ANOVA results show that the I/O read class cannot be easily differentiated with the recorded performance events and the class definition resulted in only two classes for the x86 and three classes for the EC12.

The first approach is XGBoost, the general idea of decision tree bagging and boosting is described in Section 2.5.1. XGBoost is a common implementation that employs the aforementioned techniques for decision trees. XGBoost is also known to produce good results [CG16]. In our approach, we selected the following hyperparameters shown in Table 8.5 for our evaluation that produced reasonable results. However, XGBoost's hyperparameters can be tuned with various approaches that might improve its classfication results even further.

Table 8.5: XGBoost hyperparameter for the classification tasks.

	CPU	Memory	kB Read and kB Written
number_estimators	61.0000	80.0	80.0
max_depth	4.0235	8.0	8.0
colsample_bytree	0.9705	1.0	1.0
colsample_bylevel	0.7682	1.0	1.0
eta	0.0854	0.3	0.3
subsample	0.1598	1.0	1.0
alpha	0.2179	0.0	0.0
gamma	0.3294	0.0	0.0
reg_lambda	0.7237	1.0	1.0

Our second machine learning approach, Random Forest, is also decision tree-based and described in Section 2.5.1. Compared to XGBoost, Random Forest does not use boosting. As it does not employ boosting, it acts as a comparative approach to decide if the additional effort can actually increase the classification accuracy in our use-case, classifying the resource profile.

The third machine learning approach is logistic regression for ordinal categories for the use of our resource profiles that do follow an order from lowest to highest. The principal approach for logistic regression is described in Section 2.5.3.

8.5 Concluding Remarks

This part introduces a classification approach for a DevOps environment. The rapidly changing requirements entail frequent code changes. This, in turn, leads to inaccurate power models as they are typically trained to the specific application. Based on the resource profile of an application, either averaged over the total runtime or over an execution phase, we can select a power model from a set of models that is more accurate for the given application or situation the application is in. This automated classification process addresses our *RQ B.1* ("How to classify applications in an automated manner according to their resource usage profile in order to derive suitable and, therefore, more accurate power models?"). After we defined the resource profile classes for a system, we can automatically label recorded data, classify each subsystem, and build a resource profile that determines the suitable power model.

Part IV

Power Consumption Modeling

Chapter 9

Predicting Power Consumption of High-Memory Bandwidth Workloads

We propose two novel power models after we can determine which power model is the most suitable for a given resource profile. High-performance workloads with high bandwidth memory utilization are among the most power-consuming software applications. Especially when comparing high-performance CPU workloads, workloads with additional high-bandwidth memory usage draw more power than their less memory-intensive counterparts [Kis+15b].

Software developers can directly influence the power consumption of high-bandwidth software components through their implementation and design decisions. Especially, characteristics such as data type and data traversal methods can significantly impact performance and power consumption. Explicit knowledge on the relationship between power consumption and implementation details can help developers make informed choices regarding energy efficiency during development and lead to more efficient software.

We propose a power model that bridges this gap of models being either too generic or too specific, as mentioned in Section 3.1, by modeling power consumption based on concrete software properties while considering hardware characteristics on a more abstract level. Specifically, model parameters that characterize hardware, the system-level information may be set to default values, in which case the model's power prediction is proportional to actually consumed power on a concrete system. This way, the model enables developers to compare the power consumption of implementation alternatives for high memory bandwidth software components. A user may also parameterize our model with system-specific hardware parameters, in which case the model results in absolute accuracy increases.

- We classify the differences in power consumption that high-memory-bandwidth workloads can achieve.

- We propose a model that allows for the comparison of implementation options based on data type and data traversal method.

The goal of the power prediction model is modeling power consumption of *1*) CPU and *2*) full-system power to address our research question *RQ B.2*. We are also able to estimate memory power. For each power prediction, we predict either relative changes in power depending on workload characteristics or absolute power, which requires additional system-level information on the hardware. We published our *Contribution 5* in the Proceedings of the 8th ACM/SPEC International Conference on Performance Engineering (ICPE) 2017 [SKK17b].

CPU power (P_{CPU}) and full-system power (P_{total}) are the results of our workload model, whereas memory power (P_{mem}) can be derived from the former two. To derive memory power, we assume that the high-memory-bandwidth workload exercises a negligible amount of I/O and that the static or idle power of the system (P_{static}) is constant. If so, P_{mem} can be derived as follows (Equation 9.1):

$$P_{mem} = P_{total} - P_{CPU} - P_{static} \tag{9.1}$$

Under the assumption that the idle power consumption P_{static} is constant, we can read the value from manufacturer data sheets or standard benchmark results, such as SPECpower_ssj 2008 benchmark [Lan09a] or the SPEC SERT suite [LT11].

The power models for full-system and CPU power are based on the assumption that larger data types and bigger steps in data traversal cause lower CPU power consumption. We expect this behavior due to how execution and power consumption depend on the CPU cache. The general rationale is that every memory access instruction reads data either from cache (cache hit) or directly from memory (cache miss). Each direct memory access caused by cache miss will, therefore, cause the CPU to wait. This idle time of the CPU reduces its power consumption while simultaneously causing the memory to perform work, increasing the memory power consumption. Consequently, the CPU power consumption decreases with the cache miss probability up to the minimum power consumption $P_{min,CPU}$ and $P_{min,total}$. The memory power consumption increases with the cache miss probability. However, as memory consumes far less power than CPU overall full-system power consumption scales with the CPU power consumption factor α (Equation 9.4) to $P_{max,CPU}$ as well as $P_{max,total}$, as shown in Equation 9.2 and Equation 9.3.

$$P_{CPU} = P_{min,CPU} + (P_{max,CPU} - P_{min,CPU}) * \alpha \tag{9.2}$$

$$P_{total} = P_{min,total} + (P_{max,total} - P_{min,total}) * \alpha \tag{9.3}$$

CPU and full-system power values are either system-specific or default values. Full-system power values can be read from standard benchmark results from the SERT suite [LT11]. However, the cache miss probability also depends on the workload. We, therefore, calculate cache hit probability (Equation 9.4) based on the data type size, stride, and cache size of the Last Level Cache (LLC). Cache size can also be set using a default value. However, cache sizes of existing processors are public information and available from the manufacturer's datasheets.

$$\alpha = \frac{datasize * stride}{cachesize} * inverseprefetcheraccuracy \tag{9.4}$$

The $inverseprefetcheraccuracy$ in Equation 9.4 is the only hardware parameter that can not be pulled from a manufacturer data sheet or standard benchmark result. Fortunately, it does not affect the ranking of relative results, which means that we can omit it for relative comparisons. It models the hardware cache prefetcher's ability (or rather inverse ability, as we are modeling cache misses) to pre-fetch data for the cache that would typically not be in the cache according to naive caching algorithms. The cache miss probability is the modeled hardware parameter. This hardware parameter enables the model to compare relative results if neglected as described and predict the power consumption with reasonable accuracy.

Note that we expect the model's absolute prediction accuracy to strongly depend on the accuracy of the P_{max} and P_{min} parameters. Suppose these do not correlate for an entire system and a corresponding CPU power model. In that case, it may be advisable to use different cache miss probabilities for these models, even though this is semantically unintuitive.

The data types are selected, which we see as the most commonly used data types in C programming. We do not distinguish between floating-point and integer types. Therefore longer strides for integers are not evaluated.

Chapter 10

Function-level Power Modeling

Next to modeling high-memory bandwidth workloads, developers also need additional information on where improvements are productive and can reduce their application's power consumption. Yet, this often needs careful instrumentation, specialized measuring equipment, or hardware information not available during the development phase. Taking rapid code changes into account makes it difficult as instrumentation also needs to constantly change with the source code of the business logic. We, therefore, propose a new power model in our *Contribution 4* that is based on application-level measures, as defined in our Foundation 2.2. Our approach determines the power consumption of application functions without modification of the application itself or the monitoring. The source code can remain free of instrumentation code that has no relation to the actual business logic. Existing applications can be monitored without change in a DevOps environment and specialized measurement equipment hardly present in most data centers. Our contribution of a function-level power model, therefore, addresses our research question *RQ B.3* of *Goal B*. Together with an analysis if this model is transferable across three different SUTs, it also partially addresses research question *RQ B.4*. This contribution has been published in the Proceedings of the 16th IEEE International Conference on Autonomic Computing (ICAC) 2019 [Sch+19a].

10.1 Power Consumption of Applications

The power consumption P_{total} of a system consists of two parts, the static or idle power consumption P_{static} and the dynamic power consumption $P_{dynamic}$, as shown in Equation 10.1. The dynamic power is mainly dependent on the systems utilization or CPU utilization [CP15a; AA10; FWB07]. Hence, we base our function-level approach on a CPU-based model. The idle consumption must be measured and removed from the total power to determine the power consumption caused by the running software. We assume that $P_{idle} = P_{static}$ for our approach. The relation of power and temperature is considered by

operating our SUT in a controlled environment inside an air-conditioned data center and warm-up periods before measurements, described in Section 2.3.1.

$$P_{total} = P_{static} + P_{dynamic} \tag{10.1}$$

Our approach uses performance events to derive the power consumption without actually measuring it. Performance events are a statistics feature available in modern CPUs. Their main use is identifying performance bottlenecks in an application by counting specific events occurring in a CPU. We describe performance events in more detail in Section 2.2. However, performance events have been shown to be correlated with power consumption, which can be leveraged to model power consumption as shown in many existing works [IM03; SBM09; Che+10; LPF10; BJ12; Rod+13; Son+13]. Typically, regression models (Equation 10.2) are used in this context. $\mathbf{Y}$ is the *response* variable, that is, $P_{dynamic}$ in our case. $\mathbf{X}$ is the vector of *regressor* variables for which we use the monitored counts of performance events. The regression parameters β must be trained to derive the power model.

$$\mathbf{Y} \approx f(\mathbf{X}, \beta) \tag{10.2}$$

The selection of performance events is critical to building a viable model and needs expert knowledge about the system. To avoid overfitting the regression model to specific events relevant to our scenario, we use the identified events in the work of Yasin [Yas14]. In his work, he uses performance events to identify bottlenecks systematically. As performance and power consumption are related but not identical, we see this as a good selection of performance events without fitting the model to a particular application. The following performance event descriptions are taken from the CPU manufacturers manual and shortened for brevity [Int16].

- *CPU_CLK_UNHALTED.THREAD*: "The event counts the number of core cycles while the logical processor is not in a halt state."
- *IDQ_UOPS_NOT_DELIVERED.CORE*: "Counts the number of uops that the Resource Allocation Table (RAT) issues to the Reservation Station (RS)."
- *UOPS_ISSUED_ANY*: "Counts the number of uops not delivered to Resource Allocation Table (RAT)."
- *UOPS_RETIRED.RETIRE_SLOTS*: "Counts the retirement slots used."

- *INT_MISC.RECOVERY_CYCLES*: "Core cycles the allocator was stalled due to recovery from earlier machine clear event for this thread."
- *CYCLE_ACTIVITY.STALLS_MEM_ANY*: "Execution stalls while memory subsystem has an outstanding load."
- *RESOURCE_STALLS.SB*: "Cycles stalled due to no store buffers available."

We monitor performance events per CPU core. As mentioned, we do not want to instrument our application directly; we need a method to attribute the performance events to application functions. Attributing the performance events to the function calls moves them from system-level metrics to the application-level as in our definition in Section 2.2. To keep the source code of the application as it is, we obtain the stack trace with Kieker[1] [HWH12; HH20], running concurrently to the application. Kieker claims an overhead below 3% [Hoo+09]. The application we selected is the image provider service of the TeaStore[2] [Kis+18] benchmarking application that we described in detail in Section 2.4.2. TeaStore is a Java application built explicitly for testing power and performance models. The associated image provider uses CPU (resizing images), memory (through caching), and I/O (loading uncached images from storage). It features recursive and non-recursive function calls allowing us to validate our model's compatibility with recursion. Function call overhead is attributed to the calling function.

10.2 Model

Our proposed model assigns performance events on a system-level to application function calls through a stack trace. This novel approach can introduce errors through preemptive operating systems. Hence, we introduced a correctional factor to our model. We first describe the modeling approach in Section 10.2.1 followed by the correctional factor in Section 10.2.2.

10.2.1 Regression Stack Model

The function-level power consumption model is based on a linear regression model. Instead of having a multitude of models, one for each function of the application, we train our regression model on the complete application and use the performance events caused by a function for the prediction. This section

[1] Kieker Application Monitoring: `http://kieker-monitoring.net/`
[2] TeaStore: `http://descartes.tools/teastore`

describes our approach to attribute system-level performance events to single functions of the application under test.

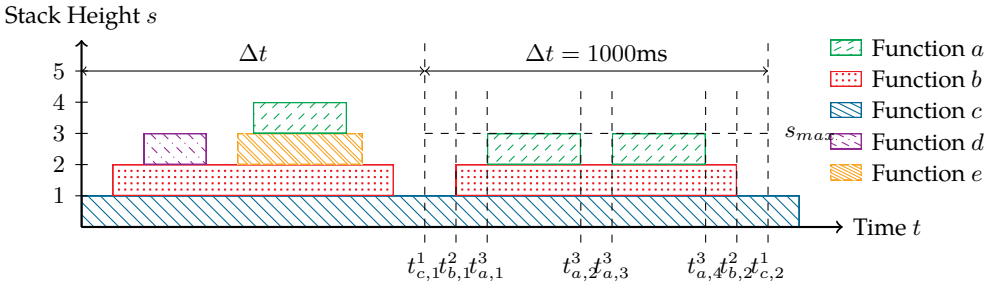


Figure 10.1: Example stack trace with five functions, performance event sampling interval Δt and stack sample times.

To calculate the power consumption of our application, we need to first know for how long a function of our application has exclusive access to the processing resources. For this, we state that if a function is on top of the call stack, it has exclusive access. An example stack trace is shown in Figure 10.1 with five different functions. The time a function x resides on stack height s can then be calculated with Equation 10.3 by adding up the times between when the function was called $t^s_{x,n}$ and its return $t^s_{x,n+1}$.

$$\sum_{i=1}^{n-1} t^s_{x,n+1} - t^s_{x,n} \tag{10.3}$$

As functions can call subroutines, we also calculate the time any function was atop the caller up to the maximal stack height s_{max} of the current interval Δt, as shown in Equation 10.4.

$$\sum_{j=s+1}^{s_{max}} \sum_{i=1}^{n-1} t^j_{i+1} - t^j_i \tag{10.4}$$

By subtracting the time of the callee from the time of the caller and dividing it by the sampling interval time $\Delta t = 1000$ms (see Equation 10.5), we calculate the relative time-share $f_x \leq 1$ of how long a function x has exclusive resource access as follows:

$$f_x = \frac{\sum_{i=1}^{n-1} t^s_{x,i+1} - t^s_{x,i} - \sum_{j=s}^{s_{max}} \sum_{i=1}^{n-1} t^j_{i+1} - t^j_i}{\Delta t} \tag{10.5}$$

The server operating systems (usually preemptive and able to withdraw resources) weaken the assumption that an application has exclusive access to a machine's resources, which introduces inaccuracies in the measurement of the stack trace and performance events. The withdrawal of resources is reflected in the stack trace by a prolonged time between the function call and its return. As our approach distributes performance events according to the time a function spends on top of the stack, this subsequently leads to more performance events assigned to a function than it actually causes. Not only can the prolonged time-share affect the performance event counts, but also the callee of a subroutine. For example, function a in Figure 10.1 is called once from function e and later twice from function b. Taking that function e generates event E that is not present for function b, then function a falsely gets assigned event E. Additionally, the preemptive operating system will also lead to higher recordings of performance events. This effect leads to two inaccuracies in our approach. First, a measurement sample can contain more performance events, and secondly, a longer time-share f_x for the interrupted functions adding performance events not associated with a call to the functions.

10.2.2 Correction Factor

As described before, our modeling approach could falsely attribute performance events to a function call in two ways. First, by breaking the assumption of exclusive computing resource access until a function returns, and second, by different callees. We, therefore, extend our approach by a correctional factor described in this section.

The first problem is countered by measuring performance events in the idle state and removing them from the performance event samples. We assume that the operating system must run independently of the workload. A network-centric application would force the operating system to empty the network interface buffers more frequently, limiting workload independence. The power consumption estimation should reflect whether the application is responsible for higher operating system interventions. Only preemptions that also take place in an idle state and are not attributable to the application should be removed. While this still leaves errors in the stack trace, the application in question does not need instrumentation to pause performance event recording if control of the resources is not available.

Introducing a correction factor to our performance event assignment addresses the second problem. Removing falsely counted performance events does not correct the prolonged time-share and still leads to disproportional assignments of performance events to interrupted functions. Hence, we introduce a correction factor c_t shown in Equation 10.6.

$$\mathbf{c}_t = \begin{bmatrix} c_{0,a} & c_{1,a} & \dots & c_{n,a} \\ c_{0,b} & c_{1,b} & \dots & c_{n,b} \\ \vdots & \vdots & \ddots & \vdots \\ c_{0,z} & c_{1,z} & \dots & c_{n,z} \end{bmatrix} \tag{10.6}$$

Each factor $c_{E,x}$ is built through an unweighted moving average over the last n number of performance events of event E assigned to function x in Equation 10.7. The smoothing over the history reduces the amount of wrongfully assigned performance events. This correction is limited, as functions only called from one specific callee are not correctable since the history never contains samples without the wrong performance events.

$$c_{E,x} = \frac{1}{n} \sum_{i=0}^{n-1} p_{corrected,x,M-i} \tag{10.7}$$

Combining a function's time on top of the stack f_x (Equation 10.5) and the monitored performance events into a column and row vector respectively in Equation 10.8, we finally calculate the corrected amount of performance events for each function $p_{corrected,t}$ at time t in Equation 10.9. The total time-share for all functions $\mathbf{f}_t^T \leq 1$ must be fulfilled.

$$\begin{aligned} \mathbf{f}_t^T &= \begin{bmatrix} f_a & f_b & \dots & f_z \end{bmatrix} \\ \mathbf{p}_t &= \begin{bmatrix} pc_0 & pc_1 & \dots & pc_n \end{bmatrix} \end{aligned} \tag{10.8}$$

$$\mathbf{p}_{corrected,t} = \left(\frac{\mathbf{f}_t \cdot \mathbf{p}_t + \mathbf{c}_t}{2} \right) \tag{10.9}$$

In combination with the regression model, we can now determine the power consumption per function as the moving average over the last $n+1$ assignments.

Hardware power management, like DVFS, is not directly modeled but indirectly through the use of performance events. Disabling DVFS would increase accuracy but would reduce applicability in practical applications. With decreasing clock speed, the number of performance events decreases as well, and the runtime increases. As we assume a linear relation between power (represented by performance events) and runtime, the energy efficiency is identical.

10.3 Testbed Setup

To investigate whether our model can distinguish between the application's functions and the overhead generated by the software stack and operating system, we set up the testbed shown in Figure 10.2. We use Apache JMeter[3] [ASB17] as a load driver to issue HTTP POST requests to the SUT over a dedicated network interface. Thus, the employed monitoring tool Kieker does not create overhead inside the network used for testing. Kieker sends generated stack traces through a second network interface to a dedicated logging server. A Hioki PW3335 power meter connected to the benchmark controller measures the SUT's wall power. After each measurement, the benchmark controller collects the request rate, performance events, and stack traces.

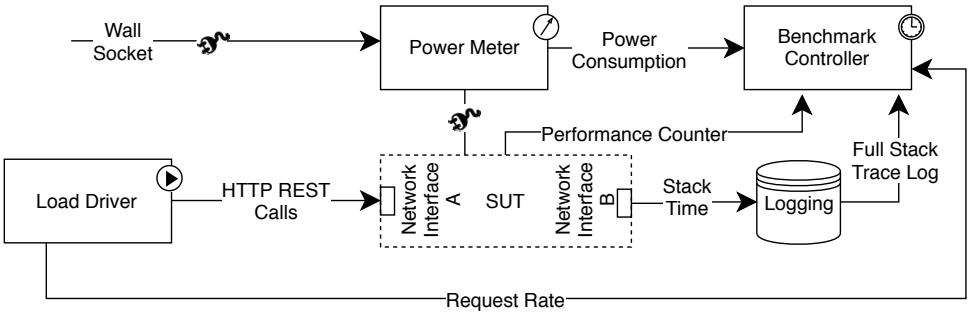


Figure 10.2: Evaluation testbed setup.

We use three physical servers as SUTs. Table 10.1 lists the SUT servers to evaluate the transferability of our approach without retraining our linear regression model. Each SUT has a different Intel Xeon CPU generation, different core and thread counts (4/8, 8/16, 12/24) and the *medium* and *large* SUT also have double memory capacity. The TeaStore image provider is deployed eight times on each bare-metal SUT in Docker containers, including application stack monitoring. Each Docker container hosts an Apache Tomcat application server running the image provider service. We selected eight instances of the image provider and limited the CPU usage to one core via Docker, which is the largest deployment runnable on all SUTs without constraints. This setup also eliminates possible accumulation errors due to multithreading. Using the thread count as the maximum for each SUT resulted in memory contention on the large SUT and subsequently a minimal ability to process any requests on time.

[3] Apache JMeter: `https://jmeter.apache.org/`

Table 10.1: Servers used as SUTs.

SUT	CPU (Cores/Threads)	Memory
Small (S)	E3-1230 v5 @ 3.40 GHz (4/8)	16 GB
Medium (M)	E5-2640 v3 @ 2.60 GHz (8/16)	32 GB
Large (L)	E5-2650 v4 @ 2.20 GHz (12/24)	32 GB

All SUTs use the same load profile with four different load levels shown in Figure 10.3. This load curve was recorded for the *small* SUT but the *medium* and *large* SUTs are similar. Variations at the highest level are due to limitations on our testbed present on all SUTs. The load driver repeatedly requests random images with sizes ranging from 800 to 950 pixels in width and height from all application instances.

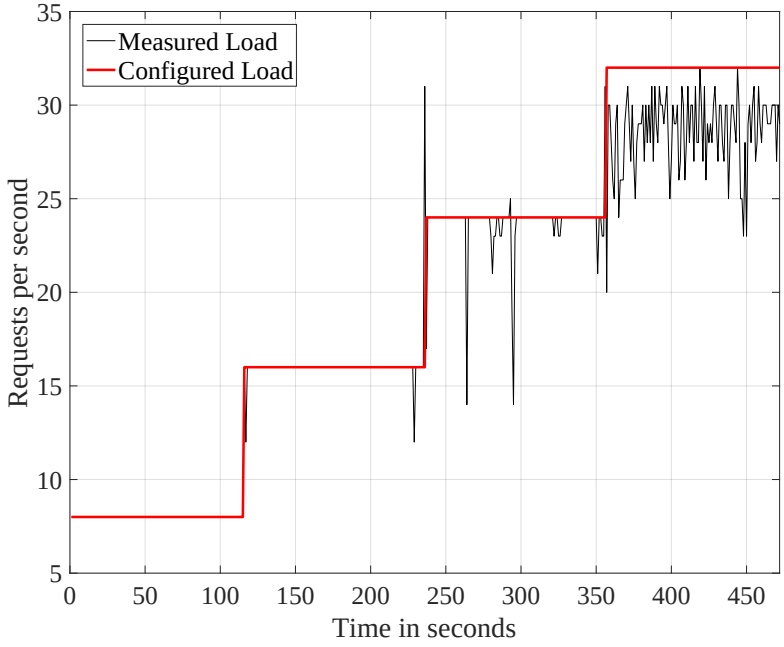


Figure 10.3: Configured and measured requests per second.

