Jóakim Gunnarsson v. Kistowski

Measuring, Rating, and Predicting the Energy Efficiency of Servers

Dissertation, Julius-Maximilians-Universität Würzburg
Fakultät für Mathematik und Informatik, 2019

Gutachter: Prof. Dr.-Ing. Samuel Kounev, Julius-Maximilians-Universität Würzburg
Prof. Dr. Hartmut Schmeck, Karlsruher Institut für Technologie

Datum der mündlichen Prüfung: 1. März 2019

Abstract

Energy efficiency of computing systems has become an increasingly important issue over the last decades. In 2015, data centers were responsible for 2% of the world's greenhouse gas emissions, which is roughly the same as the amount produced by air travel. In addition to these environmental concerns, power consumption of servers in data centers results in significant operating costs, which increase by at least 10% each year. To address this challenge, the U.S. EPA and other government agencies are considering the use of novel measurement methods in order to label the energy efficiency of servers.

The energy efficiency and power consumption of a server is subject to a great number of factors, including, but not limited to, hardware, software stack, workload, and load level. This huge number of influencing factors makes measuring and rating of energy efficiency challenging. It also makes it difficult to find an energy-efficient server for a specific use-case. Among others, server provisioners, operators, and regulators would profit from information on the servers in question and on the factors that affect those servers' power consumption and efficiency. However, we see a lack of measurement methods and metrics for energy efficiency of the systems under consideration. Even assuming that a measurement methodology existed, making decisions based on its results would be challenging. Power prediction methods that make use of these results would aid in decision making. They would enable potential server customers to make better purchasing decisions and help operators predict the effects of potential reconfigurations.

Existing energy efficiency benchmarks cannot fully address these challenges, as they only measure single applications at limited sets of load levels. In addition, existing efficiency metrics are not helpful in this context, as they are usually a variation of the simple *performance per power* ratio, which is only applicable to single workloads at a single load level. Existing data center efficiency metrics, on the other hand, express the efficiency of the data center space and power infrastructure, not focusing on the efficiency of the servers themselves. Power prediction methods for not-yet-available systems that could make use of the results provided by a comprehensive power rating methodology are also lacking. Existing power prediction models for hardware designers

have a very fine level of granularity and detail that would not be useful for data center operators.

This thesis presents a measurement and rating methodology for energy efficiency of servers and an energy efficiency metric to be applied to the results of this methodology. We also design workloads, load intensity and distribution models, and mechanisms that can be used for energy efficiency testing. Based on this, we present power prediction mechanisms and models that utilize our measurement methodology and its results for power prediction.

Specifically, the six major contributions of this thesis are:

- We present a measurement methodology and metrics for energy efficiency rating of servers that use multiple, specifically chosen workloads at different load levels for a full system characterization.

  We evaluate the methodology and metric with regard to their reproducibility, fairness, and relevance. We investigate the power and performance variations of test results and show fairness of the metric through a mathematical proof and a correlation analysis on a set of 385 servers. We evaluate the metric's relevance by showing the relationships that can be established between metric results and third-party applications.

- We create models and extraction mechanisms for load profiles that vary over time, as well as load distribution mechanisms and policies. The models are designed to be used to define arbitrary dynamic load intensity profiles that can be leveraged for benchmarking purposes. The load distribution mechanisms place workloads on computing resources in a hierarchical manner.

  Our load intensity models can be extracted in less than 0.2 seconds and our resulting models feature a median modeling error of 12.7% on average. In addition, our new load distribution strategy can save up to 10.7% of power consumption on a single server node.

- We introduce an approach to create small-scale workloads that emulate the power consumption-relevant behavior of large-scale workloads by approximating their CPU performance counter profile, and we introduce TeaStore, a distributed, micro-service-based reference application. TeaStore can be used to evaluate power and performance model accuracy, elasticity of cloud auto-scalers, and the effectiveness of power saving mechanisms for distributed systems.

  We show that we are capable of emulating the power consumption behavior of realistic workloads with a mean deviation less than 10% and down to 0.2 watts (1%). We demonstrate the use of TeaStore in the context of performance model extraction and cloud auto-scaling also showing that it may generate workloads with different effects on the power consumption of the system under consideration.