To build our regression model mentioned in Section 10.1, we stress the small SUT with the four different request rates and repeat this measurement nine times. We use six measurements as training data for our model and the remaining three for model validation. The coefficients are listed in Table 10.2.

Table 10.2: Regression parameters β for our power model.

Performance Event	β
Intercept	1.921
CPU_CLK_UNHALTED.THREAD	$-3.771 \cdot 10^{-9}$
IDQ_UOPS_NOT_DELIVERED.CORE	$-2.437 \cdot 10^{-8}$
UOPS_ISSUED_ANY	$-2.787 \cdot 10^{-9}$
UOPS_RETIRED.RETIRE_SLOTS	$6.270 \cdot 10^{-8}$
INT_MISC.RECOVERY_CYCLES	$7.338 \cdot 10^{-8}$
CYCLE_ACTIVITY.STALLS_MEM_ANY	$9.098 \cdot 10^{-9}$
RESOURCE_STALLS.SB	$6.996 \cdot 10^{-8}$

Chapter 11

Transferability of Characterization and Power Models

In this chapter, we describe our approach to evaluate if we can transfer our resource profile characterization approach from Section III. As retraining models is always linked to additional effort, reducing it to make our approach easier to use benefits software developers. Especially in a fast-changing DevOps environment. Next to our characterization, we also show that our function-level power consumption model in Section 10 can be reused on different SUTs. For the characterization, we train the model and evaluate it on a real-world application. The classifiers themselves were not trained or validated on the real-world application.

We also reuse the linear regression model for the function-level power consumption model and switch the underlying server hardware to two different systems. Switching these parts of the evaluation environment of our approaches from *Contribution 3*, and *Contribution 4* allows us to address our research question *RQ B.4* of *Goal B*.

We first describe our approach to transfer our resource profile characterization in more detail in Section 11.1, followed by our approach on transferring the function-level power consumption model across multiple servers in Section 11.2.

11.1 Classification

Transferring our resource profile classification approach is based on machine learning. As machine learning is known to handle data that it has not encountered so far well, we would expect that we can apply it to an unknown real-world application without any issues.

To transfer our approach, we first used our automated labeling approach in Section 8.4 to generate our ground truth for each second. The execution phases for the Elasticsearch application over 1000 s is shown in Figure 11.1. As the I/O

utilization, in the form of bytes read from disk and bytes written to disk, has an imbalance towards only one class, we did not include it.

We can see in Figure 11.1 that the memory also experiences less fluctuation as expected and stays at the *medium* class during the whole runtime. The exception is a warmup behavior in the first few seconds before the memory utilization settles. Also visible is the CPU utilization is more dynamic and switches throughout the execution between the *medium*, *high*, and *very_high* class. Only a few dips into the *low* CPU utilization class can be observed at the start and around the 500 and 600 second mark.

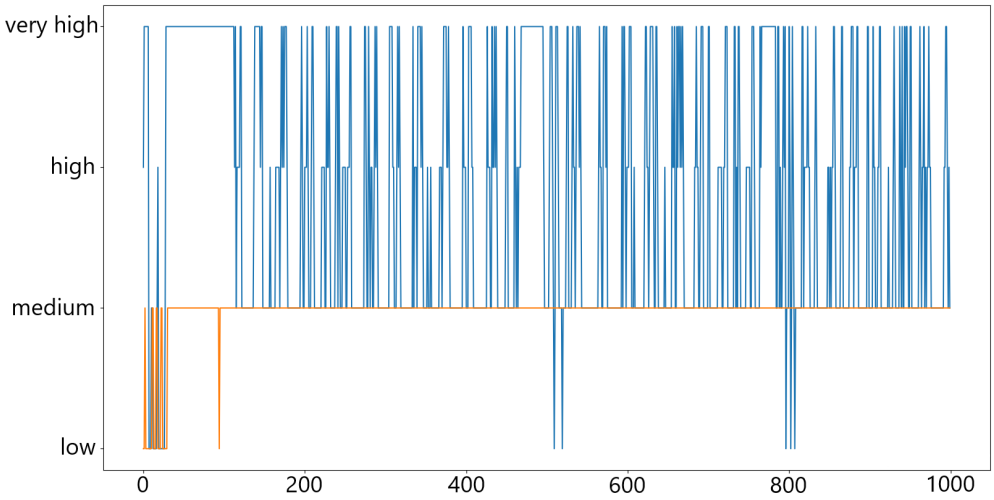


Figure 11.1: Execution phase classes for the CPU (blue) and memory (orange) of the Elasticsearch application.

After we established the ground truth, we trained two power consumption models based on FFNNs, which we described in Section 2.5.2. We used FFNNs as they can capture non-linear behavior compared to standard linear regression models. It, therefore, poses a more significant challenge to increase the accuracy of a single power model. Instead of using only the Phoronix Test Suite in our classification, we trained the FFNNs with additional benchmark runs from the stress-ng benchmark suite. The stress-ng benchmark runs are listed in Table 2 as our base model. The second FFNN is trained on the stress-ng benchmark runs only. In our evaluation, we then compare the accuracy if only the first FFNN is used and if we use both FFNNs on different classes.

11.2 Function-level Power Model

As the software is indirectly controlling the hardware, we assume that performance events occurring on one system are also observable on others. Hence, it stands to reason that the model might be transferable across different SUTs.

While the absolute power consumption in Watts is most likely not comparable, we also normalize the power prediction of the model to see if our approach is usable as a reference even without retraining the model. We then compare the normalized and absolute predictions of the trained model of the small SUT to the untrained medium and large SUTs in Table 10.1. While all SUTs use CPUs from the same manufacturer, they differ in core and thread count, clock speed, and available memory. We do not normalize over the measured load because not every request must necessarily call all functions.

Additionally, we also test if the same functions with the highest power consumption can be identified on all three SUTs without retraining our power model. Identifying the most power-hungry functions allows us to switch the development hardware without collecting additional measurements on the new system and retraining the model.

11.3 Concluding Remarks

In this part we introduced a new high-bandwidth memory model in Chapter 9 that is also usable together with our classification approach in Part III. This model allow us to either use relative metrics if the hardware information is not available or we can parameterize it and gain the absolute power consumption values. Therefore, our *Contribution 5* addresses the research question *RQ B.2* ("How to model high-bandwidth memory applications with system-level metrics?"). As application wide models are often too coarse to give valueable insights apart from comparing alternative applications, we proposed a new function-level power model in Chapter 10. It is based on performance events and can attribute them to functions according to a stack trace without instrumenting the source code itself. The function-level power model, *Contribution 4*, addresses our research question *RQ B.3* ("How to determine which function in an application consumes which amount of power and energy?"). Finally, we proposed the evaluation of our classification approach and function-level model if they are used on different software and hardware respectively without retraining for their new environment in Chapter 11, thereby addressing our research question *RQ B.4* ("CCan power models be reused for different servers and applications? This question is twofold. First, can we transfer a functional

power model across different servers; second, can a machine-learning based classification for power models be transferred across applications without re-training?").

Part V

Power Consumption Emulation

Chapter 12

Baseline and Performance Event Throughput

We developed PET to test applications that rely on external dependencies, mainly for network-driven applications. PET resolves these dependencies by emulating the power consumption of an application instead of executing the application itself with all its dependencies. The emulation is achieved by triggering the same performance events in the same frequency as the original application would do. It triggers the performance events through the use of synthetic code that is specifically built to cause a certain event [SKK17a; Kis19]. A more detailed description of PET can be found in Section 2.3.2.

However, each execution of a performance event trigger function in PET causes unwanted side effects. Side effects are performance events that are emitted but should not be explicitly triggered. For example, a level 3 cache hit event cannot be triggered without first causing a level 2 cache miss. PET can be configured to either ignore side effects, called naive method in previous publications [SKK17a; Kis19], or use one of the two built-in features to compensate for them, accumulation, and simulated annealing. While the two compensation mechanism have their benefits and drawbacks, the main challenge is to configure them for PET to work accurately. This complicated configuration makes PET cumbersome to use without expert knowledge.

Additionally, the execution time for a trigger determines the throughput with which events can be triggered. This event throughput, subsequently, also determines the upper limit for the emulation. If PET cannot reach the performance event throughput of the original application, its accuracy might degrade.

We address these challenges with a new configuration approach to automatically calibrate the number of performance events to trigger within the limits of PET, and a new implementation for the instructions retired performance event trigger that can achieve higher throughput. This *Contribution 6*, therefore, addresses our research question *RQ B.5* in *Goal B*.

We first describe the testbed with the SUTs that we use in Section 12.1, followed by determining the idle and maximum performance event rate, the achievable throughput, in Section 12.2. We describe the possible improvements for PET in Chapter 13.

12.1 Testbed

First, we describe the used testbed and measurement methodology in this section. As we use Chauffeur WDK, as described in Section 2.4.3, the setup of our testbed is similar to Figure 2.1. The three workloads Pi, XMLValidate, and SSJ, as described in the Foundation Section 2.4.3. The execution is controlled through the additional control system.

We selected three SUTs that differ in age, core and thread count, clock speed, level 3 cache size, and memory size, shown in Table 12.1. These three SUTs allows us to evaluate novel performance event triggers and new configuration approaches work on different systems.

To determine the maximum throughput and utilization, described in Section 12.2, the three workloads are run at $100\,\%$ load, the maximum amount of transactions that can be executed without delay and without queuing transactions. For the evaluation, we run the three benchmark runs from $10\,\%$ load up to $100\,\%$ load in $10\,\%$ steps. We record the seven performance events described in Section 2.3.2. Reiterating, these performance events are: level 3 cache misses, level 3 cache hits, bytes read from memory, bytes written to memory, instructions retired, hardware interrupts, and context switches.

Table 12.1: SUTs.

SUT	CPU (Cores/Threads)	L3 cache size	Memory
A	E3-1230 v5 @ 3.40 GHz (4/8)	8 MB	16 GB
B	E5-2640 v3 @ 2.60 GHz (8/16)	20 MB	32 GB
C	E3-1230 v2 @ 3.30 GHz (4/8)	8 MB	16 GB

12.2 Idle and Maximum Performance Event Rate

Before improving PET's accuracy, we need to know which performance event trigger does not perform as intended. We, therefore, determine the achievable throughput of performance events. Measuring the maximum throughput

is necessary to estimate which performance event trigger of PET should be improved because it cannot reach the necessary throughput to emulate the three Chauffeur WDK workloads. The maximal achievable throughput is calculated as the difference between the maximum performance event trigger rate and the idle performance event trigger rate. All measurements were taken as described in Section 2.3.1.

The idle performance event trigger rate is measured while the SUT is not under load, but all software artifacts necessary to execute the measurement are active. Thus, the idle rate includes the operating system, performance event sampling, and Chauffeur WDK executing the measurement. To simulate an idle system while at the same time recording performance events that would be emitted during a regular benchmark run, we continuously executed a sleep command with randomly selected sleep times between $10\,\mathrm{ms}$ to $20\,\mathrm{ms}$.

We measured the maximum performance event trigger rate for each event by running each trigger function in isolation for a duration of $240\,\mathrm{s}$. We set the number of performance events per transaction so that Chauffeur WDK reaches close to $1\,000\,000$ transactions over the measurement duration. Measurements are taken on all three SUTs. This is determined by the maximum number of performance event trigger rates a SUT can reach.

The utilization is determined by first running all three workloads. Then, out of all three runs, each event's highest performance event count is selected. We then divide the selected event count by the maximum achievable event trigger rate.

Table 12.2: Deviation and utilization of PET performance event triggers.

Trigger	Deviation	Utilization
Context Switches	$-16.01\,\%$	$40.53\,\%$
Hardware Interrupts	$1.14\,\%$	$0.76\,\%$
L3 Cache Misses	$-99.72\,\%$	-
L3 Cache Hits	$-32.97\,\%$	$128.98\,\%$
Bytes Read from Memory	$-17.23\,\%$	$167.19\,\%$
Bytes Written to Memory	$-17.10\,\%$	$54.90\,\%$
Instructions Retired	$-15.69\,\%$	$146.29\,\%$

Table 12.2 shows the average deviation from the configured trigger rate to the measured rate. Detailed results can be found in Table 4. Most performance event triggers miss their configured target value by about $17\,\%$ with two visible exceptions. First, the hardware interrupts, and second, the level 3 cache misses.

While the hardware interrupt trigger performs well, the level 3 cache misses deviate significantly. This behavior was reproducable on all three SUTs. We, therefore, remove the level 3 cache miss trigger as non-functional from the evaluation. Also visible in Table 12.2 is that three performance event triggers are not capable of reaching the necessary throughput by showing utilization values above 100 %: level 3 cache hits, bytes read from memory, and instructions retired.

Chapter 13

Improvement Potential

We have determined the maximum achievable throughput for each performance event trigger. Based on this analysis, we can determine which event trigger is suitable for improvement in Section 13.1. Additionally, we already identified that PET is cumbersome to configure without expert knowledge. Hence, we propose a new automatic configuration approach in Section 13.2.

13.1 Performance Event Triggers

The three identified performance event triggers that could be improved are the level 3 cache hits, the bytes read from memory, and the instructions retired.

Bytes Read from Memory At first glance, the bytes read from memory event trigger with the utilization of $167.19\,\%$ seems the most suitable trigger to improve. The performance event trigger only uses a memory bandwidth of $5.75\,\mathrm{GB/s}$. The theoretical peak bandwidth of the slowest SUT A is $17.1\,\mathrm{GB/s}$, higher than the used bandwidth. However, improving this trigger by increasing the usable bandwidth with the same technique as the level 3 cache hits by randomizing the access pattern to miss the cache on purpose would introduce many side effects. The current implementation does not cause large amounts of side effects due to uncachable memory allocated with `kmalloc`[1]. More side effects would possibly cause PET to become more inaccurate. As we already have determined that most applications are CPU-heavy in our application classification in Section 8.3.1, we do not change the bytes read from memory event trigger.

Level 3 Cache Hits The level 3 cache hit event trigger has a utilization of $128.98\,\%$. The trigger's low performance stems from generating random offsets when accessing memory addresses that avoid the level 2 cache prefetcher in

[1] `https://www.kernel.org/doc/htmldocs/kernel-api/API-kmalloc.html`

modern CPUs while at the same time are highly likely to be in the level 3 cache. We have not found an implementation that performs better without increasing the number of performance events emitted as side effects.

Instructions Retired The instructions retired event trigger has the second-highest utilization of 146.29 %. This performance event trigger uses a simple for-loop that increments a counter by a constant, as shown in Listing 13.1. This implementation cannot saturate the CPU pipeline due to the dependency of the increment operation, resulting in a low IPC of 1.58 on SUT A with a theoretical IPC of 3 for Streaming SIMD Extension (SSE) instructions. Increasing the IPC and, therefore, the performance event trigger throughput holds the most potential for improvements.

Listing 13.1: Old implementation of the instructions retired event trigger.

```
1 for (uint64_t i = 0; i < count; i++) {
2         x += CONSTANT;
3 }
```

Hence, we propose a new implementation for the instructions retired performance event trigger based on SSE instructions. All available SSE registers are used to saturate the processor's pipeline and prevent compiler optimizations from removing seemingly unused instructions. The code in Listing 13.2 shows the new implementation for this performance event trigger. It uses 14 integer add instructions utilizing 15 of the 16 available registers compared to the old implementation in Listing 13.1.

Listing 13.2: SSE implementation of the instructions retired event trigger.

```
1  for (uint64_t i = 0; i < count; ++i) {
2      xmm0 = _mm_add_epi32(xmm0, xmm1);
3      xmm2 = _mm_add_epi32(xmm2, xmm3);
4      xmm4 = _mm_add_epi32(xmm4, xmm5);
5      xmm6 = _mm_add_epi32(xmm6, xmm7);
6      xmm8 = _mm_add_epi32(xmm8, xmm9);
7      xmm10 = _mm_add_epi32(xmm10, xmm11);
8      xmm12 = _mm_add_epi32(xmm12, xmm13);
9      xmm1 = _mm_add_epi32(xmm1, xmm2);
10     xmm3 = _mm_add_epi32(xmm3, xmm4);
11     xmm5 = _mm_add_epi32(xmm5, xmm6);
12     xmm7 = _mm_add_epi32(xmm7, xmm8);
13     xmm9 = _mm_add_epi32(xmm9, xmm10);
14     xmm11 = _mm_add_epi32(xmm11, xmm12);
15     xmm13 = _mm_add_epi32(xmm13, xmm14);
16 }
```

13.2 PET Configuration

This thesis proposes a new configuration approach. Currently, PET has the option to ignore side effects and two options incorporating side effects, all with their drawbacks. We described the side effect avoidance in more detail in Section 2.3.2 but shortly reiterate. Ignoring side effects will decrease accuracy; the simulated annealing approach is cumbersome to use due to several hyper-parameters that influence its results and does not always perform well with high deviations in power consumption. For comparison, for the XMLValidation workload, ignoring side effect has a mean power consumption deviation of $-10.48\,\%$ while the simulated annealing has a mean deviation of $-52.57\,\%$. The accumulation approach counts the number of performance events emitted as side effects and subtracts them from the number of performance events to trigger [SKK17a]. However, the number of side effects per triggered performance event can change on different hardware and, therefore, needs expert knowledge about the underlying hardware. Hence, we propose a new automatic calibration approach that is easier to use and improves accuracy. We compare our novel approach to the naive option and a second approach that we call *expert knowledge*. The *expert knowledge* approach in our comparison should show what is achievable with PET in case an expert configures it.

Our approach needs only three measurements, the first for the performance event profile of the application to emulate, the second measurement for the maximum throughput for each performance event trigger, and the idle measurement. While the idle and maximum throughput for a performance event trigger can also be machine-dependent as the accumulation approach, no expert knowledge about the CPU, memory, and I/O subsystem is necessary. All three measurements can be automated to obtain a self-configuring approach for PET.

Equation 13.1 shows the ratio of performance events to trigger relative to the maximum achievable throughput for that trigger. Given a performance event trigger n, we subtract the events occurring in the idle state $T_{n,0}$ from the application profile $T_{n,app}$ and maximum throughput $T_{n,max}$. As shown in the previous Section 12.2, a trigger might not be able to reach the necessary event trigger rate. We, therefore, set the ratio to 1 in case the maximum throughput is lower than the application profile. We then calculate the absolute number of performance events to trigger in Equation 13.2 where $b_{n,max}$ is the maximum events per transaction defined in the maximum throughput run.

$$T_{n,\%} = \min\left(\frac{T_{n,app} - T_{n,0}}{T_{n,max} - T_{n,0}},\ 1\right) \tag{13.1}$$

$$T_n = \frac{T_{n,\%}}{\sum_{i=1}^{N} T_{i,\%}} \cdot b_{n,max} \tag{13.2}$$

13.3 Concluding Remarks

We have addressed our research question *RQ B.5* by proposing a new performance event trigger function and a novel configuration approach to accommodate side effects. The new event trigger function uses SSE calls, available on most modern CPUs, to increase IPC and achieve a higher performance event trigger rate (throughput). Our new configuration approach does not need expert knowledge of the underlying hardware and relies only on measurements that can be automated.

Part VI

Evaluation

Chapter 14

Goals

The goal of this chapter is to investigate our *Goal B* introduced in Section 1.3. This goal was set to help developers to improve the energy efficiency of their applications. We proposed an approach to classify applications and execution phases of an application to select a more accurate power model. This classification accommodates the development style in a DevOps environment with rapid changes of the application itself without the need for constant retraining of the power model, answering *RQ B.1*. We also proposed a novel high-bandwidth memory model based on system-level metrics to have a good selection of power models. High-bandwidth memory applications are among the most power-consuming applications [Kis+15b]. We also proposed a novel function-level power model to allow deeper insights into an application, aiding developers to identify the most power-consuming functions and, therefore, optimal candidates for improving energy efficiency. This contribution answers our research question RQ B.3.

The actual hardware an application gets deployed on can change quickly as well. When a new version is deployed in a new virtual machine or container or the cloud computing provider migrates the virtual machine to a different physical machine without the user's knowledge. This change in hardware might introduce inaccuracies to the power consumption prediction, and we investigate if our function-level model can work on different servers without retraining. Additionally, our classification approach should be able to handle unknown applications to raise the developer from the burden of retraining complicated machine learning techniques that could also be time-consuming, answering our research question *RQ B.4*.

In case the developer wants to select the hardware, for example, due to constraints or necessary auxiliary acceleration hardware, we also proposed to improve PET's emulation accuracy and evaluate it on more server hardware as it has only been tested on a single SUT [SKK17a; Kis19]. PET allows developers to emulate the power consumption of their application without any dependencies except a profile of performance events that need to be triggered. Showing

that PET works on more than one server, reducing its configuration overhead, and improving its synthetic code that causes performance events, gaining developers more insight into how their application behaves on a server before deployment. This contribution answers our research question *RQ B.5*.

To recapitulate, our research questions for *Goal B* are:

- ***RQ B.1*:** How to classify applications in an automated manner according to their resource usage profile in order to derive suitable and, therefore, more accurate power models?
- ***RQ B.2*:** How to model high-bandwidth memory applications with system-level metrics?
- ***RQ B.3*:** How to determine which function in an application consumes which amount of power and energy? The answer to this question can aid developers in improving the energy-efficiency of the application.
- ***RQ B.4*:** Can power models be reused for different servers and applications? This question is twofold. First, can we transfer a functional power model across different servers; second, can a machine-learning based classification for power models be transferred across applications without retraining?
- ***RQ B.5*:** How can the behavior of an application emulation on production servers be improved to predict its power and energy consumption more accurately and possibly improve the energy efficiency at deployment?

First, we evaluate the resource profile classification in Chapter 15, followed by our modelling approaches and if they are transferable in Chapter 16. Next, we evaluate our improved version of PET in Chapter 17 and conclude this part with a summary in Section 17.4.

Chapter 15

Resource Profile Classification

This chapter investigates if the accuracy of our resource profile classification approach described in Part III. We evaluate two different settings. One setting where we classify an application according to its resource profile that is averaged over the complete runtime in Section 15.1, and one where we classify execution phases of a long-running application in Section 15.2. The first averaged resource profile is evaluated on the EC12 and the execution phase classification on the x86 server. This evaluation answers our research question ***RQ B.1*****:** "How to classify applications in an automated manner according to their resource usage profile in order to derive suitable and, therefore, more accurate power models?". Due to the imbalance in the I/O utilization, specifically the bytes read from disk, we used only three classifiers for the remaining three resource types, CPU, memory, bytes written to disk, as described in Section 8.4.

We use four common metrics in machine learning to evaluate our approach, *precision*, *recall*, *accuracy*, and f_1 *score*. To calculate these metrics, we need the following four measures:

- *True Positive* (TP): The number of samples that were correctly predicted to belong to the class.
- *False Positive* (FP): The number of samples that were incorrectly predicted to belong to the class.
- *True Negative* (TN): The number of samples that were correctly predicted to not belong to the class.
- *False Negative* (FN): The number of samples that were incorrectly predicted to not belong to the class.