- We present a method for automated selection of interpolation strategies for performance and power characterization. We also introduce a configuration approach for polynomial interpolation functions of varying degrees that improves prediction accuracy for system power consumption for a given system utilization.

  We show that, in comparison to regression, our automated interpolation method selection and configuration approach improves modeling accuracy by 43.6% if additional reference data is available and by 31.4% if it is not.

- We present an approach for explicit modeling of the impact a virtualized environment has on power consumption and a method to predict the power consumption of a software application. Both methods use results produced by our measurement methodology to predict the respective power consumption for servers that are otherwise not available to the person making the prediction.

  Our methods are able to predict power consumption reliably for multiple hypervisor configurations and for the target application workloads. Application workload power prediction features a mean average absolute percentage error of 9.5%.

- Finally, we propose an end-to-end modeling approach for predicting the power consumption of component placements at run-time. The model can also be used to predict the power consumption at load levels that have not yet been observed on the running system.

  We show that we can predict the power consumption of two different distributed web applications with a mean absolute percentage error of 2.2%. In addition, we can predict the power consumption of a system at a previously unobserved load level and component distribution with an error of 1.2%.

The contributions of this thesis already show a significant impact in science and industry. The presented efficiency rating methodology, including its metric, have been adopted by the U.S. EPA in the latest version of the ENERGY STAR Computer Server program. They are also being considered by additional regulatory agencies, including the EU Commission and the China National Institute of Standardization. In addition, the methodology's implementation and the underlying methodology itself have already found use in several research publications.

Regarding future work, we see a need for new workloads targeting specialized server hardware. At the moment, we are witnessing a shift in execution hardware to specialized machine learning chips, general purpose GPU computing, FPGAs being embedded into compute servers, etc. To ensure that our measurement methodology remains relevant, workloads covering these areas are required. Similarly, power prediction models must be extended to cover these new scenarios.

Zusammenfassung

In den vergangenen Jahrzehnten hat die Energieeffizienz von Computersystemen stark an Bedeutung gewonnen. Bereits 2015 waren Rechenzentren für 2% der weltweiten Treibhausgasemissionen verantwortlich, was mit der durch den Flugverkehr verursachten Treibhausgasmenge vergleichbar ist. Dabei wirkt sich der Stromverbrauch von Rechenzentren nicht nur auf die Umwelt aus, sondern verursacht auch erhebliche, jährlich um mindestens 10% steigende, Betriebskosten. Um sich diesen Herausforderungen zu stellen, erwägen die U.S. EPA und andere Behörden die Anwendung von neuartigen Messmethoden, um die Energieeffizienz von Servern zu bestimmen und zu zertifizieren.

Die Energieeffizienz und der Stromverbrauch eines Servers wird von vielen verschiedenen Faktoren, u.a. der Hardware, der zugrundeliegenden Ausführungssoftware, der Arbeitslast und der Lastintensität, beeinflusst. Diese große Menge an Einflussfaktoren führt dazu, dass die Messung und Bewertung der Energieeffizienz herausfordernd ist, was die Auswahl von energieeffizienten Servern für konkrete Anwendungsfälle erheblich erschwert. Informationen über Server und ihre Energieeffizienz bzw. ihren Stromverbrauch beeinflussenden Faktoren wären für potentielle Kunden von Serverhardware, Serverbetreiber und Umweltbehörden von großem Nutzen. Im Allgemeinen mangelt es aber an Messmethoden und Metriken, welche die Energieeffizienz von Servern in befriedigendem Maße erfassen und bewerten können. Allerdings wäre es selbst unter der Annahme, dass es solche Messmethoden gäbe, dennoch schwierig Entscheidungen auf Basis ihrer Ergebnisse zu fällen. Um derartige Entscheidungen zu vereinfachen, wären Methoden zur Stromverbrauchsvorhersage hilfreich, um es potentiellen Serverkunden zu ermöglichen bessere Kaufentscheidungen zu treffen und Serverbetreibern zu helfen, die Auswirkungen möglicher Rekonfigurationen vorherzusagen.

Existierende Energieeffizienzbenchmarks können diesen Herausforderungen nicht vollständig begegnen, da sie nur einzelne Anwendungen bei wenigen Lastintensitätsstufen ausmessen. Auch sind die vorhandenen Energieeffizienzmetriken in diesem Kontext nicht hilfreich, da sie normalerweise nur eine Variation des einfachen Verhältnisses von *Performanz zu Stromverbrauch* darstellen, welches nur auf einzelne Arbeitslasten bei einer einzigen gemessenen Lastintensität angewandt werden kann. Im Gegensatz dazu beschreiben die

existierenden Rechenzentrumseffizienzmetriken lediglich die Platz- und Strominfrastruktureffizienz von Rechenzentren und bewerten nicht die Effizienz der Server als solche. Methoden zur Stromverbrauchsvorhersage noch nicht für Kunden verfügbarer Server, welche die Ergebnisse einer ausführlichen Stromverbrauchsmessungs- und Bewertungsmethodologie verwenden, gibt es ebenfalls nicht. Stattdessen existieren Stromverbrauchsvorhersagemethoden und Modelle für Hardwaredesigner und Hersteller. Diese Methoden sind jedoch sehr feingranular und erfordern Details, welche für Rechenzentrumsbetreiber nicht verfügbar sind, sodass diese keine Vorhersage durchführen können.

In dieser Arbeit werden eine Energieeffizienzmess- und Bewertungsmethodologie für Server und Energieeffizienzmetriken für diese Methodologie vorgestellt. Es werden Arbeitslasten, Lastintensitäten und Lastverteilungsmodelle und -mechanismen, die für Energieeffizienzmessungen und Tests verwendet werden können, entworfen. Darauf aufbauend werden Mechanismen und Modelle zur Stromverbrauchsvorhersage präsentiert, welche diese Messmethodologie und die damit produzierten Ergebnisse verwenden. Die sechs Hauptbeiträge dieser Arbeit sind:

- Eine Messmethodologie und Metriken zur Energieeffizienzbewertung von Servern, die mehrere, verschiedene Arbeitslasten unter verschiedenen Lastintensitäten ausführt, um die beobachteten Systeme vollständig zu charakterisieren.

  Diese Methodologie wird im Bezug auf ihre Wiederholbarkeit, Fairness und Relevanz evaluiert. Es werden die Stromverbrauchs- und Performanzvariationen von wiederholten Methodologieausführungen untersucht und die Fairness der Methodologie wird durch mathematische Beweise und durch eine Korrelationsanalyse anhand von Messungen auf 385 Servern bewertet. Die Relevanz der Methodologie und der Metrik wird gezeigt, indem Beziehungen zwischen Metrikergebnissen und der Energieeffizienz von anderen Serverapplikationen untersucht werden.

- Modelle und Extraktionsverfahren für sich mit der Zeit verändernde Lastprofile, sowie Lastverteilungsmechanismen und -regeln. Die Modelle können dazu verwendet werden, beliebige Lastintensitätsprofile, die zum Benchmarking verwendet werden können, zu entwerfen. Die Lastverteilungsmechanismen, hingegen, platzieren Arbeitslasten in hierarchischer Weise auf Rechenressourcen.

  Die Lastintensitätsmodelle können in weniger als 0,2 Sekunden extrahiert werden, wobei die jeweils resultierenden Modelle einen durchschnittlichen Medianmodellierungsfehler von 12,7% aufweisen. Zusätzlich dazu kann die neue Lastverteilungsstrategie auf einzelnen Servern zu Stromverbrauchseinsparungen von bis zu 10,7% führen.

- Ein Ansatz um kleine Arbeitslasten zu erzeugen, welche das Stromverbrauchsverhalten von größeren, komplexeren Lasten emulieren, indem sie ihre CPU Performance Counter-Profile approximieren sowie den TeaStore: Eine verteilte, auf dem Micro-Service-Paradigma basierende Referenzapplikation. Der TeaStore kann verwendet werden, um Strom- und Performanzmodellgenauigkeit, Elastizität von Cloud Autoscalern und die Effektivität von Stromsparmechanismen in verteilten Systemen zu untersuchen.

  Das Arbeitslasterstellungsverfahren kann das Stromverbrauchsverhalten von realistischen Lasten mit einer mittleren Abweichung von weniger als 10% und bis zu einem minimalen Fehler von 0,2 Watt (1%) nachahmen. Die Anwendung des TeaStores wird durch die Extraktion von Performanzmodellen, die Anwendung in einer automatisch skalierenden Cloudumgebung und durch eine Demonstration der verschiedenen möglichen Stromverbräuche, die er auf Servern verursachen kann, gezeigt.