The first metrics is *precision* as defined in Equation 15.1. It denotes the ratio between the correctly assigned samples of the class, compared to all samples that were assigned to the class, including the samples wrongfully assigned to it. The *recall* in Equation 15.2 is defined as the ratio between the correctly assigned

samples and the samples that belong to this class. Often both metrics are combined to the f_1-*score* shown in Equation 15.3. The f_1 *score* is high when both the *precision* and *recall* are high. The last metric is the *accuracy* in Equation 15.4. It denotes the ratio of correct predictions compared to all predictions.

$$\text{precision} = \frac{TP}{TP + FP} \tag{15.1}$$

$$\text{recall} = \frac{TP}{TP + FN} \tag{15.2}$$

$$f_1 = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}} \tag{15.3}$$

$$\text{accuracy} = \frac{TP + FN}{TP + TN + FN + FP} \tag{15.4}$$

15.1 Application Classification

As the first evaluation of our resource profile classification, we investigate the accuracy of three machine learning approaches, as described in Section 8.4. The evaluation setup is described in Section 8.1. For this section, we averaged the recorded performance events of the IBM EC12 LPAR for the complete runtime of the benchmarks of the Phoronix Test Suite. We trained the machine learning approaches based on our resource profile classes and the ANOVA.

We take a look at the CPU utilization, memory utilization, and I/O utilization in the form of the bytes written to disk. The CPU utilization classification in Table 15.1 shows that it works well across all four classes. Because the classification uses performance events from the CPU as input, we expect this behavior. We can observe that all three approaches have a lower f_1 score at the *medium* class. Although LogitNet, has a slightly lower f_1 score on the *high* class. A result that we would not expect as the *high* and *very_high* classes are the majority of observations, followed by the *low* class and visible in the CPU utilization histogram (Figure 8.2). The f_1 score for each class and the overall accuracy, as defined in Equation 15.4, show that XGBoost and Random Forest outperform LogitNet.

Regarding memory utilization, XGBoost and Random Forest outperform LogitNet again with higher accuracy and f_1 scores. Only on the *high* class, where all machine learning techniques struggle, is LogitNet equal or better than the other solutions. As the *low* and *medium* classes work reasonably well

Table 15.1: Classification results of the CPU utilization.

	XGBoost			Random Forest			LogitNet		
Class	prec	rec	f_1	prec	rec	f_1	prec	rec	f_1
low	0.83	1.00	0.91	0.82	0.90	0.86	0.82	0.90	0.86
medium	0.83	0.83	0.83	0.83	0.83	0.83	0.71	0.83	0.77
high	1.00	0.78	0.88	1.00	0.89	0.94	0.86	0.67	0.75
very_high	1.00	1.00	1.00	1.00	1.00	1.00	0.83	0.83	0.83
Accuracy		0.90			0.90			0.84	

while the *high* class is problematic, we reckon that the imbalance of the memory utilization, visible in Figure 8.4, causes the poor performance of all three machine learning techniques. Compared to the CPU utilization, all solutions have lower accuracy.

Table 15.2: Classification results of the memory utilization.

	XGBoost			Random Forest			LogitNet		
Class	prec	rec	f_1	prec	rec	f_1	prec	rec	f_1
low	0.81	1.00	0.89	0.85	0.96	0.90	0.75	0.88	0.81
medium	1.00	0.60	0.75	0.75	0.75	0.75	0.86	0.60	0.71
high	0.50	0.50	0.50	1.00	0.25	0.40	0.50	0.50	0.50
Accuracy		0.81			0.84			0.74	

The I/O utilization shows a similar picture to the memory utilization classification. The *low* class shows good results, the *medium* class reasonable results, and the *high* class poor performance on the f_1 scores. Additionally, the f_1 scores for the *high* class are even slightly worse than the memory utilization scores. Especially LogitNet is unable to correctly classify the *high* class for bytes written to disk. Regarding accuracy, as for CPU and memory classification, both XGBoost and Random Forest are outperforming LogitNet. We assume that the results are due to the imbalance of the I/O utilization that is even further skewed towards the lower classes than the memory utilization.

In conclusion, XGBoost and Random Forest are outperforming LogitNet for the resource profile classification. This result is expected as LogitNet is one of the more straightforward solutions. Because performance events capture the state of the CPU very well and a good variety of benchmark runs across the range of possible CPU utilizations. Additionally, the imbalance in memory, especially I/O-heavy applications, needs an improved approach or additional benchmark runs that stress these subsystems. This would improve the classifi-

Table 15.3: Classification results of the I/O utilization (bytes written to disk).

	XGBoost			Random Forest			LogitNet		
Class	prec	rec	f_1	prec	rec	f_1	prec	rec	f_1
low	0.88	0.96	0.92	0.85	0.96	0.90	0.88	0.91	0.89
medium	0.60	0.75	0.67	0.75	0.75	0.75	0.50	0.50	0.50
high	1.00	0.25	0.40	1.00	0.25	0.40	0.00	0.00	0.00
Accuracy		0.84			0.84			0.74	

cation tasks for memory and I/O and improve the selection of the correct power model for the classified resource profile. Nevertheless, the CPU classification task, classifying the CPU that typically has the most consuming and dynamic power consumption, works well.

15.2 Execution Phase Classification

To accommodate long running service in cloud computing that have changing behaviors due to different service requests, we also evaluate our classification approach on the execution phases. In our approach, we do not use change point detection, common in time-series analysis [AC17], but an execution phase is one second during the execution of the application. We, therefore, trained our machine learning approach XGBoost with each sample recorded during the execution of the benchmark runs described in Section 8.1 on the x86 server. We use XGBoost for this evaluation as it is one of the better performing machine learning solutions from classifying full applications. Due to the imbalance of the dataset for I/O utilization, we used the x86 server with only two classes.

Table 15.4 shows the results on the test set. We can observe from the f_1 scores that the edge classes *low* and *very_high* are classified well with scores over 0.90 for the CPU utilization. However, the *medium* and *high* classes are not well classified. Especially the *medium* class cannot be classified with our trained XGBoost.

The same is true for the memory utilization. The classes *medium* and *high* cannot be predicted while the majority class *low* has an f_1 score of 0.99. Analyzing the I/O utilization shows an analogoues result where only the *low* class is correctly classified.

We analyzed the behavior of the trained XGBoost again on the actual training set with the results listed in Table 15.5. We can observe that the CPU utilization classifier also struggles with the *medium* and *high* classes although, as expected, not to the extent as on the test set. The memory and I/O utilization show reason-

Table 15.4: Classification results of XGBoost on the test set.

	CPU			Memory			kB Written		
Class	prec	rec	f_1	prec	rec	f_1	prec	rec	f_1
low	0.98	0.85	0.91	0.98	1.00	0.99	1.00	1.00	1.00
medium	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.01	0.01
high	0.21	0.08	0.12	0.00	0.00	0.00			
very_high	0.98	0.99	0.98						

able results but only on the training set, and we would expect the classification of the *medium* and *high* classes to achieve higher f_1 scores. If we take a look at the sample distribution among the classes that were used for training, out of 401 796 samples, the *low* classes for all three resource categories has the largest sample size by a large margin with at least 300 000 samples. The *medium* and *high* classes are underrepresented.

Table 15.5: Classification results of XGBoost on the training set.

	CPU			Memory			kB Written		
Class	prec	rec	f_1	prec	rec	f_1	prec	rec	f_1
low	0.99	0.99	0.99	0.99	1.00	1.00	1.00	1.00	1.00
medium	0.65	0.48	0.55	0.89	0.63	0.74	0.88	0.64	0.75
high	0.67	0.65	0.66	0.87	0.91	0.89			
very_high	0.93	0.95	0.94						

Table 15.6: Samples for CPU, memory, and I/O utilization in the training set.

	CPU	Memory	kB Written
low	315 813	387 281	397 747
medium	9422	7940	4049
high	14 474	6575	
very_high	62 087		

We can observe from the results that some classes suffer from a low number of samples that do not benefit machine learning. Still, the CPU utilization classifier can distinguish high and low utilization. We assume that additional training data for the intermediate classes are necessary to improve the results.

Chapter 16

Modelling Approaches

We investigate the accuracy of our power models, the high-memory bandwidth model in Section 16.1, and the function-level power model in Section 16.2. Our power models should give developers the necessary information to improve their application in terms of energy efficiency and allow our classification approach to select a suitable model for memory-intensive applications. The accuracy of our power model is defined by the deviation or error from the ground truth, measured with a power analyzer. With this evaluation, we answer our following research questions:

- ***RQ B.2*:** How to model high-bandwidth memory applications with system-level metrics?
- ***RQ B.3*:** How to determine which function in an application consumes which amount of power and energy?

16.1 High-Memory Bandwidth Model

After we classified the resource profile, a more suitable power model can be selected. As many CPU-based power models exist, we focus on predicting power consumption for high-bandwidth memory workloads.

We predict power consumption for memory operations using different data types and strides. We define *stride* as the step size for each iteration over a contiguous block of memory. We compare the accuracy of comparative predictions (relative accuracy) using default hardware parameters and absolute prediction accuracy using hardware parameters for a specific system.

We evaluate the relative and absolute prediction accuracy of our memory model using measurements of nine modified versions of the stream benchmark [McC95]. Stream is a high-memory-bandwidth intensive benchmark that performs sequential memory accesses for scalar multiplication and copy operations on three large arrays. The benchmark was modified to iterate over

multiple arrays in parallel. Stream was also modified to be used according to the SPEC power and performance benchmarking methodology [KLK20; SPE14]. To this end, we execute stream in parallel using 16 system processes, each with separate arrays. The array iterations are repeated with the number of iterations per second serving as a throughput metric. Iterations are repeated for the entire measurement duration.

In adherence to the SPEC methodology [Hen19b], we run the workload for $15\,\mathrm{s}$ before beginning the measurement phase, which lasts for $120\,\mathrm{s}$, collecting throughput, system power (using an external power meter) and CPU performance events each second. CPU performance events are monitored using the IntelPCM tool [Int]. We measure CPU Power, Memory Power, and memory bytes read and written.

We modify the workload by changing the data type of the stream array (`char`, `int`, `double`, and `long`) and the stride of the array iteration (1, 2, 4, 8, 16, and 24). Differences in data types such as integer and floating point are not considered.

To determine the *inverseprefetcheraccuracy*, we select the stream workload using its standard data type `double` together with a stride of 16. We select a stride of 16 to accommodate normal software with a mixture of data with good spatial memory locality, benefiting from the hardware prefetcher and random access to main memory. The `double` data type with stride 16 is used as a training value and is therefore excluded from the evaluation. The machine onto which the model is trained is a HP DL160 Gen9 with a Xeon E5-2640 v3 ($2.6\,\mathrm{GHz}$) processor with $20\,\mathrm{MB}$ L3 cache LLC and $32\,\mathrm{GB}$ memory. The machine-specific *inverseprefetcheraccuracy* as the mean of both training values, full system power, and CPU power, is $170,535$. *inverseprefetcheraccuracy* for the trained model is obtained by measuring the L3 cache miss rate with IntelPCM.

As $P_{min,CPU}$ we use the LU benchmark from the SERT and SOR for $P_{max,CPU}$, also from the SERT. They are designed to stress the CPU with negligible memory access of 0.18KiB/s (LU) and 0.24KiB/s (SOR). The low memory access makes both benchmarks well suited to determine the maximum and minimum CPU power. All benchmarks are executed on the same machine used for the *inverseprefetcheraccuracy* shown in Table 16.1.

We use our model to predict the relative changes in power consumption. To only show the relative changes, we remove all machine-specific values from the model. $P_{min,CPU}$ and $P_{max,CPU}$ are set 0 and 1 respectively. The value for $cachesize$ and *inverseprefetcheraccuracy* are set to 1, removing all machine-specific impacts on our model. The resulting power consumption values, therefore, only rely on the data type and stride. The results, the measured full system power, and the predicted ordering for comparison are shown in Table 16.2.

Table 16.1: Benchmark results for the machine specific configuration of power consumption.

	P_{min}	P_{max}	$inversepre-fetcheraccuracy$
Full system power	$104.7W$	$140.6W$	$151,518$
CPU power	$63.81W$	$89.37W$	$189,552$

Table 16.2: Measured full system power compared to configured model and predicted ordering.

Data Type	Stride	Power Measured	Power Estimated	Ordering
`char`	1	$120.8W$	$140.31W$	8
`int`	1	$109.9W$	$139.43W$	32
`long`	1	$108.7W$	$138.26W$	64
`double`	1	$109.6W$	$138.26W$	64
`double`	2	$110.0W$	$135.26W$	128
`double`	4	$110.4W$	$131.26W$	256
`double`	8	$109.6W$	$121.92W$	512
`double`	16	$107.4W$	$103.23W$	1024
`double`	24	$105.8W$	$84.55W$	1536

As can be seen from our results, the model can keep the ordering if the $datasize * stride$ exceeds the cache line size or the size of the data type is small. Comparing the predicted ordering with the measurement results shows that the model can predict the ordering correctly if the difference in $datasize * stride$ is large. The model results also show a minor discrepancy between `long` and `double` stemming form different data types. Our model does not take different data types into account that might use different execution units of the CPU, resulting in different power consumptions.

To evaluate the prediction accuracy of our model, we measure the CPU power with IntelPCM while executing modified versions of the stream benchmark. The total system power consumption is measured concurrently with an external power meter. Both CPU and full system power consumption is estimated with the aforementioned *inverseprefetcheraccuracy* and compared to the measurement results.

The results presented in Table 16.3 shows that the model can predict the

CPU power consumption with an average deviation of $21.7W$ / 32.02%. As with the ordering, the best results are achieved if $datasize * stride$ exceeds the cache line size, with a deviation of -15.1%. For the full system power shown in Table 16.2, an average deviation of $23.5W$ / 19.34% could be achieved, with the lowest deviation using `double` together with a step size of 8.

Table 16.3: CPU power measured compared to configured model estimation.

Data Type	Stride	Power Measured	Power Estimated
`char`	1	$69.04W$	$89.14W$
`int`	1	$60.36W$	$88.44W$
`long`	1	$60.30W$	$87.52W$
`double`	1	$60.79W$	$87.52W$
`double`	2	$60.66W$	$85.67W$
`double`	4	$61.04W$	$81.95W$
`double`	8	$60.72W$	$75.58W$
`double`	16	$59.80W$	$59.80W$
`double`	24	$58.24W$	$45.02W$

The measured and estimated memory power are shown in Table 16.4. As idle power for the configured system, the IDLE workload from the SERT benchmark run is used with a full system power consumption of $36.4W$. The average memory power deviation is $0.36W$ / 3.97% with the lowest deviation at a step size of 8 with `double`. This result coincides with the CPU power estimation. With deviations from both the full system power and CPU power estimate propagating, a higher deviation than for other estimates is expected. The estimated memory power deviates with an average of $4.90W$ or 53.98% from the measurement.

Promising results at the edge cases with a high step size or small data type size stems from irregularities within our measurement results. While slight divergence from a linear model is expected, power consumption behaves non-linearly for minor differences in step size or data type size, shown in Figure 16.1, which subsequently can lead to the observed misordering and deviations in the estimation.

The figure also shows that further measurements are necessary to improve our model. Also, additional research for suitable reference benchmarks, more closely related to high-bandwidth memory workloads, is necessary to achieve better accuracy determining P_{min} and P_{max} for the CPU and full system power.

Table 16.4: Measured and estimated memory power consumption with an idle full system power of $36.4W$.

Data Type	Stride	Power Measured	Power Estimated
`char`	1	$9.13W$	$14.75W$
`int`	1	$9.11W$	$14.50W$
`long`	1	$9.12W$	$14.16W$
`double`	1	$9.14W$	$14.16W$
`double`	2	$9.10W$	$13.49W$
`double`	4	$9.12W$	$12.14W$
`double`	8	$9.09W$	$9.45W$
`double`	16	$9.04W$	$4.070W$
`double`	24	$8.99W$	$-1.32W$

Future work should also include a guideline for determining the *inverseprefetcheraccuracy* for predictions. Higher accuracy for the *inverseprefetcheraccuracy* parameter would also aid in avoiding mispredictions as seen in Table 16.4 for `double` with a step size of 24.

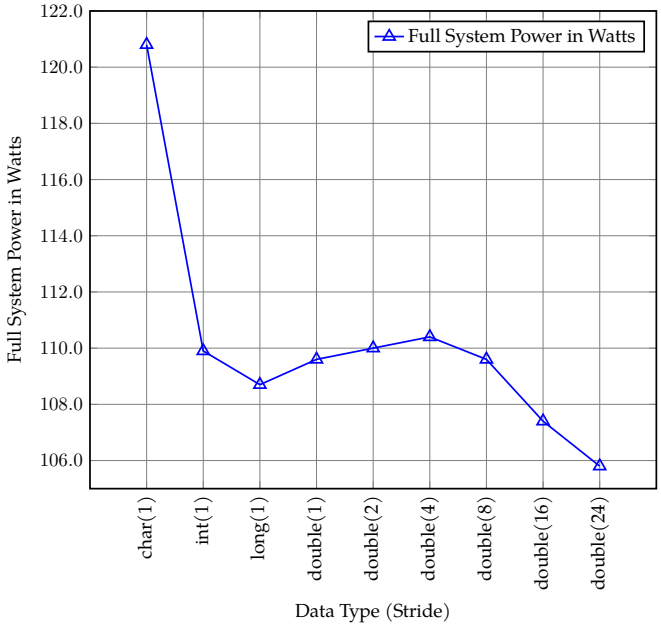


Figure 16.1: Measured full system power in watts for data types and strides.

16.2 Function-level Power Model

To help developers identify the power-consuming parts of an application, a novel function-level power model is necessary. We proposed a model based on system-level metrics that, with the addition of the stack trace, are elevated to application-level metrics by attributing them to the application's functions. The evaluation of our function-level model first separates the performance events, the system-level metrics, into the application and the overhead in Section 16.2.1. Then we evaluate the introduced correction factor in Section 16.2.2 followed by identifying the most power-consuming functions in the used reference application in Section 16.2.3. Finally, we discuss the limitations of our approach in Section 16.2.4.

16.2.1 Application Separation

To determine if our model can distinguish between power consumed by the application and overhead do we perform nine measurements on the *small* SUT. We apply our approach to the recorded time-shares and performance

events without the correction factor. The power is calculated for each function and summed. As $\mathbf{f}_t^T \leq 1$, not allocating all performance events towards the monitored application is likely. We consider this software stack overhead due to the Java Runtime Environment, Docker service, and the operating system.

Figure 16.2 shows that the power drawn from the overhead can make up a considerable amount. Notably, around 170 to 240 seconds for the TeaStore application where the overhead and application assigned power are close to equal. Neither the overhead nor the application can reach the observed wall power for the complete system as expected. If the overhead (P_o) and application power (P_a) are combined to the estimated dynamic power ($P_e = P_o + P_a$), the power consumption is close to the measured wall power of the SUT. Table 16.5 shows the mean and relative errors for the prediction.

Table 16.5: Error of the prediction absolute and relative to the measured values.

Requests per Second	Absolute Error	Relative Error
8	1.18 W	7.5 %
16	1.15 W	3.5 %
24	1.70 W	3.5 %
32	1.48 W	2.6 %

While the application power draw has rising steps in conjunction with the load level, the overhead shows different behavior. With the increasing load level, the overhead is also expected to grow with the number of calls to the operating system. The overhead climbs with the load level to 24 requests per second. As the load switches to 32 requests per second, the overhead drops sharply. We estimate that the non-linear overhead could be due to falsely assigned performance events, changes in scheduling, e.g., to avoid Java's garbage collection while close or at full CPU capacity, operating system interventions, simultaneous multithreading, or a combination of those. Further research is necessary to find the cause and determine if this results from performance events falsely assigned as overhead instead of the application's functions.

There is also a shift of 3 seconds between the model estimation and the actual power measured. We attribute this to certain inertia in the server system power supply as all monitoring probes start within $10\,\mathrm{ms}$.

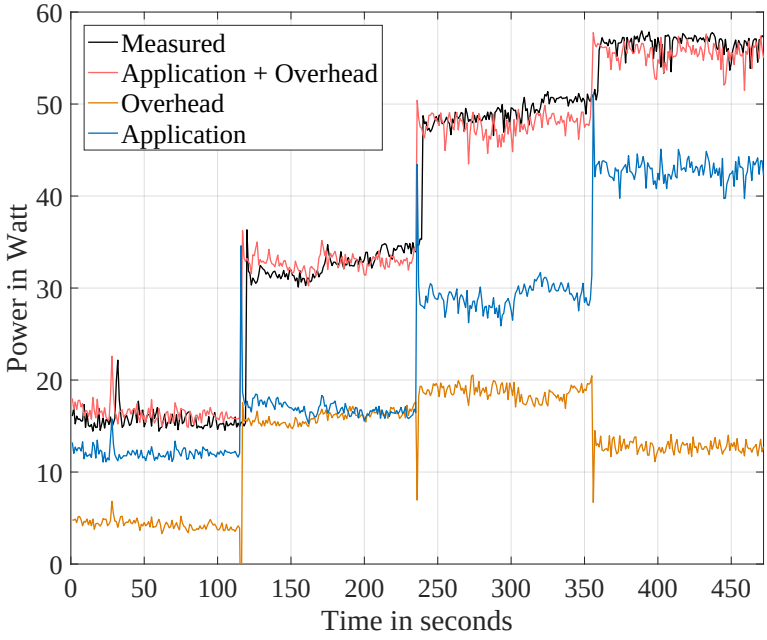


Figure 16.2: Mean predicted power consumption of the regression model over nine measurements and actual wall power for the *small* SUT.

16.2.2 Correction Factor

To check if our correction factor $\mathbf{c}_t$ (see Section 10.2.2) can dampen outliers, probably stemming from computations scheduled in between the test application, we calculated our model's power consumption with varying sizes for the moving average history. We let the size of the moving average range from $2\,\mathrm{s}$ to $60\,\mathrm{s}$ in steps of $2\,\mathrm{s}$. Against our expectations, the smoothing over the history only moved performance events from the application to the overhead but not vice versa. Hence, the correction factor in its proposed form is neglectable for further evaluation, and other possibilities should be taken into account.

16.2.3 Identifying Function Power Consumption

Our approach can identify the power consumption of a micro-service or serverless cloud application. As the sum of all functions' power consumption is accurate, we check each function's contribution to the overall power consumption.

As a first step, we summed up each function's contribution to the proportional power consumption over all load levels. Four functions are consuming over 99% of the total power attributed to the application out of in total 56 functions called during the measurements. The remaining functions only contribute marginally with one order of magnitude below *Function 12*, which is below 1%. The four functions and their relative energy consumption are:

- *Function 08* (67%): Converts the internal image representation of Java into a byte array and encodes it in base64 for the request response.
- *Function 12* (<1%): Creates a new object that can be stored in the cache holding the base64 encoded data.
- *Function 29* (1%): Converts the base64 encoded image to the Java image representation.
- *Function 47* (31%): Resizes images.

As the image provider has the least frequently used replacement strategy, it seems that the substantial impact of these four functions results from the requests selecting random images, causing a low cache hit rate. Hence, images must be converted back to the Java image representation, scaled, stored in the cache, and converted back to base64 for the request-response.