- Eine Methode zur automatisierten Auswahl von Interpolationsstrategien im Bezug auf Performanz und Stromverbrauchscharakterisierung. Diese Methode wird durch einen Konfigurationsansatz, der die Genauigkeit der auslastungsabhängigen Stromvorhersagen von polynomiellen Interpolationsfunktionen verbessert, erweitert.

  Im Gegensatz zur Regression kann der automatisierte Interpolationsmethodenauswahl- und Konfigurationsansatz die Modellierungsgenauigkeit mit Hilfe eines Referenzdatensatzes um 43,6% verbessern und kann selbst ohne diesen Referenzdatensatz eine Verbesserung von 31,4% erreichen.

- Einen Ansatz, der explizit den Einfluss von Virtualisierungsumgebungen auf den Stromverbrauch modelliert und eine Methode zur Vorhersage des Stromverbrauches von Softwareapplikationen. Beide Verfahren nutzen die von der in dieser Arbeit vorgegestellten Stromverbrauchsmessmethologie erzeugten Ergebnisse, um den jeweiligen Stromverbrauch von Servern, die den Vorhersagenden sonst nicht zur Verfügung stehen, zu ermöglichen.

  Die vorgestellten Verfahren können den Stromverbrauch für verschiedene Hypervisorkonfigurationen und für Applikationslasten zuverlässig vorhersagen. Die Vorhersage des Stromverbrauchs von Serverapplikationen erreicht einen mittleren absoluten Prozentfehler von 9,5%.

- Ein Modellierungsansatz zur Stromverbrauchsvorhersage für Laufzeitplatzierungsentscheidungen von Softwarekomponenten, welcher auch dazu verwendet werden kann den Stromverbrauch für bisher nicht beobachtete Lastintensitäten auf dem laufenden System vorherzusagen.

  Der Modellierungsansatz kann den Stromverbrauch von zwei verschiedenen, verteilten Webanwendungen mit einem mittleren absoluten Prozentfehler von 2,2% vorhersagen. Zusätzlich kann er den Stromverbrauch von einem System bei einer in der Vergangenheit nicht beobachteten Lastintensität und Komponentenverteilung mit einem Fehler von 1,2% vorhersagen.

Die Beiträge in dieser Arbeit haben sich bereits signifikant auf Wissenschaft und Industrie ausgewirkt. Die präsentierte Energieeffizienzbewertungsmethodologie, inklusive ihrer Metriken, ist von der U.S. EPA in die neueste Version des ENERGY STAR Computer Server-Programms aufgenommen worden und wird zurzeit außerdem von weiteren Behörden, darunter die EU Kommission und die Nationale Chinesische Standardisierungsbehörde, in Erwägung gezogen. Zusätzlich haben die Implementierung der Methodologie und die zugrundeliegende Methodologie bereits Anwendung in mehreren wissenschaftlichen Arbeiten gefunden.

In Zukunft werden im Rahmen von weiterführenden Arbeiten neue Arbeitslasten erstellt werden müssen, um die Energieeffizienz von spezialisierter Hardware zu untersuchen. Zurzeit verändert sich die Server-Rechenlandschaft in der Hinsicht, dass spezialisierte Ausführungseinheiten, wie Chips zum maschinellen Lernen, GPGPU Rechenchips und FPGAs in Servern verbaut werden. Um sicherzustellen, dass die Messmethodologie aus dieser Arbeit weiterhin relevant bleibt, wird es nötig sein, Arbeitslasten zu erstellen, welche diese Fälle abdecken, sowie Stromverbrauchsmodelle zu entwerfen, die in der Lage sind, derartige spezialisierte Hardware zu betrachten.

Contents

Publications

Journals

Kistowski, J. von, N. Herbst, S. Kounev, H. Groenda, C. Stier, and S. Lehrig (2017a). “Modeling and Extracting Load Intensity Profiles”. In: *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 11.4, 23:1–23:28.

Peer-Reviewed International Conference Contributions

Full Papers

Eismann, S., J. Grohmann, J. Walter, J. von Kistowski, and S. Kounev (2019). “Integrating Statistical Response Time Models in Architectural Performance Models”. In: *2019 IEEE International Conference on Software Architecture (ICSA)*. Full paper acceptance Rate: 21.9%. Hamburg, Germany.

Kistowski, J. von, J. Grohmann, N. Schmitt, and S. Kounev (2019a). “Predicting Server Power Consumption from Standard Rating Results”. In: *Proceedings of the 19th ACM/SPEC International Conference on Performance Engineering (ICPE 2019)*. Full paper acceptance rate: 18.6%. Mumbai, India: ACM.

Kistowski, J. von, J. Pais, T. Wahl, K.-D. Lange, H. Block, J. Beckett, and S. Kounev (2019b). “Measuring the Energy Efficiency of Transactional Loads on GPGPU”. In: *Proceedings of the 19th ACM/SPEC International Conference on Performance Engineering (ICPE 2019)*. Mumbai, India: ACM.

Eismann, S., J. Walter, J. von Kistowski, and S. Kounev (2018). “Modeling of Parametric Dependencies for Performance Prediction of Component-based Software Systems at Run-time”. In: *2018 IEEE International Conference on Software Architecture (ICSA)*. Full paper acceptance Rate: 25.6%.

Kistowski, J. von, S. Eismann, N. Schmitt, A. Bauer, J. Grohmann, and S. Kounev (2018b). “TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research”. In: *Proceedings of the 26th IEEE International Symposium on the Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2018)*. Milwaukee, WI, USA.

Kistowski, J. von, H. Block, J. Beckett, C. Spradling, K.-D. Lange, and S. Kounev (2016a). "Variations in CPU Power Consumption". In: *Proceedings of the 7th ACM/SPEC International Conference on Performance Engineering (ICPE 2016)*. Delft, the Netherlands: ACM.

Kistowski, J. von, J. Beckett, K.-D. Lange, H. Block, J. A. Arnold, and S. Kounev (2015b). "Energy Efficiency of Hierarchical Server Load Distribution Strategies". In: *Proceedings of the IEEE 23nd International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2015)*. Full paper acceptance rate: 19%. Atlanta, GA, USA: IEEE.

Kistowski, J. von, H. Block, J. Beckett, K.-D. Lange, J. A. Arnold, and S. Kounev (2015c). In: *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE 2015)*. Acceptance rate: 27%. Austin, TX, USA: ACM.

Kistowski, J. von, N. R. Herbst, D. Zoller, S. Kounev, and A. Hotho (2015d). "Modeling and Extracting Load Intensity Profiles". In: *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2015)*. Acceptance rate: 29%. Firenze, Italy.

Short Papers

Kistowski, J. von, M. Deffner, and S. Kounev (2018a). "Run-time Prediction of Power Consumption for Component Deployments". In: *Proceedings of the 15th IEEE International Conference on Autonomic Computing (ICAC 2018)*. Trento, Italy.

Schmitt, N., J. von Kistowski, and S. Kounev (2017a). "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 2017)*. Banff, Canada.

Kistowski, J. von and S. Kounev (2015). "Univariate Interpolation-based Modeling of Power and Performance". In: *Proceedings of the 9th EAI International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2015)*. Berlin, Germany.

Work-In-Progress/Vision Papers

Schmitt, N., J. von Kistowski, and S. Kounev (2017b). "Predicting Power Consumption of High-Memory-Bandwidth Workloads". (Work-In-Progress Pa-

per). In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering (ICPE 2017)*. L'Aquila, Italy: ACM, pp. 353–356.
– (2017c). "Towards a Scalability and Energy Efficiency Benchmark for VNF". (Vision Paper). In: *Proceedings of the 9th TPC Technology Conference on Performance Engineering and Benchmarking (TPCTC 2017)*. Munich, Germany.

Tutorial/Tool Papers

Bucek, J., K.-D. Lange, and J. von Kistowski (2018). "SPEC CPU2017: Next-Generation Compute Benchmark". (Poster Paper). In: *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE 2018)*. Berlin, Germany: ACM, pp. 41–42.

Kistowski, J. von, K.-D. Lange, J. A. Arnold, S. Sharma, J. Pais, and H. B. (2018) (2018c). "Measuring and Benchmarking Power Consumption and Energy Efficiency". (Tutorial Paper). In: *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. Berlin, Germany: ACM, pp. 57–65.