We further break down the contribution of each function to the power consumption by load level. Figure 16.3 shows the power consumption for the four functions. The sharp changes in the load level are not included to let the load driver stabilize. We cut off three seconds before and after each load change as the power consumption for each function rises with the load level indicating that our approach can correctly attribute a function's impact on the energy usage, variations in the measurement increase with the load level. This behavior matches with the random sizes requested by the load driver querying more images of different sizes, also coinciding with the increasing contribution of *Function 47*, scaling images to the correct size.

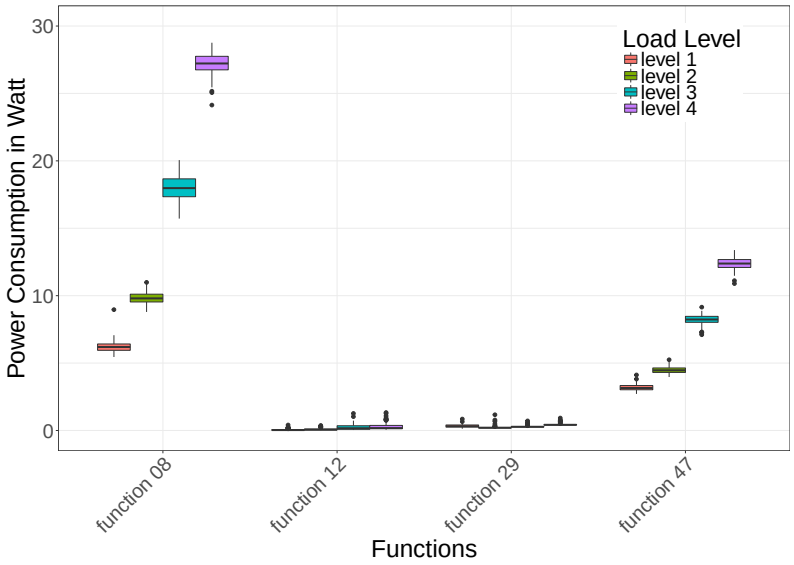


Figure 16.3: Function power consumption for four load levels.

16.2.4 Discussion and Limitations

In our work, we use the performance events pre-selected by Yasin [Yas14]. Yasin selected his performance events to detect performance bottlenecks as opposed to predicting a function's power consumption. This choice might impact the resulting quality of our approach. Further exploration of the performance event configuration space might yield a set of performance events even better suitable for our use case. Nevertheless, the presented results already exhibit high accuracy. Therefore, the used events are sufficient to confirm the applicability of our approach and are within the accuracy of typical power models. Since the performance events are a proxy metric, they represent the true values only to a certain degree. The same is true for RAPL counters, as described in Section 2.3.1. We opted for a self-trained performance event model to not rely on an opaque software model, not under our control. Additionally, the most influential power consumer in a server is the CPU. While AMD and Intel both have RAPL counters, others might not. To keep our approach portable and transparent, we do not use RAPL counters.

Serverless and virtualized environments often do not expose performance events. This issue can be resolved by recording the performance events on the host system and a dynamic mapping to VM or container. The available events or power consumption could then be made available to a tenant.

As we try to keep the overhead introduced by the performance event sampling minimal, we only collect what is necessary. We use a low sampling rate of one second, corresponding to the sampling rate of the power meter. As the evaluation shows, one second is enough to reach a reasonable accuracy. A higher frequency, e.g., on a function basis, may increase the accuracy but incur a higher overhead. A possible technical limitation of our evaluation is the timer resolution of the monitoring solution Kieker. While Kieker claims a resolution of one nanosecond, inaccuracies are possible. However, throughout our measurements, these inaccuracies should compensate for each other.

In this work, we assume that power consumption is linearly correlated to the runtime. However, idle phases like I/O waits might lead to a non-linear correlation. Micro-service workloads are often a mixture of CPU, memory, and I/O-based computations, but linear models have proven to have good accuracy. However, for I/O heavy workloads, other models could prove more accurate.

The applied error correction approach has shown no improvement to the uncorrected results. While, as presented above, our results are suitable for our use case, more complex error correction or smoothing algorithms might be beneficial.

16.3 Transferability of Classification and Power Model

We investigate how well our classification approach works on unknown real-world applications in Section 16.3.1 and if our function-level model predicts power accurately independent of the SUT in Section 16.3.2. It, therefore, answers our research question ***RQ B.4*****:** "Can power models be reused without training? This question is twofold. First, can we transfer a functional power model across different servers without remodeling; second, can a machine-learning classification for power models be transferred across applications without retraining?".

16.3.1 Classification

In addition to our previous evaluation of the classification approach, we investigate how well our approach is transferable to an unknown real-world application. This unknown application evaluation shows how well the selected machine learning approach, XGBoost, can abstract vital information for classification. It also shows if switching between different power models achieves a better result than just a single model.

We evaluate our real-world application Elasticsearch on the x86 server and use the already trained model for the execution phase classification. First, we established and described the ground truth in Section 11.1. The labels for each second of execution in the ground truth were done according to our automated labeling approach according to the defined resource utilization classes in Table 8.2 for the x86 server. Because the memory utilization class stays almost constant throughout the evaluation with the exception below $50\,\mathrm{s}$, it has no impact on the decision of which power model to choose.

Figure 16.4 shows the power consumption over time when we predict it with only a single FFNN power model trained on the Phoronix Test Suite and stress-ng benchmark runs. We can observe that the power model has problems with the lower CPU utilizations. This behavior is expected as the classification accuracy for the *medium* and *high* CPU utilization classes is low (see Table 15.4). As the SUT switches continuously between three classes, including the *medium* and *high* class, reduces its power prediction accuracy.

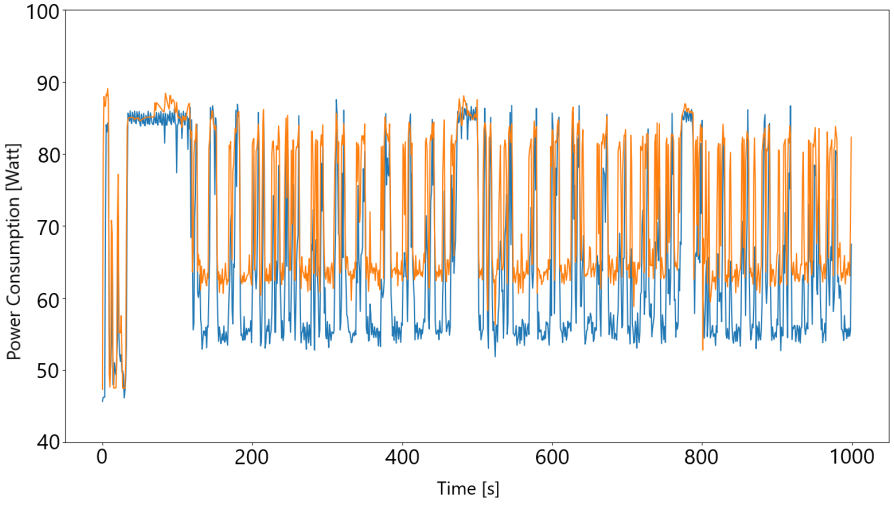


Figure 16.4: Measured (blue) and predicted (orange) power consumption of the x86 server with a single FFNN model.

We compare our results with a single FFNN power model with our approach using two FFNN power models. The first model is the original described before. We also trained a new power model solely on the stress-ng benchmark runs. The stress-ng benchmarks are more synthetic than the Phoronix Test Suite benchmarks. Therefore, we assume the second model will abstract better for classes where the first model struggles with the correct classification. We use

the first FFNN power model for the CPU utilization classes *high* and *very_high* while we use the second power model for prediction on the *low* and *medium* classes.

Figure 16.5 is showing the power consumption prediction for both models. It is visible in the figure that the lower power consumption where the CPU utilization is also lower can be predicted more accurately. However, the peaks at around 90 W show a higher power prediction than the measured value. The accuracy of our approach still increases from 12.65% to 9.71% as shown in Table 16.6.

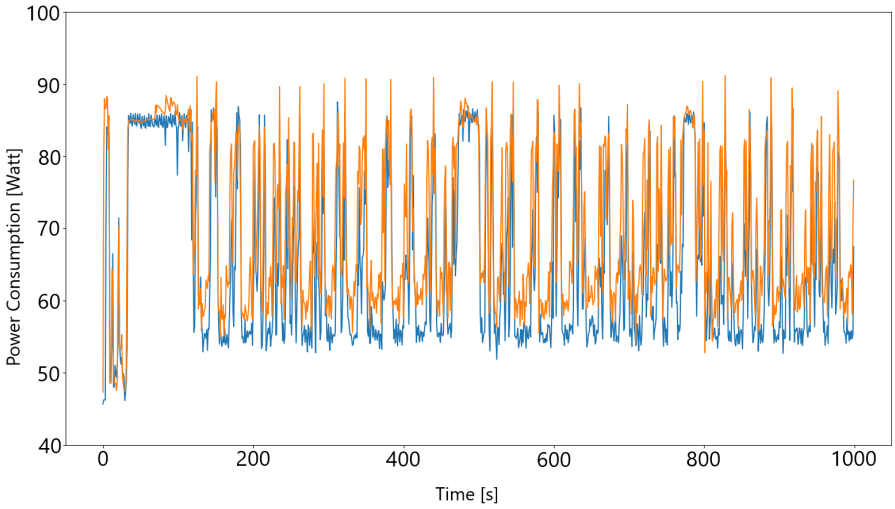


Figure 16.5: Measured (blue) and predicted (orange) power consumption of the x86 server with two FFNN models.

Table 16.6: Power consumption prediction results.

Model	MAE	MAPE
Single FFNN	7.5977	12.65
Two FFNN	5.9431	9.71

This result has two implications. First, our classification approach that selects more suitable power models works under the assumption that the power models are adequately trained for the classified resource profile. Second, we can transfer FFNN-based power models to predict the power consumption of

unknown applications with reasonable accuracy of near 13%, or 10% if our classification approach is used.

16.3.2 Function-level Power Model

As the software is indirectly controlling the hardware, we assume that performance events occurring on one system are also observable on others. We normalize the power estimation of our function-level power model (described in Section 10) to see if our approach is usable as a reference even without retraining the model. We then compare the normalized estimations of the trained model of the small SUT to the untrained medium and large SUTs. We do not normalize over the measured load. Not every request must necessarily call all functions.

Figure 16.6 shows the results for a single measurement of each SUT. The small SUT works well as expected, while the medium and large SUT exhibit large variations. Especially at the lowest and highest load. The same behavior is present for both SUTs in Figure 16.7, clearly visible at the large outliers at the highest load level. The experienced behavior is observed in the measurement as well as the relative application power. It, therefore, can not be completely attributed to the model being untrained on the medium and large SUT's data. The model also overestimates the power consumption for the large SUT. The measured power consumption in Figure 16.7 shows an ordering from large to small SUT for identical request rates, with the large instance mostly (88% of the time) below the medium instance.

Table 16.7: Mean absolute error, standard deviation and variance between the small, medium and large SUT.

		Small/Medium	Small/Large
Normalized	MAE	0.1547 (15%)	0.1495 (15%)
	σ	0.0975	0.1586
	σ^2	0.0095	0.0251
Measured	MAE	5.7683 (10%)	12.4118 (21%)
	σ	2.9736	4.5653
	σ^2	8.8421	20.8416

Calculating the absolute error between the trained *small* server and the *medium* and *large* servers, we can see in Table 16.7 that the power consumption deviates strongly from the mean value. While the relative application

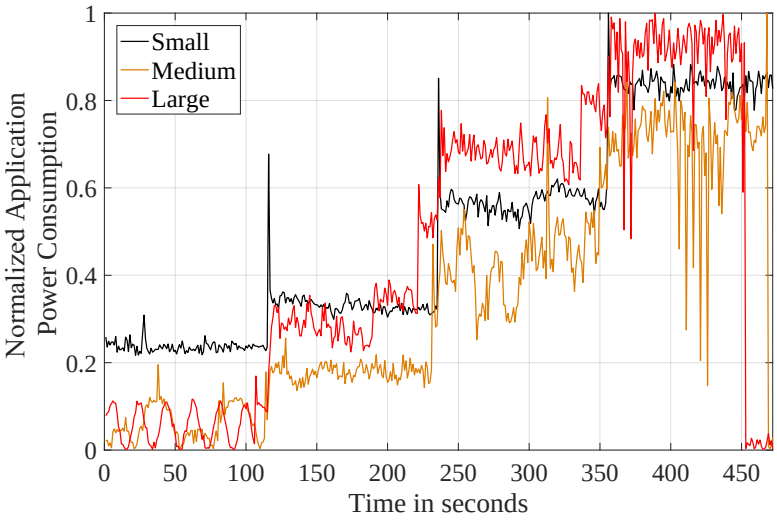


Figure 16.6: Normalized application power consumption of all three SUTs.

power seems unsuitable for comparison, we examine if the proportional power consumption remains portable.

In accordance with Section 16.2.3, we examine the proportional power consumption of the application functions in Table 16.8. Proportional accounting to functions works well, yet small deviations are present and were expected. The functions vary between 3% to 2% for *Function 08* and *Function 29*, respectively. The remaining functions account for 2% of the total power consumption of the medium SUT. We estimate that small changes in the CPU's architecture are responsible for the minor changes shown in Table 16.8.

Table 16.8: Proportional power consumption of functions for all SUTs.

	Functions				
SUT	08	12	29	47	others
Small	67%	$< 1\%$	1%	31%	$< 1\%$
Medium	64%	$< 1\%$	3%	31%	2%
Large	66%	$< 1\%$	2%	31%	$< 1\%$

We showed that our approach distinguishes between the power draw of our test application and the overhead of the operating system and software stack. The correction factor as a moving average did not yield the expected

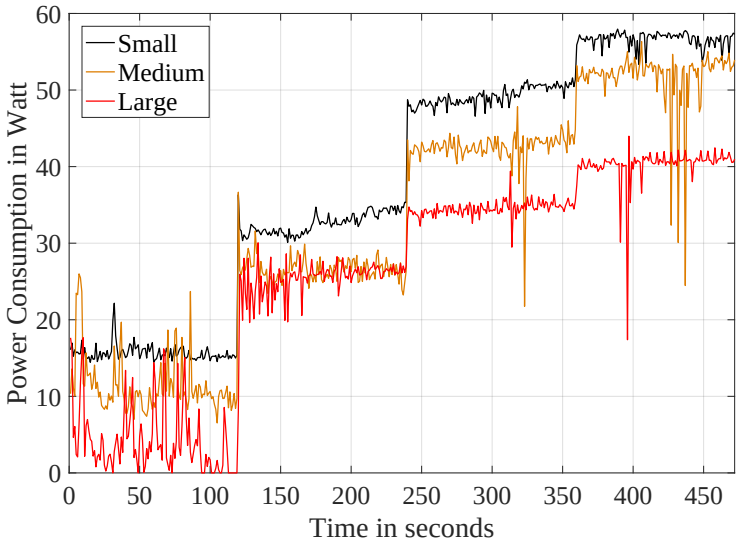


Figure 16.7: Measured application power consumption of all three SUTs.

improvements and is rejected. Further, we showed that allocating performance events to specific functions works well, and we can identify functions with a high impact on power consumption. It also achieves good results across multiple SUTs.

Chapter 17

Power Consumption Emulation

In the last chapter of the evaluation, we investigate the power consumption emulation of applications based on performance events. First, we analyze our novel approach for triggering instruction retired events in Section 17.1, followed by our proposed automatic calibration approach in Section 17.2. Finally, we evaluate the reproducibility of our emulation in Section 17.3 and summarize the complete evaluation in Section 17.4. With an improved performance event trigger and an automatic calibration, we address our research question *RQ B.5*: "How can the behavior of an application emulation on production servers be improved to predict its power and energy consumption more accurately and possibly improve the energy efficiency at deployment?".

17.1 Instructions Retired Performance Event Trigger

We first investigate the accuracy of our novel instructions retired performance event trigger function. We, therefore, use the same metrics deviation and utilization as described in Section 12.2, where the deviation describes the distance between the target number of performance events that should be triggered, and the utilization describes the event trigger's throughput relative to its maximum. The evaluation is performed on all three SUTs. For comparison, we used the same configuration as in Section 12.2.

The Table 17.1 shows the results for SUT A. Our novel implementation using SSE instructions to increase the IPC works as expected. We can see that the IPC has increased from 1.59 to its theoretical maximum of 3.01. The IPC is slightly higher than three because the instructions for the loop can be executed different Arithmetic Logic Units (ALUs) than the SSE instructions (SUT A has three SSE ALUs). The utilization of the novel implementation has nearly halved and can reach higher performance event target values. However, the deviation has not improved. The instructions retired performance event trigger shows a slightly higher deviation than the original implementation by about one percent.

Table 17.1: Comparison of implementation variants for the instructions retired performance event trigger on SUT A.

Trigger	Deviation	Utilization	IPC
Original	$-15.69\,\%$	$146.29\,\%$	1.59
SSE	$-16.91\,\%$	$77.18\,\%$	3.01

If we compare our results from SUT A to SUT B and C in Table 17.2, we can see that the deviation is almost constant across all three SUTs. The utilization, on the other hand, changes as expected with the SUT. The fact that SUT B has a utilization of over $100\,\%$ is due to its lower clock rate of $2.60\,\mathrm{GHz}$ compared to $3.40\,\mathrm{GHz}$ and $3.30\,\mathrm{GHz}$ for SUT A and C respectively.

Given the higher maximum throughput achievable with our novel implementation, we did not increase the accuracy of the performance event trigger. However, we can now emulate applications that demand a higher instructions retired throughput for emulation with higher accuracy. We also reckon that the constant deviation could be remedied with a constant scaling factor.

Table 17.2: Deviation and utilization of the new instructions retired event trigger on all three SUTs.

SUT	A	B	C
Deviation	$-16.91\,\%$	$-16.98\,\%$	$-16.98\,\%$
Utilization	$77.18\,\%$	$103.74\,\%$	$84.37\,\%$

17.2 Auto-Calibration

As the next step in our emulation evaluation, we investigate our novel automatic calibration approach. We compare the auto-calibration with the workload itself (reference), the expert knowledge configuration, and the naive approach ignoring side effects from the original publications [SKK17a; Kis19] on all three workloads described in Section 2.4.3.

We determined the ratio $T_{n,\%}$ of performance events to trigger with our automatic calibration approach defined in Equation 13.1. Table 17.3 shows that the Pi workload is only emitting a small range of events, mainly instructions retired, as expected, and context switches. The XMLValidate workload is also

CPU-heavy but also stresses the memory subsystem and caches. The SSJ workload is specifically stressful for the caches and memory, while the CPU load is lower than for Pi and XMLValidate.

Table 17.3: Performance event ratio $T_{n,\%}$ for the workloads.

Event	Context Switches	Hardware Interrupts	L3 Hits	Bytes Read	Bytes Written	Instructions Retired
Pi	47.80 %	1.64 %	0.56 %	0.11 %	0.14 %	49.75 %
XML	3.78 %	0.55 %	12.33 %	30.22 %	22.08 %	31.04 %
SSJ	0.19 %	0.52 %	38.65 %	38.65 %	13.95 %	8.03 %

Figure 17.2 shows the three workloads measured on SUT A, the identical system used in the original publication. The Pi workload in the naive configuration shows a dent between the 60 % and 80 % load level. The naive approach also breaks the expected monotony as the 60 % load level consumes more power than the 70 % load level. Our expert knowledge approach is very close to the Pi workload itself and only deviates visibly starting at the 70 % load level. Emulating the Pi workload by auto-calibrating PET with our novel approach shows a slightly higher power consumption throughout all but the 100 % load level.

The XMLValidate workload has the highest power consumption of all three workloads. We can see in Figure 17.2 that all approaches are below the reference power consumption. Our novel approach performs as well as configuring PET with expert knowledge. Both deviate by about 30 %. However, the naive approach performs worse, deviating up to 40 % at the highest load level.

The SSJ workload shows the lowest power consumption. Despite the deviations shown in Table 12.2, both the manually configured and the automatic configuration approach performs well. The expert knowledge approach shows a slight overestimation of power consumption between the 40 % and 90 % load levels. The automatic calibration consistently emulates a higher power consumption except at the 100 % load level. Also, it shows the same overestimation between the 40 % and 90 % load level. However, both are similar. The naive approach also emulates a higher power consumption up to the 40 % load level and plateaus, starting at the 60 % load level. It cannot reach the power consumption at higher load levels.

We further investigate the automatic calibration approach by calculating the relative error from the reference measurement. Figure 17.1 shows the box plot of the compared approaches. It is visible that the naive approach has the highest variation and a quarter of all measurements deviate more than 20 %

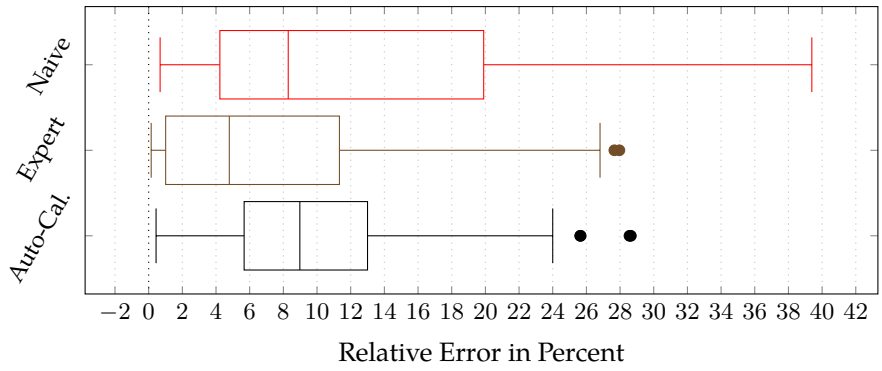


Figure 17.1: Relative error of the three different configuration approaches from the reference run.

and up to $40\,\%$. The expert approach shows a lower variation reaching up to $27\,\%$. It also shows the lowest median error of about $5\,\%$. Our automatic configuration approach has an even lower variation with up to $24\,\%$. At the same time, its median error is only marginally higher than the naive approach.

We have shown that our novel automatic configuration approach emulates the power consumption of the workloads closer than the naive approach. The automatic calibration also does not show unexpected behavior in the form of depression for the Pi workload and plateauing at the SSJ workload. The automatic calibration also significantly reduced variation, even below the expertly configured PET emulations, without degrading the accuracy of the emulation.

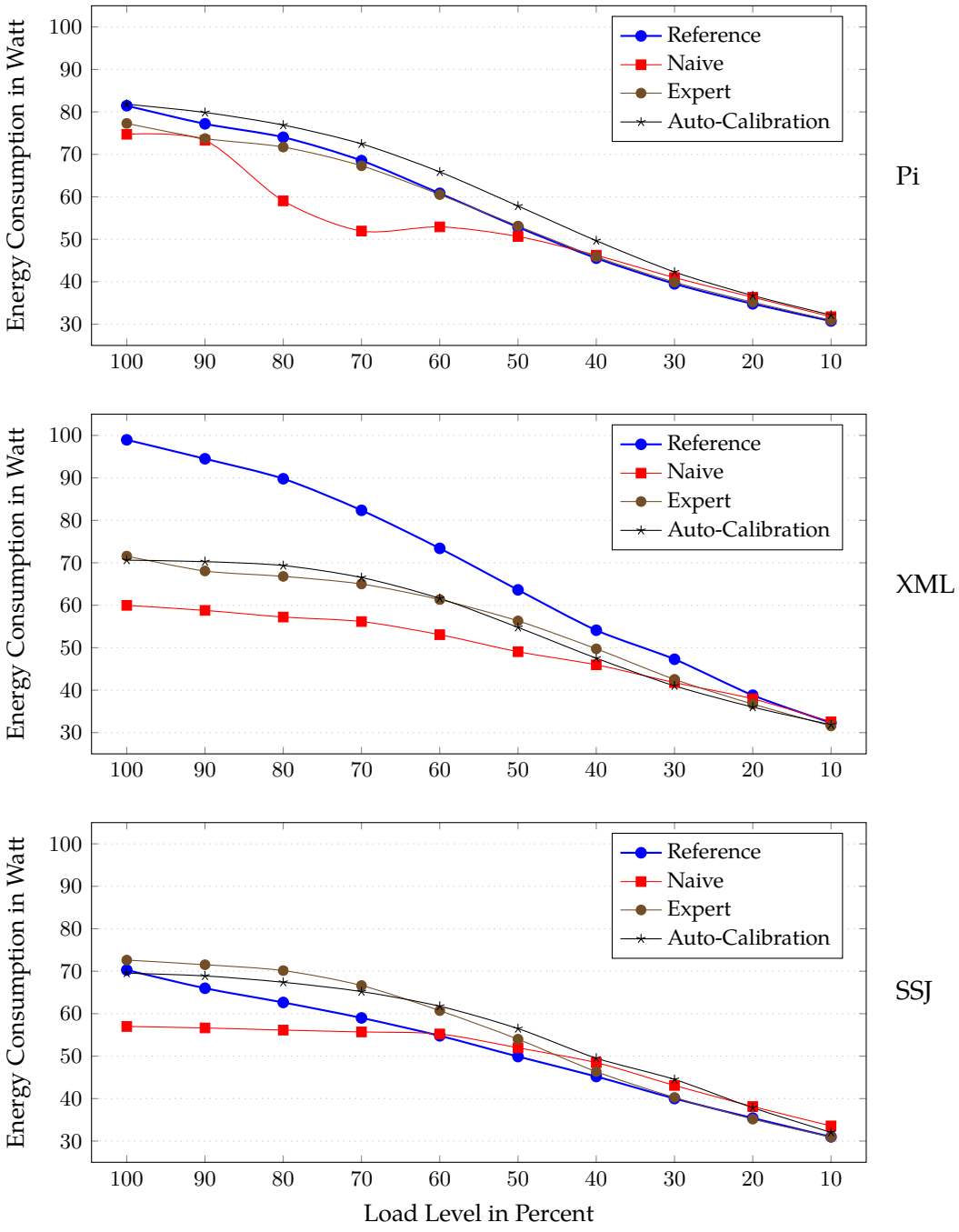


Figure 17.2: Comparison of the different calibration approaches on SUT A.

17.3 Reproducibility

Even though the Chauffeur WDK is ensuring stable and reliable measurements, the performance event trigger functions of PET should yield reproducible results. We ran all experiments described in Part V and evaluated in this chapter a second time, and calculated the relative run-to-run error. We also ran additional setups and on different SUTs. In total, 100 setups have been measured.

Figure 17.3 shows the relative run-to-run error. We can see that PET achieves a good reproducibility. The run-to-run error is, even in the worst case, below 5 %. Most runs are below 2 %. The median run-to-run error is, with 0.27 %, very low. Most outliers shown in Figure 17.3 are produced by the Pi workload emulations on SUT A.

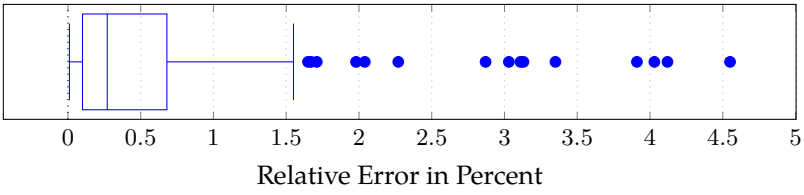


Figure 17.3: Relative error between two runs of PET.

We also take a look at the relative run-to-run error of each performance event trigger, visible in Figure 17.4. We calculated the relative error by building the quotient of run-to-run difference and the average of all events emitted for that type. Outliers that have a relative error above 25 % are not shown.

Visible in Figure 17.4 is that all performance event trigger functions perform reasonable well and stay below an error of 25 %. No event trigger function has a median above 2.5 %. However, the context switches and level 3 cache hit event triggers show a worse behavior in the upper quartiles. The other performance event triggers show reasonable variation with some outliers.

The analysis shows that the reproducibility of PET and its performance event triggers is reasonable. Especially the emulated power consumption shows promising results. The context switches and level 3 cache hit event trigger could be improved in terms of reproducibility.

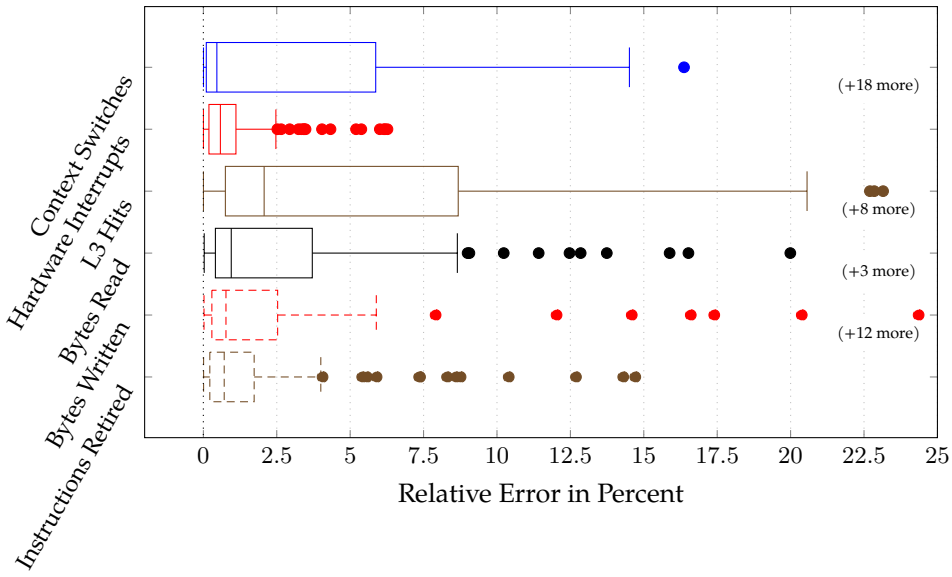


Figure 17.4: Relative error of performance event triggers.

17.4 Concluding Remarks

We have shown in Chapter 15 that classifying applications according to their resource profile works but that certain classes might be under-represented. Still, our approach can improve the accuracy of power consumption estimations by selecting a more suitable model, thereby answering our research goal *RQ B.1*. We answered our research questions *RQ B.2*, and *RQ B.3* in Chapter 16. We evaluated our new high-bandwidth memory model and our novel function-level power model, able to identify where the power is consumed in an application. Additionally, we answered research question *RQ B.4* in Chapter 16 by showing that our classification approach works on unknown applications and our function-level power model can identify the same most power-consuming functions regardless of the underlying hardware. Finally, we answer the research question *RQ B.5* in Chapter 17 showing that our novel configuration approach for the power emulation achieves better results without requiring expert knowledge.

Part VII

Conclusion and Future Work

Chapter 18

Conclusion

This chapter concludes the thesis with a summary of our contributions in the following Section 18.1. We also discuss the benefits of this thesis in Section 18.2.

18.1 Summary

This thesis addresses two primary goals. The *Goal A* (raising awareness among developers that software influences power consumption and energy efficiency) is addressed by presenting two case studies. They show the possible ranges when developers select compiler optimizations and algorithms with care. The second *Goal B* (support developers in modeling and finding energy-consuming application parts, as well as emulate the power consumption of applications) is addressed by providing an application classification to select suitable power models of fast-changing applications. Additionally, we address the second goal with two novel power models. The first model is for high-bandwidth memory applications, and the second is for identifying functions that consume the most power. We also present that the function-level power model and classification approach can be reused without retraining on unknown applications and servers. To further increase energy efficiency, we also presented improvements in configuration and triggering performance events to emulate the power consumption without dependencies more accurately. The contribution summary of this thesis are:

Contribution 1: Case Study Demonstrating the Impact of Compiler Optimizations

This thesis presents a case study on the impact of compiler optimizations on energy efficiency based on the SPEC CPU 2017 benchmark suite. The case study describes two scenarios, a real-world scenario in which compiler optimizations are set representing a typical development environment and a scenario in which an expert tunes compiler optimizations. The SPEC CPU 2017 benchmark

suite covers a wide variety of applications, and the two scenarios allow us to determine the possible range of achievable energy efficiency improvements. The presented results show that most applications benefit from carefully selected compiler optimizations with increased energy efficiency. We also present that C-like programming languages are more accessible to improve energy efficiency than functional languages like Fortran.

Contribution 2: Case Study Demonstrating the Impact of Sorting Algorithms

The second case study of this thesis presents the possible effects of algorithm selection based on the example of sorting algorithms. We present six common sorting algorithms with different time and space complexity that are implemented in two languages (C and Python) and two different variants. We ran all sorting algorithms on two servers, representative for cloud computing hardware, and with three different sizes of elements to sort. We show that even sorting algorithms with $n * \log(n)$ can perform very diverse in terms of energy efficiency. These results also show that selecting an appropriate algorithm can have a significant impact on energy efficiency. We also show that the type of implementation, like recursive or iterative, impacts the energy efficiency of an application significantly.

Contribution 3: Classifying Software Resource Usage Profiles

This contribution presents an approach to classify applications according to their resource usage profile based on performance events. We train three different supervised machine learning techniques from literature and compare their classification performance. We show that XGBoost and Random Forest perform better than LogitNet. Additionally, we can show that memory- and I/O-heavy applications are not well represented even with the large selection of training data from the employed benchmark suites. However, our approach is able to classify applications into resource usage profiles and improve accuracy by selecting the most suitable power model. Our approach also works on unknown applications and execution phases instead of averaging over the entire execution time. This approach allows developers to focus on writing applications rather than maintaining the accuracy of power models in a fast-changing DevOps environment.

Contribution 4: Fine-Granular Power Model for Energy-Efficient Development

Our fourth contribution presents a novel approach to attribute an application's power consumption to its functions based on performance events. It uses a stack trace to determine the functions with exclusive access to computational resources during a specified time frame. We are able to show that our approach can identify the most power-consuming functions of an application. In our approach, we can also differentiate between the actual application and the remaining software stack, in our case, the Tomcat application server. Additionally, the power model also identifies the most power-consuming functions despite changing the underlying hardware. Therefore, our function-level power model allows developers to focus improvements on energy efficiency on the most power-consuming functions achieving the highest impact.

Contribution 5: High-Bandwidth Memory Workload Power Model

This contribution presents a new power model for applications with a high-bandwidth memory demand. It uses the stream memory benchmark in a modified version to show that we can model high-bandwidth memory applications. The model allows a comparison of different access patterns, named *stride*, and different data types (`char`, `int`, `long`, and `double`). Hence, this model allows developers to compare different application implementations relatively or by absolute power consumption if the model is complemented with system-specific information.

Contribution 6: PET Improvements for Better Power Emulation

Our last contribution presents an improvement to PET, a framework to trigger performance events with synthetic code to emulate the power consumption of applications. We show that PET can be improved at varying positions. First, we present a novel automatic configuration approach for PET to accommodate side effects and also improve a performance event trigger function. We can show that the new automatic configuration reaches similar accuracy as if PET is configured by an expert with in-depth knowledge of the hardware, increasing PET's accuracy. Additionally, the new implementation for the performance event trigger achieves a higher throughput of triggering events, allowing PET to emulate an even wider variety of applications without decreasing accuracy. Therefore, this contribution allows developers and operators to emulate an application more accurately on many servers regardless of dependencies and without a cumbersome emulator configuration.

18.2 Benefits

We identified four main challenges that this thesis addresses in the Problem Statement Section 1.2 of the introduction. The goals and research questions are based on these challenges. This section presents the benefits that result in achieving our goals and answering the research questions. Specifically, developers benefit from addressing these challenges. This thesis has the following four main benefits:

- Both our case studies showed that significant improvements in energy efficiency are possible. The case studies also raise awareness about the possible range of improvements – both when using compiler optimizations and selecting the suitable algorithm for a problem. The results show developers that it is worthwhile to invest in energy efficiency. It is not only the hardware that is responsible for the power consumption when running an application. These results can also be used as an argument to include the energy efficiency of an application as a non-functional requirement for new developments.
- Our novel power models for high-bandwidth memory applications and attributing power consumption to functions allow developers to identify where power is consumed in their applications. The high-bandwidth memory model allows developers to compare algorithms or applications relatively. On the other hand, the fine-granular model allows developers to focus their effort on the essential functions. These contributions, therefore, help developers improve energy efficiency by giving them the information necessary for which parts are most lucrative to enhance or rewrite.
- This thesis also helps developers select the most suitable power model for their application to estimate the power consumption more accurately. By classifying applications according to their resource usage profile based on performance events, developers do not need to invest in searching a power model or modeling the application themselves but get a suggestion on which power model to use. This classification approach is also helpful in a DevOps environment where applications change rapidly. Continuously changing models are challenging to maintain, and just selecting a new, suitable model without retraining relieves the developers from constant maintenance of their models.
- It is beneficial to emulate the power consumption before deployment, allowing developers to select the most energy-efficient hardware for their

application. An application with external dependencies is difficult to test on a wide variety of deployment options available in a DevOps and cloud computing environment. The improved emulation that removes the need to resolve dependencies and accurately emulate the power consumption of an application aids developers and operators in deploying their application most energy efficient.

Chapter 19

Future Work

During this thesis, we identified several ideas through our presented work that can be pursued and improved further. We also identified challenges to address in the future that we present in this chapter.

19.1 Resource Efficiency Benchmark

Benchmarks can have a meaningful impact on the stakeholder for which the benchmark is designed. The SPECpower_ssj benchmark enabled hardware manufacturers to increase the energy efficiency of servers by 19 times on average. Similar to server hardware, an application benchmark could achieve similar results and raise awareness among developers but also other stakeholders.

We use resource profiles or resource utilization measures to classify or model power consumption through multiple contributions of our work. The power consumption is related to the resource usage of the application. Therefore, we identified the need for a resource efficiency benchmark to provide the foundation for more resource-efficient software. However, multiple open challenges have to be addressed in the future. We published this vision in the Proceedings of the 12th ACM/SPEC International Conference on Performance Engineering (ICPE) 2021 [Sch+21c].

Challenge 1 *How to describe the resource demand of software?*
To the best of our knowledge, there is currently no commonly accepted standard to express the resource demand of software. While resource profiles are an approach to describing a software system's resource demand, they are dependent on the corresponding workload [BWK14]. For server-side software capacity planning, trial and error is a commonly accepted norm to deal with software resource requirements [WFP07; Smi07]. However, capacity planning results only describe the implications of a specific workload on that software, not the actual software resource demands. One of the main consequences of this

approach is that even today, there are a lot of underutilized servers because the resource requirements have been estimated using wrong assumptions [Jan+11]. Therefore, it would be good to make the resource demand more transparent. This transparency would allow the software users to consider the actual resource demands during their purchasing decisions. It would create pressure on the vendors to reduce these numbers to convince the buyers to choose their product [BWK14]. Eventually, this would drive down the resource and energy demand of all software systems.

Challenge 2 *How to specify and standardize workloads?*
In previous work [BK17], the authors presented an approach based on [BBS07] to describe the resource demand of individual transactions of a software system. However, the workload needs to be standardized to make it comparable. The workload of software systems is very dependent on their type and can only be standardized to a certain degree. According to [Vög+18], a workload is defined by behavior models combined into a behavior mix, which then has a certain workload intensity. Behavior models describe a typical usage flow of a user type. The behavior mix specifies which percentage of the overall user population follows which behavior model. The workload intensity defines how many users with a given behavior mix are active in the system at a time. These workload parts will differ per application type (e.g., ERP, DBMS, CRM, Industry 4.0 software), and software vendors need to agree on a standard workload model for each application type to make resource demands comparable.

Challenge 3 *What are possible incentives to increase awareness and acceptance?*
As the experience with other devices such as cars or fridges shows, even if we had the technical means and standardized workloads, the stakeholders' awareness (e.g., software buyers, operators, and providers) needs to be increased to make the overall concept work. Therefore, this is a socio-technical problem that cannot be solved solely by providing a technical solution. Thus it might be possible to adapt working approaches from similar incentive-based mechanisms. Examples for this are the adoption of labels like the Energy Star [SMK17].

19.2 Automated Class Definitions

In our *Contribution 3*, we define the resource usage profiles of our applications by hand. This procedure can be error prone and cumbersome. Additionally, the imbalance of memory- and I/O-heavy applications towards one or two classes should be addressed in the future.

We propose an automated approach that can define theses resource usage classes for each subsystem, CPU, memory, and I/O. It should also be able to detect if an imbalance is present for a subsystem. If an imbalance is detected, the approach should automatically use a suitable technique to try to remedy the problem if possible. This would allow developers to collect measurement data suitable for their use-case and train the classifier accordingly and, in turn, could improve accuracy of the power model selection for better power consumption estimations.

19.3 Stack Sampling for Function-level Power Models

Given the limitations of our *Contribution 4*, the first step is to validate our approach on broader hardware set with different applications. More importantly, though, is to introduce a sample-based stack monitoring compared to the event-based Kieker solution, minimizing the interference and overhead even further and thus allowing us to monitor large-scale cloud systems. Additionally, our model should be compared to existing and well-studied power models. For the DevOps community, providing valuable insight into the power consumption of application functions can be beneficial to conserve energy and support autonomous placement and deployment decisions.

To further reduce the monitoring overhead, we envision a fast stack simulation that returns the current stack layout, together with a detailed overhead evaluation. Compared to the current event-based stack monitoring, sample-based or simulation could achieve a lesser overhead. However, such a solution needs to monitor additional input parameters instead.

As the correction factor for our approach did not work as expected, different alternatives include a Kalman filter and time-series prediction. A prediction, including a confidence value, could improve our work and determine falsely allocated performance events. Static code analysis can also help reduce errors. For example, by analyzing functions, finding disk access through known functions in the language's API, performance events could be shifted towards the causing function away from functions without disk access.

19.4 Background Noise for Performance and Power Models

It is common in cloud computing to have multiple users share a pool of computational resources through virtualization techniques. These users could

influence each other as the performance isolation of virtual machines is not perfect. Therefore, performance and power models should be trained while other applications are present and not when it has exclusive access to computational resources. We, therefore, propose to develop PET further and evaluate it so that PET can act as a background noise generator for modeling. As PET is versatile and can emulate a wide range of applications, it would allow developers to produce more accurate models of their application.

To allow PET to generate background noise, and for the accuracy of its emulation in general, we also propose to increase the accuracy of the performance event trigger functions further. Specifically, the performance events context switches and level 3 cache hits. Better performance event triggers would also allow PET to emulate single functions of an application. This more fine-granular emulation could also prove helpful in a FaaS context where only functions are deployed instead of microservices or monolithic applications.

List of Figures

List of Tables

Acronyms

ALU Arithmetic Logic Unit. 159

CI/CD Continuous Integration / Continuous Deployment. 17

CRM Customer Relationship Management. 42, 176

DBMS Database Management System. 42, 176

DDA Direct Drive Access. 51

DVFS Dynamic Voltage and Frequency Scaling. 2, 49, 114

ERP Enterprise Resource Planning. 3, 42, 61, 176

FaaS Function as a Service. 15, 16, 178

FFNN Feed-Forward Neural Network. 33, 34, 40, 120, 154, 155, 179, 180

HMC Hardware Management Console. 87

HPC High Performance Computing. 39, 40, 45, 46

IaaS Infrastructure as a Service. 16

IPC Instruction Per Cycle. 89, 130, 132, 159, 160

JSP Java Server Page. 29

LFU Least Frequently Used. 50, 51

LLC Last Level Cache. 107, 144

LOC Lines of Code. 26

LPAR Logical Partition. 86, 87, 95, 99, 138, 181

MAE Mean Absolute Error. 155

MAPE Mean Average Percentage Error. 155

MSE Mean Squared Error. 24

MSR Model Specific Register. 17, 40

NIST National Institute of Standards and Technology. 15, 16

NPB NAS Parallel Benchmark. 45

OLTP Online Transaction Processing. 32

OSG Open Systems Group. 25

PaaS Platform as a Service. 16

PEBS Performance Event Based Sampling. 18

PET Performance Event Trigger Framework. 5, 10, 11, 15, 19, 21–24, 125–127, 129, 131, 132, 135, 136, 161, 162, 164, 171, 178–180, 182, 219

PMU Performance Monitoring Unit. 17, 18, 89

PSU Power Supply Unit. 20

RAPL Running Average Power Limit. 19–21, 152

REST Representational State Transfer. 29, 50

RR Random Replacement. 50, 51

SaaS Software as a Service. 16

SERT Server Efficiency Rating Tool. 5, 19–21, 31, 32, 106, 107, 144

SoC System on Chip. 40

SPEC Standard Performance Evaluation Corporation. 8, 19, 25, 26, 45, 106, 144

SPEC RG Standard Performance Evaluation Corporation, Research Group. 15, 29

SQL Structured Query Language. 30

SSE Streaming SIMD Extension. 130–132, 159, 160

SSJ Server Side Java. 31, 32

SUT System under Test. 8–10, 20, 21, 26, 27, 31, 32, 54–56, 58, 59, 71, 72, 75–77, 87–92, 94, 95, 97, 109, 110, 115, 116, 119, 121, 126–130, 135, 148, 150, 153, 154, 156–161, 163, 164, 180–182

SVM Support Vector Machine. 45

TLB Translation Lookaside Buffer. 90

XML Extensible Markup Language. 32

Bibliography

[ASB17] R. Abbas, Z. Sultan, and S. N. Bhatti. "Comparative analysis of automated load testing tools: Apache JMeter, Microsoft Visual Studio (TFS), LoadRunner, Siege". In: *2017 International Conference on Communication Technologies (ComTech)*. 2017, pp. 39–44 (see page 115).

[Adv21] Advanced Micro Devices. *AMD64 Architecture Programmer's Manual Volume 2: System Programming*. Advanced Micro Devices. Santa Clara, California, U.S., 2021 (see pages 17, 18).

[Agg18] C. C. Aggarwal. "An Introduction to Neural Networks". In: *Neural Networks and Deep Learning: A Textbook*. Cham: Springer International Publishing, 2018. Chap. 1, pp. 1–52 (see page 34).

[AHS15] K. Aggarwal, A. Hindle, and E. Stroulia. "GreenAdvisor: A tool for analyzing the impact of software evolution on energy consumption". In: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2015, pp. 311–320 (see page 42).

[Agg+14] K. Aggarwal, C. Zhang, J. C. Campbell, A. Hindle, and E. Stroulia. "The power of system call traces: predicting the software energy consumption impact of changes." In: *CASCON*. Vol. 14. 2014, pp. 219–233 (see page 42).

[AC17] S. Aminikhanghahi and D. J. Cook. "A survey of methods for time series change point detection". In: *Knowledge and information systems* 51.2 (2017), pp. 339–367 (see page 140).

[AE15] A. Andrae and T. Edler. "On Global Electricity Usage of Communication Technology: Trends to 2030". In: *Challenges* 6.1 (2015), pp. 117–157 (see page 1).