Kistowski, J. von, M. Deffner, J. A. Arnold, K.-D. Lange, J. Beckett, and S. Kounev (2017b). "Autopilot: Enabling easy Benchmarking of Workload Energy Efficiency". In: *Proceedings of the 8th ACM/SPEC International Conference on Performance Engineering (ICPE 2017)*. Best Demo Award. L'Aquila, Italy: ACM.

Kistowski, J. von, J. A. Arnold, K. Huppler, K.-D. Lange, J. L. Henning, and P. Cao (2015a). "How to Build a Benchmark". In: *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE 2015)*. Austin, TX, USA: ACM.

Kistowski, J. G. von, N. R. Herbst, and S. Kounev (2014a). "LIMBO: A Tool For Modeling Variable Load Intensities". In: *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*. Dublin, Ireland: ACM.

Peer-Reviewed International Workshop Contributions

Kistowski, J. von, M. Schreck, and S. Kounev (2016b). "Predicting Power Consumption in Virtualized Environments". In: *Computer Performance Engineering: 13th European Workshop, EPEW 2016, Chios, Greece, October 5-7, 2016, Proceedings*. Ed. by D. Fiems, M. Paolieri, and N. A. Platis. Cham: Springer International Publishing, pp. 79–93.

Kistowski, J. G. von, N. R. Herbst, and S. Kounev (2014b). “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.

Kistowski, J. von, N. R. Herbst, and S. Kounev (2014c). “Using and Extending LIMBO for the Descriptive Modeling of Arrival Behaviors”. In: *Proceedings of the Symposium on Software Performance 2014*. Best Poster Award. Stuttgart, Germany: University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, pp. 131–140.

Book Chapters

Lange, K.-D. and J. von Kistowski (2018). “Energy Benchmarking”. In: *Encyclopedia of Big Data Technologies*. Ed. by S. Sakr and A. Zomaya. Cham: Springer International Publishing, pp. 1–6.

Technical Reports

Kistowski, J. von, K.-D. Lange, J. A. Arnold, H. Block, G. Darnell, J. Beckett, and M. Tricker (2017c). *The SERT Metric and the Impact of Server Configuration*. Tech. rep. 7001 Heritage Village Plaza, Suite 225, Gainesville, VA 20155, USA: Standard Performance Evaluation Corporation (SPEC).

Brunnert, A. et al. (2015). *Performance-oriented DevOps: A Research Agenda*. Tech. rep. SPEC-RG-2015-01. SPEC Research Group — DevOps Performance Working Group, Standard Performance Evaluation Corporation (SPEC).

Kuperberg, M., N. R. Herbst, J. G. von Kistowski, and R. Reussner (2011). *Defining and Quantifying Elasticity of Resources in Cloud Computing and Scalable Platforms*. Tech. rep. Am Fasanengarten 5, 76131 Karlsruhe, Germany: Karlsruhe Institute of Technology (KIT).

Tool and Software Contributions

Peer Reviewed Tool Contributions

Jóakim von Kistowski and Simon Eismann and Norbert Schmitt and André Bauer and Johannes Grohmann and Samuel Kounev (2019). *TeaStore*. Standard Performance Evaluation Corporation Research Group (SPEC RG) accepted Tool.

Jóakim von Kistowski and Andreas Weber and Nikolas Herbst (2015). *LIMBO*. Standard Performance Evaluation Corporation Research Group (SPEC RG) accepted Tool. https://research.spec.org/tools/overview/limbo.html.

Contributions to Industry Standard Tools

Standard Performance Evaluation Corporation (SPEC) (2017a). *Chauffeur Worklet Development Kit v2*. http://spec.org/chauffeur-wdk/.

– (2017b). *Server Efficiency Rating Tool (SERT) v2*. http://spec.org/sert2/.

Chapter 1

Introduction

In the introduction to this doctoral thesis, we first motivate the topic of research and set the general context. We then proceed to present the concrete problem statement addressed in this work, before introducing the current state-of-the-art and show that it does not sufficiently address the identified problems. Based on this, we define concrete goals and research questions. Next, we summarize the contributions of this work, which address these goals. We also describe how we evaluate these contributions. Finally, we provide an outline of the thesis.

1.1 Motivation

Energy efficiency of computing systems has become an ever more important issue over the last decades. In addition to mobile and other end-user devices, the energy efficiency of data centers and servers has gained attention. In 2010, the U.S. Environmental Protection Agency (U.S. EPA) estimated that 3% of the entire energy consumption in the U.S. is caused by data center power draw (Lange and Tricker, 2011). According to a New York Times study from 2012, data centers worldwide consume about 30 billion watts. This is equivalent to the approximate output of 30 nuclear power plants (Babcock, 2012). According to The Guardian in 2015 (Vaughan, 2015), they are responsible for 2% of the world's greenhouse gas emissions, which is similar to the amount produced by air travel.

In addition to these environmental concerns, power consumption of servers in data centers results in significant operating costs. These costs are ever increasing. Cecci and Pultz, 2016 estimate that ongoing data center power costs are increasing at least 10% per year. They also estimate the power consumption's share of a data center's total operating costs will increase during the next years. This effect is mostly attributable to higher density of servers within data centers and to the higher power density of many state-of-the-art devices. They conclude that optimizing the IT power consumption within the data center is one of the major cost saving measures.

Coming to a similar conclusion, the U.S. EPA launched its ENERGY STAR Computer Server program (EPA, 2013) with the goal of labeling energy-efficient servers. Servers with a high energy efficiency are to receive the ENERGY STAR label, enabling potential buyers to purchase more efficient devices. To this end, the U.S. EPA has called for measurement methodologies to facilitate the server labeling. The implementation of the methodology presented in this thesis is the result of this call and is now part of the ENERGY STAR Computer Server specification in its 3rd iteration. Similarly, other regulatory bodies, such as the China National Institute of Standardization (CNIS) and the EU Commission, are currently considering adoption of this methodology.

1.2 Problem Statement

The energy efficiency and power consumption of a server are subject to a great number of factors. Firstly, and most obviously, the server's hardware components have a direct effect on power consumption and performance. In addition, power consumption and efficiency depend on the workload executed by the server, the workload's load intensity/load level and its load distribution, i.e., the resources on the server allocated for processing the workload. Finally, the server's software stack, in addition to the application workload, may have an effect on power consumption and/or efficiency. This stack includes operating systems and potential hypervisors and/or language runtimes.

This high number of influencing factors makes finding an energy-efficient server for a specific use-case very difficult. Among others, server provisioners, operators, and regulators would profit from sufficient and structured information on the servers in question and on the factors that affect those servers' power consumption and efficiency. Specifically, it is hard for server provisioners to acquire sufficient information on a server and its interaction with all the mentioned influencing factors in order to make good purchasing decisions. Similarly, an operator must be able to obtain this information for his or her servers in order to be able to deploy software artifacts on those servers on which they consume the least power. Government regulators, in turn, would like to certify low energy or energy-efficient devices, but must do so in a very general way, as they have very little information on the device's intended use.

In general, we see a lack of measurement methods for power consumption and energy efficiency. The concrete steps for performing such measurements and ratings based on those measurements remain undefined: Which workloads to execute for efficiency measurements, how to place and run these workloads, how to configure measurements for different load levels, and where to instrument the system under test? Finally, which metrics to use for comparing the obtained results regarding efficiency and power consumption remains an open question. Note that a fair metric is especially important for government regulators, who need it to define pass/fail criteria for energy efficiency labels and certifications based on measurement results.

Even assuming that a measurement methodology were to exist, making decisions based on its results would still be a non-trivial process. Ideally, a server's potential buyer would like to know the power consumption of a target application for the server. Even if standardized measurement results are provided, the consumption of the specific target application would remain unknown. A server operator, on the other hand, may have the opportunity to obtain measurements at run-time. Yet, predicting the effects potential reconfigurations have on a running server environment is still challenging.

In summary, the problem addressed by this thesis is twofold: Firstly, no comprehensive measurement and rating methodology for server energy efficiency exists, covering the range of influencing factors. In addition, there is no metric that would help to compare the results of such a methodology. Secondly, we see a need for power prediction mechanisms to enable intelligent decision making based on the results provided by such a methodology.