[And19] A. S. Andrae. "Projecting the chiaroscuro of the electricity use of communication and computing from 2018 to 2030". In: *Preprint* (2019), pp. 1–23 (see pages 1, 49).

[AA10] R. B. ad Anton Beloglazov and J. H. Abawajy. "Energy-Efficient Management of Data Center Resources for Cloud Computing: A Vision, Architectural Elements, and Open Challenges". In: *CoRR* abs/1006.0308 (2010). eprint: 1006.0308 (see page 109).

[Arm21] Arm Limited. *Arm Architecture Reference Manual. Armv8, for A-profile architecture*. Arm Limited. Cambridge, England, 2021 (see page 17).

[Arn13] J. Arnold. "Chauffeur: A framework for measuring Energy Efficiency of Servers". In: *Master's project, University of Minnesota* (2013) (see pages 22, 31).

[Ban+14] A. Banerjee, L. K. Chong, S. Chattopadhyay, and A. Roychoudhury. "Detecting Energy Bugs and Hotspots in Mobile Apps". In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2014. Hong Kong, China: Association for Computing Machinery, 2014, pp. 588–598 (see page 42).

[BH07] L. A. Barroso and U. Hölzle. "The case for energy-proportional computing". In: *Computer* 40.12 (2007), pp. 33–37 (see page 19).

[BD12] R. Basmadjian and H. De Meer. "Evaluating and modeling power consumption of multi-core processors". In: *Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy), 2012 Third International Conference on*. 2012, pp. 1–10 (see page 40).

[Bau+20] A. Bauer, M. Züfle, J. Grohmann, N. Schmitt, N. Herbst, and S. Kounev. "An Automated Forecasting Framework based on Method Recommendation for Seasonal Time Series". In: *Proceedings of the 11th ACM/SPEC International Conference on Performance Engineering (ICPE 2020)*. Edmonton, Canada: ACM, 2020.

[BJ07] W. L. Bircher and L. K. John. "Complete system power estimation: A trickle-down approach based on performance events". In: *2007 ieee international symposium on performance analysis of systems & software*. IEEE. 2007, pp. 158–168 (see pages 39, 89, 90).

[BJ12] W. L. Bircher and L. K. John. "Complete System Power Estimation Using Processor Performance Events". In: *IEEE Transactions on Computers* 61.4 (2012), pp. 563–577 (see pages 4, 38, 39, 110).

[BC10] A. Bohra and V. Chaudhary. "VMeter: Power modelling for virtualized clouds". In: *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*. 2010, pp. 1–8 (see page 40).

[BBS07] R. Brandl, M. Bichler, and M. Ströbel. "Cost accounting for shared IT infrastructures". In: *Wirtschaftsinformatik* 49.2 (2007), pp. 83–94 (see page 176).

[Bre01] L. Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32 (see page 33).

[Bru+15] A. Brunnert, A. van Hoorn, et al. *Performance-oriented DevOps: A Research Agenda*. 2015. arXiv: `1508.04752 [cs.SE]` (see pages 4, 17, 85).

[BK17] A. Brunnert and H. Krcmar. "Continuous performance evaluation and capacity planning using resource profiles for enterprise applications". In: *Journal of Systems and Software* 123 (2017), pp. 239–262 (see page 176).

[BWK14] A. Brunnert, K. Wischer, and H. Krcmar. "Using Architecture-Level Performance Models as Resource Profiles for Enterprise Applications". In: *Proceedings of the 10th International ACM Sigsoft Conference on Quality of Software Architectures*. QoSA '14. Marcq-en-Bareul, France: Association for Computing Machinery, 2014, pp. 53–62 (see pages 175, 176).

[BLK18] J. Bucek, K.-D. Lange, and J. v. Kistowski. "SPEC CPU2017: Next-Generation Compute Benchmark". In: *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. ICPE '18. Berlin, Germany: ACM, 2018, pp. 41–42 (see pages 26, 27, 53).

[Bun+09] C. Bunse, H. Höpfner, S. Roychoudhury, and E. Mansour. "Choosing the" Best" Sorting Algorithm for Optimal Energy Consumption." In: *ICSOFT (2)*. 2009, pp. 199–206 (see pages 43, 71).

[CP15a] C. Calero and M. Piattini. *Green in software engineering*. Vol. 3. Springer, 2015 (see pages 50, 109).

[CP15b] C. Calero and M. Piattini. "Introduction to Green in Software Engineering". In: *Green in Software Engineering*. Ed. by C. Calero and M. Piattini. Cham: Springer International Publishing, 2015. Chap. 1, pp. 3–27 (see page 3).

[CFS12] E. Capra, C. Francalanci, and S. A. Slaughter. "Is software "green"? Application development environments and energy efficiency in open source applications". In: *Information and Software Technology* 54.1 (2012), pp. 60–71 (see pages 3, 42, 50, 51, 61).

[CPK13] T. B. Chandra, V. Patle, and S. Kumar. "New horizon of energy efficiency in sorting algorithms: green computing". In: *Proceedings of National Conference on Recent Trends in Green Computing. School of Studies in Computer in Computer Science & IT, Pt. Ravishankar Shukla University, Raipur, India*. 2013, pp. 24–26 (see pages 43, 71).

[CG16] T. Chen and C. Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 785–794 (see pages 33, 100).

[Che+20] W.-Y. Chen, K.-J. Ye, C.-Z. Lu, D.-D. Zhou, and C.-Z. Xu. "Interference analysis of co-located container workloads: a perspective from hardware performance counters". In: *Journal of Computer Science and Technology* 35 (2020), pp. 412–417 (see page 4).

[Che+10] X. Chen, C. Xu, R. P. Dick, and Z. M. Mao. "Performance and Power Modeling in a Multi-programmed Multi-core Environment". In: *Proceedings of the 47th Design Automation Conference*. DAC '10. Anaheim, California: ACM, 2010, pp. 813–818 (see pages 38, 110).

[CPN18] J. Choi, G. Park, and D. Nam. "Efficient Classification of Application Characteristics by Using Hardware Performance Counters with Data Mining". In: *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. 2018, pp. 24–29 (see page 45).

[Dob+13] I. Dobos et al. *IBM zEnterprise EC12 Technical Guide*. 2013 (see page 87).

[DZ11] M. Dong and L. Zhong. "Self-constructive High-rate System Energy Modeling for Battery-powered Mobile Systems". In: *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*. MobiSys '11. Bethesda, Maryland, USA: ACM, 2011, pp. 335–348 (see pages 39, 41).

[Eis+19] S. Eismann, J. Kistowski, J. Grohmann, A. Bauer, N. Schmitt, and S. Kounev. "TeaStore - A Micro-Service Reference Application (Demo)". In: *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*. 2019, pp. 263–264 (see page 29).

[Eis+18] S. Eismann, J. v. Kistowski, J. Grohmann, A. Bauer, N. Schmitt, N. Herbst, and S. Kounev. "TeaStore: A Micro-Service Reference Application for Cloud Researchers (Poster)". In: *Proceedings of 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. 2018, pp. 11–12 (see page 29).

[FWB07] X. Fan, W.-D. Weber, and L. A. Barroso. "Power Provisioning for a Warehouse-sized Computer". In: *The 34th ACM International Symposium on Computer Architecture*. 2007 (see pages 40, 109).

[Fle20] Flexera. *State of the Cloud Report*. 2020 (see page 1).

[Flo+15] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya. "Mobile code offloading: from concept to practice and beyond". In: *IEEE Communications Magazine* 53.3 (2015), pp. 80–88 (see page 4).

[Gad+18] D. Gadioli, R. Nobre, P. Pinto, E. Vitali, A. H. Ashouri, G. Palermo, J. Cardoso, and C. Silvano. "SOCRATES - A seamless online compiler and system runtime autotuning framework for energy-aware applications". In: *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2018, pp. 1143–1146 (see page 44).

[Gar20] Gartner. *Gartner Forecasts Worldwide Public Cloud End-User Spending to Grow 18 percent in 2021*. 2020 (see page 1).

[GCB05] S. V. Gheorghita, H. Corporaal, and T. Basten. "Iterative compilation for energy reduction". In: *Journal of Embedded Computing* 1.4 (2005), pp. 509–520 (see page 44).

[Gla12] J. Glanz. "Power, Pollution and the Internet". In: *New York Times* (2012) (see page 1).

[GT15] C. Gormley and Z. Tong. *Elasticsearch: the definitive guide: a distributed real-time search and analytics engine*. " O'Reilly Media, Inc.", 2015 (see pages 85, 88).

[Gur+02] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, and L. K. John. "Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach". In: *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*. HPCA '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 141– (see page 40).

[HWT10] G. Hager, G. Wellein, and J. Treibig. "LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments". In: *2012 41st International Conference on Parallel Processing Workshops*. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 207–216 (see page 17).

[Häh+12] M. Hähnel, B. Döbel, M. Völp, and H. Härtig. "Measuring Energy Consumption for Short Code Paths Using RAPL". In: *SIGMETRICS Perform. Eval. Rev.* 40.3 (2012), pp. 13–17 (see page 19).

[Hao+13] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan. "Estimating mobile application energy consumption using program analysis". In: *2013 35th International Conference on Software Engineering (ICSE)*. 2013, pp. 92–101 (see pages 4, 41).

[HH20] W. Hasselbring and A. van Hoorn. "Kieker: A monitoring framework for software engineering research". In: *Software Impacts* 5 (2020), pp. 1–5 (see page 111).

[HM19] R. Hebbar SR and A. Milenković. "SPEC CPU2017: Performance, Event, and Energy Characterization on the Core i7-8700K". In: *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*. ACM. 2019, pp. 111–118 (see page 50).

[HJJ03] D. Henderson, S. H. Jacobson, and A. W. Johnson. "The theory and practice of simulated annealing". In: *Handbook of metaheuristics*. Springer, 2003, pp. 287–319 (see page 24).

[Hen19a] J. Henning. *SPEC CPU 2017® benchmarks*. 2019 (see pages 28, 57).

[Hen19b] J. Henning. *SPEC CPU 2017® run and reporting rules*. 2019 (see pages 26, 27, 58, 144).

[Her18] N. Herbst. "Methods and Benchmarks for Auto-Scaling Mechanisms in Elastic Cloud Environments". PhD thesis. University of Würzburg, Germany, 2018 (see pages 4, 16).

[Hil09] J. M. Hilbe. *Logistic regression models*. Chapman and hall/CRC, 2009 (see pages 34, 35).

[Hoo+09] A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey, and D. Kieselhorst. *Continuous Monitoring of Software Services: Design and Application of the Kieker Framework*. Forschungsbericht. Kiel University, 2009 (see pages 29, 111).

[HWH12] A. van Hoorn, J. Waller, and W. Hasselbring. "Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis". In: *Proceedings of the 3rd joint ACM/SPEC International Conference on Performance Engineering (ICPE 2012)*. ARRAY(0x9b39db0), 2012, pp. 247–248 (see pages 29, 111).

[HE08] K. Hoste and L. Eeckhout. "Cole: compiler optimization level exploration". In: *Proceedings of the 6th annual IEEE/ACM international symposium on Code generation and optimization*. ACM. 2008, pp. 165–174 (see page 44).

[HK03] C.-H. Hsu and U. Kremer. "The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction". In: *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*. PLDI '03. San Diego, California, USA: ACM, 2003, pp. 38–48 (see page 44).

[Hup09] K. Huppler. "The Art of Building a Good Benchmark". In: *Performance Evaluation and Benchmarking*. Ed. by R. Nambiar and M. Poess. Vol. 5895. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 18–30 (see page 25).

[IBM18] IBM. *POWER9 Performance Monitor Unit User's Guide*. IBM. Somers, New York, U.S., 2018 (see page 17).

[Int] Intel. *Intel Performance Counter Monitor* (see page 144).

[Int17] Intel Corporation. *Intel® 64 and IA-32 Architectures Performance Monitoring Events*. Intel Corporation. Santa Clara, California, U.S., 2017 (see pages 17, 18).

[Int16] Intel Corporation. *Intel® 64 and IA-32 Architectures Software Developer's Manual*. Intel Corporation. Santa Clara, California, U.S., 2016 (see pages 18, 19, 21, 110).

[IM03] C. Isci and M. Martonosi. "Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data". In: *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO 36. Washington, DC, USA: IEEE Computer Society, 2003 (see pages 4, 38, 110).

[Jab+16] R. Jabbari, N. bin Ali, K. Petersen, and B. Tanveer. "What is DevOps? A Systematic Mapping Study on Definitions and Practices". In: *Proceedings of the Scientific Workshop Proceedings of XP2016*. XP '16 Workshops. Edinburgh, Scotland, UK: Association for Computing Machinery, 2016 (see page 17).

[Jam+13] G. James, D. Witten, T. Hastie, and R. Tibshirani. "Tree-Based Methods". In: *An Introduction to Statistical Learning: with Applications in R*. New York, NY: Springer New York, 2013. Chap. 8, pp. 303–335 (see page 33).

[Jam13] James, Gareth and Witten, Daniela and Hastie, Trevor and Tibshirani, Robert. "Statistical Learning". In: *An Introduction to Statistical Learning: with Applications in R*. New York, NY: Springer New York, 2013. Chap. 2, pp. 15–57 (see page 33).

[Jan+11] J. Jang, M. Jeon, H. Kim, H. Jo, J. Kim, and S. Maeng. "Energy Reduction in Consolidated Servers through Memory-Aware Virtual Machine Scheduling". In: *IEEE Transactions on Computers* 60.4 (2011), pp. 552–564 (see page 176).

[Jia+13] Z. Jia, L. Wang, J. Zhan, L. Zhang, and C. Luo. "Characterizing data analysis workloads in data centers". In: *2013 IEEE International Symposium on Workload Characterization (IISWC)*. 2013, pp. 66–76 (see page 44).

[JWC12] Y. Jin, Y. Wen, and Q. Chen. "Energy Efficiency and Server Virtualization in Data Centers: An Empirical Investigation". In: *2012 IEEE Conference on Computer Communications Workshops*. 2012, pp. 133–138 (see pages 4, 16, 49).

[Joh+11] T. Johann, M. Dick, E. Kern, and S. Naumann. "Sustainable development, sustainable software, and sustainable software engineering: An integrated approach". In: *2011 International Symposium on Humanities, Science and Engineering Research*. 2011, pp. 34–39 (see page 3).

[Kan+14] A. Kandalintsev, R. Lo Cigno, D. Kliazovich, and P. Bouvry. "Profiling cloud applications with hardware performance counters". In: *The International Conference on Information Networking 2014 (ICOIN2014)*. 2014, pp. 52–57 (see pages 4, 45).

[Kan+00] M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and W. Ye. "Influence of Compiler Optimizations on System Power". In: *Proceedings of the 37th Annual Design Automation Conference*. DAC '00. Los Angeles, California, USA: ACM, 2000, pp. 304–307 (see page 43).

[Kin19] C. I. King. *Stress-ng*. 2019 (see pages 85, 88).

[Kis19] J. von Kistowski. "Measuring, Rating, and Predicting the Energy Efficiency of Servers". PhD thesis. University of Würzburg, Germany, 2019 (see pages 5, 10, 21, 24, 125, 135, 160).

[Kis+15a] J. von Kistowski, J. A. Arnold, K. Huppler, K.-D. Lange, J. L. Henning, and P. Cao. "How to Build a Benchmark". In: *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE 2015)*. ICPE '15. Austin, TX, USA: ACM, 2015 (see page 25).

[Kis+15b] J. von Kistowski, H. Block, J. Beckett, K.-D. Lange, J. A. Arnold, and S. Kounev. "Analysis of the Influences on Server Power Consumption and Energy Efficiency for CPU-Intensive Workloads". In: *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE 2015)*. ICPE '15. Austin, TX, USA: ACM, 2015 (see pages 5, 105, 135).

[Kis+19a] J. von Kistowski, S. Eismann, N. Schmitt, A. Bauer, J. Grohmann, L. Bui, and S. Kounev. *TeaStore*. Standard Performance Evaluation Corporation (SPEC) Research Group. Accepted tool. 2019 (see page 29).

[Kis+18] J. von Kistowski, S. Eismann, N. Schmitt, A. Bauer, J. Grohmann, and S. Kounev. "TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research". In: *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE. 2018, pp. 223–236 (see pages 9, 29, 30, 111).

[Kis+19b] J. von Kistowski, J. Grohmann, N. Schmitt, and S. Kounev. "Predicting Server Power Consumption from Standard Rating Results". In: *Proceedings of the 19th ACM/SPEC International Conference on Performance Engineering*. ICPE '19. Full Paper Acceptance Rate: 18.6% (13/70). New York, NY, USA: Association for Computing Machinery (ACM), 2019, pp. 301–312.

[KHK14] J. G. von Kistowski, N. R. Herbst, and S. Kounev. "Modeling Variations in Load Intensity over Time". In: *Proceedings of the 3rd International Workshop on Large-Scale Testing (LT 2014), co-located with the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*. Dublin, Ireland: ACM, 2014 (see page 5).

[KKP02] D. G. Kleinbaum, M. Klein, and E. R. Pryor. "Logistic regression: a self-learning text". In: (2002) (see pages 34, 35).

[Kou+17] S. Kounev, J. O. Kephart, A. Milenkoski, and X. Zhu, eds. *Self-Aware Computing Systems*. Berlin Heidelberg, Germany: Springer Verlag, 2017 (see page 15).

[KLK20] S. Kounev, K.-D. Lange, and J. von Kistowski. *Systems Benchmarking. For Scientists and Engineers*. 1st ed. Springer International Publishing, 2020 (see pages 19, 21, 25, 53, 72, 144).

[Lan09a] K.-D. Lange. "Identifying Shades of Green: The SPECpower Benchmarks". In: *Computer* 42.3 (2009), pp. 95–97 (see page 106).

[LT11] K.-D. Lange and M. G. Tricker. "The Design and Development of the Server Efficiency Rating Tool (SERT)". In: *Proceedings of the 2nd ACM/SPEC International Conference on Performance Engineering*. ICPE '11. Karlsruhe, Germany: ACM, 2011, pp. 145–150 (see pages 106, 107).

[Lan09b] K.-D. Lange. "The Next Frontier for Power/Performance Benchmarking: Energy Efficiency of Storage Subsystems". In: *Proceedings of the 2009 SPEC Benchmark Workshop on Computer Performance Evaluation and Benchmarking*. Austin, TX: Springer-Verlag, 2009, pp. 97–101 (see page 49).

[Li+13] D. Li, S. Hao, W. G. Halfond, and R. Govindan. "Calculating source line level energy information for android applications". In: *Proceedings of the 2013 International Symposium on Software Testing and Analysis*. ACM. 2013, pp. 78–89 (see pages 41, 42).

[Li+09] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures". In: *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO 42. New York, New York: ACM, 2009, pp. 469–480 (see page 40).

[LPF10] M. Y. Lim, A. Porterfield, and R. Fowler. "SoftPower: Fine-grain Power Estimations Using Performance Counters". In: *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. HPDC '10. Chicago, Illinois: ACM, 2010, pp. 308–311 (see pages 4, 38, 110).

[Liv+12] C. Lively, X. Wu, V. Taylor, S. Moore, H.-C. Chang, C.-Y. Su, and K. Cameron. "Power-aware predictive models of hybrid (MPI/OpenMP) scientific applications on multicore systems". In: *Computer Science-Research and Development* 27.4 (2012), pp. 245–253 (see pages 38, 39).

[MD04] M. Mamidipaka and N. Dutt. "eCACTI: An enhanced power estimation model for on-chip caches". In: *Center for Embedded Computer Systems, Technical Report TR* (2004), pp. 04–28 (see page 40).

[Mam+19] N. Mammeri, M. Neu, S. Lal, and B. Juurlink. "Performance Counters based Power Modeling of Mobile GPUs using Deep Learning". In: *2019 International Conference on High Performance Computing Simulation (HPCS)*. 2019, pp. 193–200 (see page 40).

[Mar+16] L. G. Martins, R. Nobre, J. M. Cardoso, A. C. Delbem, and E. Marques. "Clustering-based selection for the exploration of compiler optimization sequences". In: *ACM Transactions on Architecture and Code Optimization (TACO)* 13.1 (2016), p. 8 (see page 44).

[Mas+20] E. Masanet, A. Shehabi, N. Lei, S. Smith, and J. Koomey. "Recalibrating global data center energy-use estimates". In: *Science* 367.6481 (2020), pp. 984–986 (see pages 1, 49).

[McC95] J. D. McCalpin. "STREAM benchmark". In: *Link: www. cs. virginia. edu/stream/ref. html# what* 22 (1995) (see pages 10, 143).

[MG11] P. Mell and T. Grance. *The NIST Definition of Cloud Computing*. Tech. rep. NIST, 2011 (see page 15).

[Nag19a] M. kumar raj Nagarajan. *Clang - the C, C++ Compiler*. 2019 (see page 54).

[Nag19b] M. kumar raj Nagarajan. *Flang - the Fortran Compiler*. 2019 (see page 54).

[Net12] Netflix Inc. *Netflix Ribbon*. 2012 (see page 29).

[NRC18] R. Nobre, L. Reis, and J. M. Cardoso. "Compiler Phase Ordering as an Orthogonal Approach for Reducing Energy Consumption". In: *CoRR* abs/1807.00638 (2018). arXiv: 1807.00638 (see page 43).

[PHB13] J. Pallister, S. J. Hollis, and J. Bennett. "Identifying compiler options to minimize energy consumption for embedded platforms". In: *The Computer Journal* 58.1 (2013), pp. 95–109 (see page 43).

[PJ14] R. Panda and L. K. John. "Data analytics workloads: Characterization and similarity analysis". In: *2014 IEEE 33rd International Performance Computing and Communications Conference (IPCCC)*. 2014, pp. 1–9 (see page 45).

[Pan+16] C. Pang, A. Hindle, B. Adams, and A. E. Hassan. "What Do Programmers Know about Software Energy Consumption?" In: *IEEE Software* 33.3 (2016), pp. 83–89 (see page 3).

[PHZ12] A. Pathak, Y. C. Hu, and M. Zhang. "Where is the Energy Spent Inside My App?: Fine Grained Energy Accounting on Smartphones with Eprof". In: *Proceedings of the 7th ACM European Conference on Computer Systems*. EuroSys '12. Bern, Switzerland: ACM, 2012, pp. 29–42 (see page 42).

[PC17] G. Pinto and F. Castor. "Energy Efficiency: A New Concern for Application Software Developers". In: *Commun. ACM* 60.12 (2017), pp. 68–75 (see pages 4, 41, 49).

[RAT15] M. Rashid, L. Ardito, and M. Torchiano. "Energy Consumption Analysis of Algorithms Implementations". In: *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 2015, pp. 1–4 (see pages 43, 71).

[Red18] RedHat. *The path to cloud-native applications*. 2018 (see page 17).

[RJ00] G. Reinman and N. P. Jouppi. "CACTI 2.0: An integrated cache timing and power model". In: *Western Research Lab Research Report* 7 (2000) (see page 40).

[RRK08] S. Rivoire, P. Ranganathan, and C. Kozyrakis. "A Comparison of High-level Full-system Power Models". In: *Proceedings of the 2008 Conference on Power Aware Computing and Systems*. HotPower'08. San Diego, California: USENIX Association, 2008, pp. 3–3 (see page 40).

[Rod+13] R. Rodrigues, A. Annamalai, I. Koren, and S. Kundu. "A Study on the Use of Performance Counters to Estimate Power in Microprocessors". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 60.12 (2013), pp. 882–886 (see pages 38, 39, 110).

[Rot+12] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan. "Power-management architecture of the intel microarchitecture code-named sandy bridge". In: *Ieee micro* 32.2 (2012), pp. 20–27 (see page 19).

[SMK17] A. Sanderford, A. McCoy, and M. Keefe. "Adoption of Energy Star certifications: theory and evidence compared". English (US). In: *Building Research and Information* (2017), pp. 1–13 (see page 176).

[Sch19] N. Schmitt. "Improving the Energy Efficiency of IoT-Systems and its Software". In: *Organic Computing: Doctoral Dissertation Colloquium 2018*. Ed. by S. Tomforde and B. Sick. kassel university press GmbH, 2019.

[Sch+20a] N. Schmitt, J. Bucek, J. Beckett, A. Cragin, K.-D. Lange, and S. Kounev. "Performance, Power, and Energy-Efficiency Impact Analysis of Compiler Optimizations on the SPEC CPU 2017 Benchmark Suite". In: *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. Los Alamitos, CA, USA: IEEE Computer Society, 2020, pp. 292–301 (see pages 8, 53).

[Sch+20b] N. Schmitt, J. Bucek, J. Beckett, A. Cragin, K.-D. Lange, and S. Kounev. *SPEC CPU 2017 Benchmark Suite Results for the HPE ProLiant DL385 Gen10 server*. Zenodo, 2020 (see pages 8, 53, 58).

[Sch+20c] N. Schmitt, J. Bucek, K.-D. Lange, and S. Kounev. "Energy Efficiency Analysis of Compiler Optimizations on the SPEC CPU 2017 Benchmark Suite". In: *Proceedings of the 11th ACM/SPEC International Conference on Performance Engineering (ICPE 2020)*. New York, NY, USA: ACM, 2020 (see pages 8, 53).

[Sch+19a] N. Schmitt, L. Iffländer, A. Bauer, and S. Kounev. "Online Power Consumption Estimation for Functions in Cloud Applications". In: *2019 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE. 2019, pp. 63–72 (see pages 10, 109).

[Sch+19b] N. Schmitt, L. Iffländer, A. Bauer, and S. Kounev. "Online Power Consumption Estimation for Functions in Cloud Applications (Poster)". In: *2019 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE. 2019.

[Sch+21a] N. Schmitt, S. Kamthania, N. Rawtani, L. Mendoza, K.-D. Lange, and S. Kounev. "Energy-Efficiency Comparison of Common Sorting Algorithms". In: *Proceedings of the 29th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and*

Telecommunication Systems. MASCOTS '21. New York, NY, USA, 2021 (see pages 9, 71).

[SKK17a] N. Schmitt, J. von Kistowski, and S. Kounev. "Emulating the Power Consumption Behavior of Server Workloads using CPU Performance Counters". In: *Proceedings of the 25th IEEE International Symposium on the Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*. MASCOTS '17. 2017 (see pages 5, 10, 21, 125, 131, 135, 160).

[SKK17b] N. Schmitt, J. von Kistowski, and S. Kounev. "Predicting Power Consumption of High-Memory-Bandwidth Workloads". In: *Proceedings of the 8th ACM/SPEC International Conference on Performance Engineering*. ICPE '17. New York, NY, USA: ACM, 2017, pp. 353–356 (see pages 10, 106).

[SKK17c] N. Schmitt, J. von Kistowski, and S. Kounev. "Towards a Scalability and Energy Efficiency Benchmark for VNF". In: *Proceedings of the 9th TPC Technology Conference on Performance Engineering and Benchmarking*. TPCTC '17. 2017.

[Sch+21b] N. Schmitt, K.-D. Lange, S. Sharma, N. Rawtani, C. Ponder, and S. Kounev. "The SPECpowerNext Benchmark Suite, Its Implementation and New Workloads from a Developers Perspective". In: *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. ICPE '21. Virtual Event, France: Association for Computing Machinery, 2021, pp. 225–232 (see pages 2, 49, 50).

[Sch+21c] N. Schmitt, R. Vobl, A. Brunnert, and S. Kounev. "Towards a Benchmark for Software Resource Efficiency". In: *Companion of the ACM/SPEC International Conference on Performance Engineering*. ICPE '21. Virtual Event, France: Association for Computing Machinery, 2021, pp. 179–182 (see page 175).

[SMM07] C. Seo, S. Malek, and N. Medvidovic. "An Energy Consumption Framework for Distributed Java-based Systems". In: *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering*. ASE '07. Atlanta, Georgia, USA: ACM, 2007, pp. 421–424 (see pages 4, 39, 40).

[SNM15] J. Singh, K. Naik, and V. Mahinthan. "Impact of Developer Choices on Energy Consumption of Software on Servers". In: *Procedia Computer Science* 62 (2015). Proceedings of the 2015 International Conference on Soft Computing and Software Engineering (SCSE'15), pp. 385–394 (see page 51).

[SBM09] K. Singh, M. Bhadauria, and S. A. McKee. "Real Time Power Estimation and Thread Scheduling via Performance Counters". In: *SIGARCH Comput. Archit. News* 37.2 (2009), pp. 46–55 (see pages 38, 110).

[Smi07] C. U. Smith. "Introduction to Software Performance Engineering: Origins and Outstanding Problems". In: *Formal Methods for Performance Evaluation: 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007, Bertinoro, Italy, May 28-June 2, 2007, Advanced Lectures*. Ed. by M. Bernardo and J. Hillston. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 395–428 (see page 175).

[Son+13] S. Song, C. Su, B. Rountree, and K. W. Cameron. "A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures". In: *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. 2013, pp. 673–686 (see page 110).

[SPE19] SPEC. *Beyond performance to power consumption*. 2019. (Visited on 10/06/2020) (see pages 2, 49).

[SPE13] SPEC. *Server Efficiency Rating Tool (SERT) Design Document*. Gainsville, VA, USA, 2013 (see pages 5, 21, 31, 32).

[SPE17] SPEC. *SPEC Chauffeur WDK*. Gainsville, VA, USA, 2017 (see page 22).

[SPE21a] SPEC. *SPEC CPU 2006*. 2021 (see page 45).

[SPE21b] SPEC. *SPEC Fair Use Rules*. 2021 (see page 12).

[SPE14] SPEC. *SPEC Power and Performance Benchmark Methodology*. Gainsville, VA, USA, 2014 (see pages 19, 144).

[SPE12] SPEC. *SPEC PTDaemon Design Document*. 2012 (see pages 22, 27).

[SPE21c] SPEC. *SPECjbb 2013*. 2021 (see page 45).

[Sta16] Statista. *Internet of Things (IoT) Connected Devices Installed Base Worldwide from 2015 to 2025*. 2016 (see page 1).

[Sti+15] C. Stier, A. Koziolek, H. Groenda, and R. Reussner. "Model-Based Energy Efficiency Analysis of Software Architectures". In: *Software Architecture*. Ed. by D. Weyns, R. Mirandola, and I. Crnkovic. Cham: Springer International Publishing, 2015, pp. 221–238 (see page 39).

[Sui21] P. T. Suite. *Documentation*. 2021 (see pages 85, 88, 207).

[Toc+19] K. Tocze, N. Schmitt, I. Brandic, A. Aral, and S. Nadjm-Tehrani. "Towards Edge Benchmarking: A Methodology for Characterizing Edge Workloads". In: *Proceedings of Workshop on Hot Topics in Cloud Computing Performance (HotCloudPerf) as part of FAS*(IEEE ICAC/SASO) conferences companion*. IEEE, 2019.

[TPC21] TPC. *TPC-H*. 2021 (see page 45).

[TPV17] A. Tripathi, I. Pathak, and D. P. Vidyarthi. "Energy Efficient VM Placement for Effective Resource Utilization using Modified Binary PSO". In: *The Computer Journal* 61.6 (2017), pp. 832–846. eprint: `https://academic.oup.com/comjnl/article-pdf/61/6/832/24979049/bxx096.pdf` (see pages 4, 16).

[Tsa+14] 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". In: *Future Generation Computer Systems* vol. 36 (2014), pp. 287–298 (see pages 38, 39).

[Umw21] Umweltbundesamt. *Indikator: Emission von Treibhausgasen*. 2021 (see page 2).

[Van+19] E. Van Eyk, J. Grohmann, S. Eismann, A. Bauer, L. Versluis, L. Toader, N. Schmitt, N. Herbst, C. Abad, and A. Iosup. "The SPEC-RG Reference Architecture for FaaS: From Microservices and Containers to Serverless Platforms". In: *IEEE Internet Computing* (2019) (see page 15).

[VAN08] A. Verma, P. Ahuja, and A. Neogi. "Power-aware Dynamic Placement of HPC Applications". In: *Proceedings of the 22Nd Annual International Conference on Supercomputing*. ICS '08. Island of Kos, Greece: ACM, 2008, pp. 175–184 (see page 40).

[Vög+18] C. Vögele, A. van Hoorn, E. Schulz, W. Hasselbring, and H. Krcmar. "WESSBAS: extraction of probabilistic workload specifications for load testing and performance prediction–a model-driven approach for session-based application systems". In: *Software & Systems Modeling* 17.2 (2018), pp. 443–477 (see page 176).

[WTM13] V. M. Weaver, D. Terpstra, and S. Moore. "Non-determinism and overcount on modern hardware performance counter implementations". In: *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2013, pp. 215–224 (see page 18).

[Wei+19] G. Wei, D. Qian, H. Yang, and Z. Luan. "Modeling Power Consumption of The Code Execution Using Performance Counters Statistics". In: *2019 20th International Conference on Parallel and Distributed Computing, Applications and Technologies* (*PDCAT*). 2019, pp. 381–385 (see page 46).

[WFP07] C. M. Woodside, G. Franks, and D. C. Petriu. "The Future of Software Performance Engineering". In: *Future of Software Engineering* (*FOSE '07*) (2007), pp. 171–187 (see page 175).

[Yas14] A. Yasin. "A Top-Down method for performance analysis and counters architecture". In: *2014 IEEE International Symposium on Performance Analysis of Systems and Software* (*ISPASS*). 2014, pp. 35–44 (see pages 110, 152).

[ZAM15] M. Zaman, A. Ahmadi, and Y. Makris. "Workload characterization and prediction: A pathway to reliable multi-core systems". In: *2015 IEEE 21st International On-Line Testing Symposium* (*IOLTS*). 2015, pp. 116–121 (see page 45).

[ZJH09] D. Zaparanuks, M. Jovic, and M. Hauswirth. "Accuracy of performance counter measurements". In: *2009 IEEE International Symposium on Performance Analysis of Systems and Software*. 2009, pp. 23–32 (see page 40).

[Zha+10] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. "Accurate online power estimation and automatic battery behavior based power model generation for smartphones". In: *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM. 2010, pp. 105–114 (see page 42).

Appendices

1 Phoronix Test Suite Benchmarks

The descriptions are taken verbatim from the test profiles provided by the Phoronix Test Suite [Sui21]. In case the benchmark provided configuration options to limit the CPU utilization, additional benchmark runs have been made. The configured CPU utilization levels are listed in parantheses in the configuration column of Table 1.

Table 1: Tests executed in the Phoronix Test Suite.

Name	Description	Configuration	Evaluation
aobench	Lightweight ambient occlusion renderer, written in C. The test profile is using a size of 2048 x 2048.	(50% 70% 100%)	Classification
aircrack-ng	Tool for assessing WiFi/WLAN network security.	(50% 70% 100%)	
blosc	A simple, compressed, fast and persistent data store library for C.	(50% 70% 100%)	Classification
bork	Small, cross-platform file encryption utility. It is written in Java and designed to be included along with the files it encrypts for long-term storage. This test measures the amount of time it takes to encrypt a sample file.	(50% 70% 100%)	Classification
botan	BSD-licensed cross-platform open-source C++ crypto library "cryptography toolkit" that supports most publicly known cryptographic algorithms. The project's stated goal is to be "the best option for cryptography in C++ by offering the tools necessary to implement a range of practical systems, such as TLS protocol, X.509 certificates, modern AEAD ciphers, PKCS#11 and TPM hardware support, password hashing, and post quantum crypto schemes.	1: AES-256 2) Blowfish 3) CAST-256 4) KASUMI 5) Twofish (50% 70% 100%)	
build-apache	This test times how long it takes to build the Apache HTTPD web server.	(50% 70% 100%)	
build-eigen	This test times how long it takes to build all Eigen examples. The Eigen examples are compiled serially. Eigen is a C++ template library for linear algebra.	(50% 70% 100%)	
build-erlang	This test times how long it takes to compile Erlang/OTP. Erlang is a programming language and run-time for massively scalable soft real-time systems with high availability requirements.		
build-ffmpeg	This test times how long it takes to build FFmpeg.	(50% 70% 100%)	
build-imagemagick	This test times how long it takes to build ImageMagick.	(50% 70% 100%)	
build-linux-kernel	This test times how long it takes to build the Linux kernel in a default configuration	(50% 70% 100%)	
build-mplayer	This test times how long it takes to build the MPlayer media player program.	(50% 70% 100%)	
byte	This is a test of BYTE.		
cachebench	This is a performance test of CacheBench, which is part of LLCbench. CacheBench is designed to test the memory and cache bandwidth performance	(50% 70% 100%)	
clomp	CLOMP is the C version of the Livermore OpenMP benchmark developed to measure OpenMP overheads and other performance impacts due to threading in order to influence future system designs. This particular test profile configuration is currently set to look at the OpenMP static schedule speed-up across all available CPU cores using the recommended test configuration.	(50% 70% 100%)	

compress-7zip	This is a test of 7-Zip using p7zip with its integrated benchmark feature or upstream 7-Zip for the Windows x64 build.	(50% 70% 100%)	
compress-gzip	This test measures the time needed to archive/compress two copies of the Linux 4.13 kernel source tree using Gzip compression.		
core-latency	This is a test of core-latency, which measures the latency between all core combinations on the system processor(s). Reported is the average latency.		
dacapobench	This test runs the DaCapo Benchmarks written in Java and intended to test system/CPU performance.	2) H2 3) Jython 4) Tradebeans	
dcraw	This test times how long it takes to convert several high-resolution RAW NEF image files to PPM image format using dcraw.	(50% 70% 100%)	
dolfyn	Dolfyn is a Computational Fluid Dynamics (CFD) code of modern numerical simulation techniques. The Dolfyn test profile measures the execution time of the bundled computational fluid dynamics demos that are bundled with Dolfyn.	(50% 70% 100%)	
ebizzy	This is a test of ebizzy, a program to generate workloads resembling web server workloads.	(50% 70% 100%)	
encode-ape	This test times how long it takes to encode a sample WAV file to Monkey's Audio APE format.	(50% 70% 100%)	
encode-mp3	LAME is an MP3 encoder licensed under the LGPL. This test measures the time required to encode a WAV file to MP3 format.	(50% 70% 100%)	
encode-opus	Opus is an open audio codec. Opus is a lossy audio compression format designed primarily for interactive real-time applications over the Internet. This test uses Opus-Tools and measures the time required to encode a WAV file to Opus.	(50% 70% 100%)	
espeak	This test times how long it takes the eSpeak speech synthesizer to read Project Gutenberg's The Outline of Science and output to a WAV file. This test profile is now tracking the eSpeak-NG version of eSpeak.		
etcpak	Etcpack is the self-proclaimed "fastest ETC compressor on the planet" with focused on providing open-source, very fast ETC and S3 texture compression support.	1: ETC1 (50% 70% 100%)	Classification
ffmpeg	This test uses FFmpeg for testing the system's audio/video encoding performance.	(50% 70% 100%)	Classification
ffte	FFTE is a package by Daisuke Takahashi to compute Discrete Fourier Transforms of 1-, 2- and 3- dimensional sequences of length $(2^p)*(3^q)*(5^r)$.	(50% 70% 100%)	
gcrypt	Libgcrypt is a general purpose cryptographic library developed as part of the GnuPG project. This is a benchmark of libgcrypt's integrated benchmark and is measuring the time to run the benchmark command with a cipher/mac/hash repetition count set for 50 times as simple, high level look at the overall crypto performance of the system under test.	(50% 70% 100%)	
graphics-magick	This is a test of GraphicsMagick with its OpenMP implementation that performs various imaging tests on a sample 6000x4000 pixel JPEG image.	1) HWB Color Space 2) Noise-Gaussian 3) Enhanced 4) Resizing 5) Rotate 6) Sharpen 7) Swirl (50% 70% 100%)	
hackbench	This is a benchmark of Hackbench, a test of the Linux kernel scheduler.	1: 1 Process 2: 2 Processes 3: 32 Processes 4: 1 Thread 5: 2 Threads	
himeno	The Himeno benchmark is a linear solver of pressure Poisson using a point-Jacobi method.	(50% 70% 100%)	

java-scimark2	This test runs the Java version of SciMark 2.0, which is a benchmark for scientific and numerical computing developed by programmers at the National Institute of Standards and Technology. This benchmark is made up of Fast Foruier Transform, Jacobi Successive Over-relaxation, Monte Carlo, Sparse Matrix Multiply, and dense LU matrix factorization benchmarks.	1) Composite 2) Fast Fourier Transform 3) Jacobi Successive Over-Relaxation 4) Monte Carlo 5) Sparse Matrix Multiply 6) Dense LU Matrix Factorization
john-the-ripper	This is a benchmark of John The Ripper, which is a password cracker.	1) MD5 2) Blowfish
ipc-benchmark	IPC_benchmark is a Linux inter-process communication benchmark.	1) Unnamed Pipe, 128 Bytes 2) Unnamed Pipe, 256 Bytes 3) Unnamed Pipe, 512 Bytes 4) Unnamed Pipe, 1024 Bytes 5) Unnamed Pipe, 2048 Bytes 6)Unnamed Pipe, 4096 Bytes 7) FIFO Named Pipee, 128 Bytes 8) FIFO Named Pipe, 256 Bytes 9) FIFO Named Pipe, 512 Bytes 10) FIFO Named Pipe, 1024 Bytes 11) FIFO Named Pipe, 2048 Bytes 12) FIFO Named Pipe, 4096 Bytes 13) Unnamed Unix Domain Socket, 128 Bytes 14) Unnamed Unix Domain Socket,256 Bytes 15) Unnamed Unix Domain Socket,512 Bytes 16) Unnamed Unix Domain Socket,1024 Bytes 17) Unnamed Unix Domain Socket, 2048 Bytes 18) Unnamed Unix Domain Socket, 4096 Bytes 19) TCP Socket, 128 Bytes 20) TCP Socket, 256 Bytes 21) TCP Socket, 512 Bytes 22) TCP Socket, 1024 Bytes 23) TCP Socket, 2048 Bytes 24) TCP Socket, 4096 Bytes
libgav1	Libgav1 is an AV1 decoder developed by Google for AV1 profile 0/1 compliance.	(50% 70% 100%) 1) Summer Nature 1080p
libraw	LibRaw is a RAW image decoder for digital camera photos. This test profile runs LibRaw's post-processing benchmark.	
liquid-dsp	LiquidSDR's Liquid-DSP is a software-defined radio (SDR) digital signal processing library. This test profile runs a multi-threaded benchmark of this SDR/DSP library focused on embedded platform usage.	1) 1 Thread 2) 2 Threads
mafft	This test performs an alignment of 100 pyruvate decarboxylase sequences.	
mencoder	This test uses mplayer's mencoder utility and the libavcodec family for testing the system's audio/video encoding performance.	

minion	Minion is an open-source constraint solver that is designed to be very scalable. This test profile uses Minion's integrated benchmarking problems to solve.	1) Graceful	
mpcbench	GNU MPC is a C library for the arithmetic of complex numbers.		
neat	NEAT is the Nebular Empirical Analysis Tool for empirical analysis of ionised nebulae, with uncertainty propagation.		
nettle	GNU Nettle is a low-level cryptographic library.	1) sha512 2) poly1305-aes 3) aes256 4) chacha	
ngspice	Ngspice is an open-source SPICE circuit simulator. Ngspice was originally based on the Berkeley SPICE electronic circuit simulator. Ngspice supports basic threading using OpenMP. This test profile is making use of the ISCAS 85 benchmark circuits.	1) C2670	
noise-level	This test measures background activity.		
openssl	OpenSSL is an open-source toolkit that implements SSL (Secure Sockets Layer) and TLS (Transport Layer Security) protocols. This test measures the RSA 4096-bit performance of OpenSSL.		
perl-benchmark	Perl benchmark suite that can be used to compare the relative speed of different versions of perl.	1) Pod2html 2) Interpreter	
polybench-c	PolyBench-C is a C-language polyhedral benchmark suite made at the Ohio State University.	1) 3 Matrix Multiplications 2) Correlation Computation 3) Covariance Computation https://github.com/phoronix-test-suite/test-profiles/tree/master/pts	
renaissance	Renaissance is a suite of benchmarks designed to test the Java JVM from Apache Spark to a Twitter-like service to Scala and other features.	1) Finagle HTTP Requests	
smallpt	Smallpt is a C++ global illumination renderer written in less than 100 lines of code. Global illumination is done via unbiased Monte Carlo path tracing and there is multi-threading support via the OpenMP library.		
stockfish	This is a test of Stockfish, an advanced C++11 chess benchmark that can scale up to 128 CPU cores.		
sudokut	This is a test of Sudokut, which is a Sudoku puzzle solver written in Tcl. This test measures how long it takes to solve 100 Sudoku puzzles.		
swet	Swet is a synthetic CPU/RAM benchmark, includes multi-processor test cases.		
synthmark	SynthMark is a cross platform tool for benchmarking CPU performance under a variety of real-time audio workloads. It uses a polyphonic synthesizer model to provide standardized tests for latency, jitter and computational throughput.		
system-decompress-bzip2	This test measures the time to decompress a Linux kernel tarball using BZIP2.		
system-decompress-gzip	This simple test measures the time to decompress a gzipped tarball (the Qt5 toolkit source package).		Classification, Execution, Modelling
system-decompress-xz	This test measures the time to decompress a Linux kernel tarball using XZ.		
system-decompress-zlib	This test measures the time to decompress a Linux kernel tarball using ZLIB.		
system-libxml2	This test measures the time to parse a random XML file with libxml2 via xmllint using the streaming API.		
tscp	This is a performance test of TSCP, Tom Kerrigan's Simple Chess Program, which has a built-in performance benchmark.		

ttsiod-renderer	A portable GPL 3D software renderer that supports OpenMP and Intel Threading Building Blocks with many different rendering modes. This version does not use OpenGL but is entirely CPU/software based.		Classification, Execution, Modelling
vpxenc	This is a standard video encoding performance test of Google's libvpx library and the vpxenc command for the VP9/WebM format using a sample 1080p video.		
webp	This is a test of Google's libwebp with the cwebp image encode utility and using a sample 6000x4000 pixel JPEG image as the input.	1) Default 2) Quality 100 3) Quality 100, Highest Compression 4) Quality 100, Lossless 5) Quality 100, Lossless, Highest Compression	
x264	This is a simple test of the x264 encoder run on the CPU (OpenCL support disabled) with a sample video file.		Classification
m-queens	A solver for the N-queens problem with multi-threading support via the OpenMP library.		Classification, Execution, Modelling
n-queens	This is a test of the OpenMP version of a test that solves the N-queens problem. The board problem size is 18.		Classification, Execution, Modelling
parboil	The Parboil Benchmarks from the IMPACT Research Group at University of Illinois are a set of throughput computing applications for looking at computing architecture and compilers. Parboil test-cases support OpenMP, OpenCL, and CUDA multi-processing environments. However, at this time the test profile is just making use of the OpenMP and OpenCL test workloads.	1) OpenMP MRI Gridding	Classification, Execution, Modelling
primesieve	Primesieve generates prime numbers using a highly optimized sieve of Eratosthenes implementation. Primesieve benchmarks the CPU's L1/L2 cache performance.		
rodinia	Rodinia is a suite focused upon accelerating compute-intensive applications with accelerators. CUDA, OpenMP, and OpenCL parallel models are supported by the included applications. This profile utilizes select OpenCL, NVIDIA CUDA and OpenMP test binaries at the moment.		
scimark2	This test runs the ANSI C version of SciMark 2.0, which is a benchmark for scientific and numerical computing developed by programmers at the National Institute of Standards and Technology. This test is made up of Fast Foruier Transform, Jacobi Successive Over-relaxation, Monte Carlo, Sparse Matrix Multiply, and dense LU matrix factorization benchmarks.	1) Composite 2) Fast Fourier Transform 3) Jacobi Successive Over-Relaxation 4) Monte Carlo 5) Sparse Matrix Multiply 6) Dense LU Matrix Factorization	
t-test1	This is a test of t-test1 for basic memory allocator benchmarks. Note this test profile is currently very basic and the overall time does include the warmup time of the custom t-test1 compilation. Improvements welcome.	1) 1 Thread 2) 2 Threads	
mbw	This is a basic/simple memory (RAM) bandwidth benchmark for memory copy operations.	Classification, Execution, Modelling 1) Memory Copy, 128 MiB 2) Memory Copy, 512 MiB 3) Memory Copy, 1024 MiB 4) Memory Copy, 4096 MiB 5) Memory Copy (Fixed Block Size), 128 MiB 6) Memory Copy (Fixed Block Size), 512 MiB 7) Memory Copy (Fixed Block Size), 1024 MiB 8) Memory Copy (Fixed Block Size), 4096 MiB	

ramspeed	This benchmark tests the system memory (RAM) performance.	1) Copy, Integer 2) Scale, Integer 3) Add, Integer 4) Triad, Integer 5) Average, Integer 6) Copy, Floating Point 7) Scale, Floating Point 8) Add, Floating Point 9) Triad, Floating Point 10) Average, Floating Point
stream	This is a benchmark of Stream, the popular system memory (RAM) benchmark.	1) Copy 2) Scale 3) Add 4) Triad
tinymembench	This benchmark tests the system memory (RAM) performance.	
stressapptest	This is a pass/fail benchmark of stressapptest (Stressful Application Test) for verifying memory/RAM stability of the system.	1) 1 Thread, 32MB 2) 1 Thread, 64MB 3) 1 Thread,128MB 4) 1 Thread, 256MB 5) 1 Thread, 512MB 6) 1 Thread, 1GB 7) 1 Thread, 2GB 8) 1 Thread, 4GB 9) 2 Threads, 32MB 10) 2 Threads, 64MB 11) 2 Threads, 128MB 12) 2 Threads, 256MB 13) 2 Threads, 512MB 14) 2 Threads, 1GB 15) 2 Threads, 2GB 16) 2 Threads, 4GB

iozone	The IOzone benchmark tests the hard disk drive / file-system performance.	1) Record Size 4Kb, File Size 512MB, Write 2) Record Size 64Kb, File Size 512MB, Write 3) Record Size 1MB, File Size 512MB, Write 4) Record Size 2MB, File Size 512MB, Write 5) Record Size 4Kb, File Size 2GB, Write 6) Record Size 64Kb, File Size 2GB, Write 7) Record Size 1MB, File Size 2GB, Write 8) Record Size 2MB, File Size 2GB, Write 9) Record Size 4Kb, File Size 4GB, Write 10) Record Size 64Kb, File Size 4GB, Write 11) Record Size 1MB, File Size 4GB, Write 12) Record Size 2MB, File Size 4GB, Write 13) Record Size 4Kb, File Size 512MB, Read 14) Record Size 64Kb, File Size 512MB, Read 15) Record Size 1MB, File Size 512MB, Read 16) Record Size 2MB, File Size 512MB, Read 17) Record Size 4Kb, File Size 2GB, Read 18) Record Size 64Kb, File Size 2GB, Read 19) Record Size 1MB, File Size 2GB, Read 20) Record Size 2MB, File Size 2GB, Read 21) Record Size 4Kb, File Size 4GB, Read 22) Record Size 64Kb, File Size 4GB, Read 23) Record Size 1MB, File Size 4GB, Read 24) Record Size 2MB, File Size 4GB, Read
dbench	Dbench is a benchmark designed by the Samba project as a free alternative to net-bench, but dbench contains only file-system calls for testing the disk performance.	1) 1 Client 2) 6 Clients 3) 12 Clients 4) 48 Clients 5) 128 Clients 6) 256 Clients
fio	Fio is an advanced disk benchmark that depends upon the kernel's AIO access library.	1) Random Read, Linux AIO (Engine), 4MB 2) Random Read, Linux AIO (Engine), 8MB 3) Random Write, Linux AIO (Engine), 4MB 4) Random Write, Linux AIO (Engine), 8MB

fs-mark	FS_Mark is designed to test a system's file-system performance.	1) 1000 Files, 1MB Size 2) 1000 Files, 1MB Size, No Sync/FSync 4) 4000 Files, 32 Sub Dirs, 1MB Size	Classification, Execution, Modelling
leveldb	LevelDB is a key-value storage library developed by Google that supports making use of Snappy for data compression and has other modern features.	1) Sequential Fill 2) Random Fill 3) Overwrite 4) Fill Sync 5) Random Read 6) Random Delete 7) Hot Read 8) Seek Random	
postmark	This is a test of NetApp's PostMark benchmark designed to simulate small-file testing similar to the tasks endured by web and mail servers. This test profile will set PostMark to perform 25,000 transactions with 500 files simultaneously with the file sizes ranging between 5 and 512 kilobytes.		Classification
sqlite	This is a simple benchmark of SQLite. At present this test profile just measures the time to perform a pre-defined number of insertions on an indexed database.	1) 1 Thread 2) 8 Threads 3) 32 Threads 4) 64 Threads 5) 128 Threads	
tiobench	Tiotester (Threaded I/O Tester) benchmarks the hard disk drive / file-system performance.	1) Write, Random Write, Read, Random Read, 256MB per thread, 16 Threads	Classification, Execution, Modelling
unpack-linux	This test measures how long it takes to extract the .tar.xz Linux kernel package.		Classification, Execution, Modelling
apache	This is a test of ab, which is the Apache benchmark program. This test profile measures how many requests per second a given system can sustain when carrying out 1,000,000 requests with 100 requests being carried out concurrently.		
apache-siege	This is a test of the Apache web server performance being facilitated by the Siege web server benchmark program.	1) 200 Users 2) 250 Users 3) 500 Users	
brl-cad	BRL-CAD is a cross-platform, open-source solid modeling system with built-in benchmark mode.		
git	This test measures the time needed to carry out some sample Git operations on an example, static repository that happens to be a copy of the GNOME GTK tool-kit repository.		Classification
hbase	This is a benchmark of the Apache HBase non-relational distributed database system inspired from Google's Bigtable.	1) Random Write, Async Random Write, Random Read, Async Random Read, Sequential Write, Sequential Read, Increment, 1 Client	
hint	This test runs the U.S. Department of Energy's Ames Laboratory Hierarchical INTegration (HINT) benchmark.		
nginx	This is a test of ab, which is the Apache Benchmark program running against nginx. This test profile measures how many requests per second a given system can sustain when carrying out 2,000,000 requests with 500 requests being carried out concurrently.		
pgbench	This is a simple benchmark of PostgreSQL using pgbench.	1) Scaling Factor 100, 1 Client, 50 Clients, 100 Clients, 250 Clients, Read and Write	Classification
php	Various small PHP micro-benchmarks.	1) Zend bench, Zend micro_bench	

phpbench	PHPBench is a benchmark suite for PHP. It performs a large number of simple tests in order to bench various aspects of the PHP interpreter. PHPBench can be used to compare hardware, operating systems, PHP versions, PHP accelerators and caches, compiler options, etc.	
pybench	This test profile reports the total time of the different average timed test results from PyBench. PyBench reports average test times for different functions such as BuiltinFunctionCalls and NestedForLoops, with this total result providing a rough estimate as to Python's average performance on a given system. This test profile runs PyBench each time for 20 rounds.	
pyperformance	PyPerformance is the reference Python performance benchmark suite.	1) 2to3, chaos, crypto_pyaes, django_template, float, go, json_loads, nbody, pathlib, pickle_pure_python, python_startup, raytrace, regex_compile
redis	Redis is an open-source in-memory data structhttps://github.com/phoronix-test-suite/test-profiles/tree/master/ptsure store, used as a database, cache, and message broker.	1) SET, GET, LPUSH, LPOP, SADD
sqlite-speedtest	This is a benchmark of SQLite's speedtest1 benchmark program with an increased problem size of 1,000.	
tjbench	tjbench is a JPEG decompression/compression benchmark part of libjpeg-turbo.	
tnn	TNN is an open-source deep learning reasoning framework developed by Tencent.	1) DenseNet 2) MobileNet v2
unpack-firefox	This simple test profile measures how long it takes to extract the .tar.xz source package of the Mozilla Firefox Web Browser.	
stress-ng	Stress-NG is a Linux stress tool developed by Colin King of Canonical.	1) CPU Stress 2) Crypto 3) Memory Copying 4) Glibc Qsort Data Sorting 5) Glibc C String Functions 6) Vector Math 7) Matrix Math 8) Forking 9) System V Message Passing 10) Semaphores 11) Socket Activity 12) Context Switching 13) Atomic 14) CPU Cache 15) Malloc 16) MEMFD 17) MMAP 18) NUMA 19) RdRand 20) SENDFILE

2 Stress-ng Benchmarks

Table 2: Executed stress-ng configurations.

Stressed subsystem	Configuration
CPU	–cpu 1 –cpu-load 10% -t 10m
	–cpu 1 –cpu-load 15% -t 10m
	–cpu 1 –cpu-load 20% -t 10m
	–cpu 1 –cpu-load 25% -t 10m
	–cpu 1 –cpu-load 30% -t 10m
	–cpu 1 –cpu-load 35% -t 10m
	–cpu 1 –cpu-load 40% -t 10m
	–cpu 1 –cpu-load 45% -t 10m
	–cpu 1 –cpu-load 50% -t 10m
	–cpu 1 –cpu-load 55% -t 10m
	–cpu 1 –cpu-load 60% -t 10m
	–cpu 1 –cpu-load 65% -t 10m
	–cpu 1 –cpu-load 70% -t 10m
	–cpu 1 –cpu-load 75% -t 10m
	–cpu 1 –cpu-load 80% -t 10m
	–cpu 1 –cpu-load 85% -t 10m
	–cpu 1 –cpu-load 90% -t 10m
	–cpu 1 –cpu-load 95% -t 10m
	–cpu 1 –cpu-load 100% -t 10m
	–cpu 1 –cpu-load 18% -t 4m
	–cpu 1 –cpu-load 22% -t 4m
	–cpu 1 –cpu-load 52% -t 4m
	–cpu 1 –cpu-load 73% -t 4m
	–cpu 1 –cpu-load 77% -t 4m
	–cpu 1 –cpu-load 98% -t 4m
Memory	–vm 1 –vm-bytes 5% –vm-method all -t 5m
	–vm 1 –vm-bytes 10% –vm-method all -t 5m
	–vm 1 –vm-bytes 15% –vm-method all -t 5m
	–vm 1 –vm-bytes 20% –vm-method all -t 5m
	–vm 1 –vm-bytes 25% –vm-method all -t 5m
	–vm 1 –vm-bytes 30% –vm-method all -t 5m
	–vm 1 –vm-bytes 35% –vm-method all -t 5m
	–vm 1 –vm-bytes 40% –vm-method all -t 5m
	–vm 1 –vm-bytes 45% –vm-method all -t 5m
	–vm 1 –vm-bytes 50% –vm-method all -t 5m
	–vm 1 –vm-bytes 55% –vm-method all -t 5m
	–vm 1 –vm-bytes 60% –vm-method all -t 5m
	–vm 1 –vm-bytes 65% –vm-method all -t 5m
	–vm 1 –vm-bytes 70% –vm-method all -t 5m
	–vm 1 –vm-bytes 75% –vm-method all -t 5m
	–vm 1 –vm-bytes 85% –vm-method all -t 5m
	–vm 1 –vm-bytes 90% –vm-method all -t 5m
	–vm 1 –vm-bytes 100% –vm-method all -t 5m
	–brk 2 –stack 2 –bigheap 2 -t 4m
	–brk 0 –stack 0 –bigheap 0 -t 4m
	–vm 2 –vm-bytes 5G –mmap 2 –mmap-bytes 5G –page-in -t 4m
	–vm 2 –vm-bytes 100% -t 5m
	–vm 2 –vm-bytes 75% -t 4m
	–vm 2 –vm-bytes 98% -t 4m
	–vm 1 –vm-bytes 100% -t 4m
	–vm 1 –vm-bytes 92% -t 4m
	–vm 1 –vm-bytes 80% -t 4m
	–vm 1 –vm-bytes 76% -t 4m
	–shm 100 -t 4m
	–shm 110 -t 4m
	–shm 120 -t 4m
	–shm 130 -t 4m
	–shm 140 -t 4m
	–shm 150 -t 4m
	–shm 200 -t 4m
	–shm 400 -t 4m
I/O	–hdd 1000 –hdd-bytes 50% -t 4m
	–hdd 1000 –hdd-bytes 65% -t 4m

3 EC12 Performance Events

The descriptions of the EC12 performance events are taken verbatim from the CPU measurement facility utilities.

Table 3: Performance events recorded on the IBM EC12.

Event	Description
CPU_CYCLES	Cycle Count
INSTRUCTIONS	Instruction Count
L1I_DIR_WRITES	Level-1 I-Cache Directory Write Count
L1I_PENALTY_CYCLES	Level-1 I-Cache Penalty Cycle Count
L1D_DIR_WRITES	Level-1 D-Cache Directory Write Count
L1D_PENALTY_CYCLES	Level-1 D-Cache Penalty Cycle Count
PRNG_FUNCTIONS	Total number of the PRNG functions issued by the CPU
PRNG_CYCLES	Total number of CPU cycles when the DEA/AES coprocessor is busy performing PRNG functions issued by the CPU
PRNG_BLOCKED_FUNCTIONS	Total number of the PRNG functions that are issued by the CPU and are blocked because the DEA/AES coprocessor is busy performing a function issued by another CPU
PRNG_BLOCKED_CYCLES	Total number of CPU cycles blocked for the PRNG functions issued by the CPU because the DEA/AES coprocessor is busy performing a function issued by another CPU
SHA_FUNCTIONS	Total number of SHA functions issued by the CPU
SHA_CYCLES	Total number of CPU cycles when the SHA coprocessor is busy performing the SHA functions issued by the CPU
SHA_BLOCKED_FUNCTIONS	Total number of the SHA functions that are issued by the CPU and are blocked because the SHA coprocessor is busy performing a function issued by another CPU
SHA_BLOCKED_CYCLES	Total number of CPU cycles blocked for the SHA functions issued by the CPU because the SHA coprocessor is busy performing a function issued by another CPU
DEA_FUNCTIONS	Total number of the DEA functions issued by the CPU
DEA_CYCLES	Total number of CPU cycles when the DEA/AES coprocessor is busy performing the DEA functions issued by the CPU
DEA_BLOCKED_FUNCTIONS	Total number of the DEA functions that are issued by the CPU and are blocked because the DEA/AES coprocessor is busy performing a function issued by another CPU
DEA_BLOCKED_CYCLES	Total number of CPU cycles blocked for the DEA functions issued by the CPU because the DEA/AES coprocessor is busy performing a function issued by another CPU
AES_FUNCTIONS	Total number of AES functions issued by the CPU
AES_CYCLES	Total number of CPU cycles when the DEA/AES coprocessor is busy performing the AES functions issued by the CPU
AES_BLOCKED_FUNCTIONS	Total number of AES functions that are issued by the CPU and are blocked because the DEA/AES coprocessor is busy performing a function issued by another CPU
AES_BLOCKED_CYCLES	Total number of CPU cycles blocked for the AES functions issued by the CPU because the DEA/AES coprocessor is busy performing a function issued by another CPU
DTLB1_MISSES	Level-1 Data TLB miss in progress. Incremented by one for every cycle a DTLB1 miss is in progress
ITLB1_MISSES	Level-1 Instruction TLB miss in progress. Incremented by one for every cycle a ITLB1 miss is in progress
L1D_L2I_SOURCED_WRITES	A directory write to the Level-1 Data cache directory where the returned cache line was sourced from the Level-2 Instruction cache
L1I_L2I_SOURCED_WRITES	A directory write to the Level-1 Instruction cache directory where the returned cache line was sourced from the Level-2 Instruction cache
L1D_L2D_SOURCED_WRITES	A directory write to the Level-1 Data cache directory where the returned cache line was sourced from the Level-2 Data cache.
DTLB1_WRITES	A translation entry has been written to the Level-1 Data Translation Lookaside Buffer
L1D_LMEM_SOURCED_WRITES	A directory write to the Level-1 Data cache where the installed cache line was sourced from memory that is attached to the same book as the Data cache (Local Memory)
L1I_LMEM_SOURCED_WRITES	A directory write to the Level-1 Instruction cache where the installed cache line was sourced from memory that is attached to the same book as the Instruction cache (Local Memory)
L1D_RO_EXCL_WRITES	A directory write to the Level-1 D-Cache where the line was originally in a Read-Only state in the cache but has been updated to be in the Exclusive state that allows stores to the cache line
DTLB1_HPAGE_WRITES	A translation entry has been written to the Level-1 Data Translation Lookaside Buffer for a one-megabyte page
ITLB1_WRITES	A translation entry has been written to the Level-1 Instruction Translation Lookaside Buffer
TLB2_PTE_WRITES	A translation entry has been written to the Level-2 TLB Page Table Entry arrays
TLB2_CRSTE_HPAGE_WRITES	A translation entry has been written to the Level-2 TLB Common Region Segment Table Entry arrays for a one-megabyte large page translation
TLB2_CRSTE_WRITES	A translation entry has been written to the Level-2 TLB Common Region Segment Table Entry arrays

L1D_ONCHIP_L3_SOURCED_WRITES	A directory write to the Level-1 Data cache directory where the returned cache line was sourced from an On Chip Level-3 cache without intervention.
L1D_OFFCHIP_L3_SOURCED_WRITES	A directory write to the Level-1 Data cache directory where the returned cache line was sourced from an Off Chip/On Book Level-3 cache without intervention
L1D_OFFBOOK_L3_SOURCED_WRITES	A directory write to the Level-1 Data cache directory where the returned cache line was sourced from an Off Book Level-3 cache without intervention
L1D_ONBOOK_L4_SOURCED_WRITES	A directory write to the Level-1 Data cache directory where the returned cache line was sourced from an On Book Level-4 cache
L1D_OFFBOOK_L4_SOURCED_WRITES	A directory write to the Level-1 Data cache directory where the returned cache line was sourced from an Off Book Level-4 cache
L1D_ONCHIP_L3_SOURCED_WRITES_IV	A directory write to the Level-1 Data cache directory where the returned cache line was sourced from a On Chip Level-3 cache with intervention
L1D_OFFCHIP_L3_SOURCED_WRITES_IV	A directory write to the Level-1 Data cache directory where the returned cache line was sourced from an Off Chip/On Book Level-3 cache with intervention
L1D_OFFBOOK_L3_SOURCED_WRITES_IV	A directory write to the Level-1 Data cache directory where the returned cache line was sourced from an Off Book Level-3 cache with intervention
L1I_ONCHIP_L3_SOURCED_WRITES	A directory write to the Level-1 Instruction cache directory where the returned cache line was sourced from an On Chip Level-3 cache without intervention
L1I_OFFCHIP_L3_SOURCED_WRITES	A directory write to the Level-1 Instruction cache directory where the returned cache line was sourced from an Off Chip/On Book Level-3 cache without intervention
L1I_OFFBOOK_L3_SOURCED_WRITES	A directory write to the Level-1 Instruction cache directory where the returned cache line was sourced from an Off Book Level-3 cache without intervention
L1I_ONBOOK_L4_SOURCED_WRITES	A directory write to the Level-1 Instruction cache directory where the returned cache line was sourced from an On Book Level-4 cache
L1I_OFFBOOK_L4_SOURCED_WRITES	A directory write to the Level-1 Instruction cache directory where the returned cache line was sourced from an Off Book Level-4 cache
L1I_ONCHIP_L3_SOURCED_WRITES_IV	A directory write to the Level-1 Instruction cache directory where the returned cache line was sourced from an On Chip Level-3 cache with intervention
L1I_OFFCHIP_L3_SOURCED_WRITES_IV	A directory write to the Level-1 Instruction cache directory where the returned cache line was sourced from an Off Chip/On Book Level-3 cache with intervention
L1I_OFFBOOK_L3_SOURCED_WRITES_IV	A directory write to the Level-1 Instruction cache directory where the returned cache line was sourced from an Off Book Level-3 cache with intervention

4 Detailed PET Results

Table 4: PET results for the maximum throughput measurement. Values are accumulated over the measurement period of 240 s.

Server		Context Switches	Interrupts	L3 Misses	L3 Hits
	Events per Transaction	114	29	79 329	27 808
A	Transactions	814 328	993 221	1 074 624	957 024
	Baseline	3.42×10^5	1.60×10^5	1.02×10^5	2.16×10^7
	Maximum	7.80×10^7	6.01×10^7	1.86×10^8	1.93×10^{10}
B	Transactions	1 190 412	1 935 652	1 557 588	1 384 136
	Baseline	6.47×10^5	1.88×10^5	1.63×10^6	6.44×10^7
	Maximum	1.03×10^8	5.62×10^7	1.88×10^9	3.03×10^{10}
C	Transactions	2 149 538	956 897	935 376	746 517
	Baseline	3.48×10^5	1.58×10^5	2.07×10^7	2.20×10^7
	Maximum	1.85×10^8	2.77×10^7	1.06×10^9	8.76×10^9

Server		Mem-Reads	Mem-Writes	Instructions
	Events per Transaction	25 934	23 026	12 307 786
A	Transactions	1 004 181	998 525	1 015 981
	Baseline	1.80×10^8	4.35×10^8	1.15×10^{10}
	Maximum	1.38×10^{12}	1.22×10^{12}	1.04×10^{13}
B	Transactions	1 569 190	1 921 913	1 057 970
	Baseline	8.59×10^7	1.29×10^8	4.46×10^9
	Maximum	2.16×10^{12}	2.36×10^{12}	1.08×10^{13}
C	Transactions	1 123 700	1 647 394	616 909
	Baseline	7.63×10^7	3.33×10^7	1.62×10^9
	Maximum	1.54×10^{12}	2.01×10^{12}	6.31×10^{12}