1.3 State-of-the-Art

Existing measurement and rating methods for energy efficiency exist in the form of energy efficiency benchmarks. Server energy efficiency benchmarks run a stable load for a period of time and then compute energy efficiency as a function of the benchmark's performance and power consumption measured during the benchmark run. Energy efficiency benchmarks, such as JouleSort (Rivoire et al., 2007b), measure the energy efficiency of a single workload, potentially run with several different settings. Yet they always run at maximum speed, not accounting for the different load levels a server would be subject to in production use. As a first step towards this work, SPECpower_ssj2008 (Lange, 2009) addresses this by executing its workload at ten different load levels. However, no existing energy efficiency benchmark or methodology takes multiple workloads with different load levels into account. The benchmarks are all designed to demonstrate efficiency for a single use case, instead of the broad application scenario described in our problem statement.

Efficiency metrics used for comparison are limited to variations of the *performance per power* ratio, as used by Rivoire et al., 2007b and Metri et al., 2012 (or in reverse by Poess et al., 2010). This metric has the major drawback of only being applicable to a single load level and workload. In return, power proportionality metrics, such as those of Schall et al., 2012 and Hsu and Poole, 2015, only display the range of power consumption and are not intended for energy efficiency comparisons.

Making informed decisions based on rating results requires prediction of the decisions' effects. In our case, this means prediction of power consumption for unavailable systems (e.g., when considering a system for purchase) or prediction of power consumption effects of changes on running systems. Offline power prediction models for not-yet-available systems do exist. However, these models, such as those of Brooks et al., 2000 and Kahng et al., 2009 are intended for hardware designers and require a highly detailed level of knowledge on the hardware components and their electrical properties. Other offline prediction models, such as those by Basmadjian et al., 2011 and Stier et al., 2015, are software architecture models intended for software designers comparing different potential design decisions. They do not make use of measurement results provided by benchmarks or rating methodologies. Online power prediction models, on the other hand, are included in power management mechanisms, such as those of Beloglazov et al., 2012 and Urgaonkar et al., 2010. They only apply very generic utilization based power models, which generally feature low accuracy (Rivoire et al., 2008).

Summarizing, no comprehensive energy efficiency rating method for multiple workloads and load levels exists. Consequently, metrics and prediction methods based on such a rating methodology are also needed.

1.4 Goals and Research Questions

Based on our problem statement and the drawbacks of the current state-of-the-art, we formulate two major overarching goals. The contributions in the two following parts of this thesis address these goals. Part II primarily addresses the first goal, whereas Part III addresses the second. To specify the goals in further detail, we pose specific research questions to be considered for each of the goals. We pose a total of eight research questions (RQs), five of which are related to the first goal and three of which are related to the second goal.

Goal A: Create a comprehensive power and energy efficiency measurement and rating methodology for servers.
The methodology should enable rating of a broad range of servers and should be applicable for users in different contexts, including regulators, system designers, and potential buyers.

This major goal can be split into several research questions that have to be answered in order to arrive at a comprehensive methodology. We formulate five questions. The first two questions must be answered in order to create the base methodology, whereas the additional three expand on those first two questions.

RQ A.1: How to place, execute, and measure workloads in a reproducible and representative manner for energy efficiency rating?

RQ A.2: How to aggregate results of different multi-stage energy efficiency tests producing a fair energy efficiency metric?

RQ A.3: How to model and create realistic, varying load profiles for energy efficiency testing?

RQ A.4: How to heterogeneously distribute load for different placements in energy efficiency testing?

RQ A.5: How to create reference workloads for complex test setups in distributed scenarios?

Goal B: Provide methods for using the results of the measurement methodology for data center provisioners and/or operators.
Data center provisioners and/or operators should be given methods to use the data provided in the measurement results for better decision making by helping to predict the effects of their actions.

We specify this second goal further by splitting it into three research questions. Each of these questions formulates a concrete use-case for measurement results obtained using our rating methodology.

RQ B.1: How to obtain information about load level power consumption for a load level not covered by the measurements made when applying the methodology?

RQ B.2: How to predict the power consumption of a concrete target application or software stack (such as a hypervisor) for a server for which the methodology's results are available?

RQ B.3: How to use power measurements or pre-measured results at run-time to predict the power consumption of software component placements?

1.5 Contribution and Evaluation Summary

This thesis contains six major contributions, some of which consist of two sub-contributions. These contributions address our research questions and problem statement. They can be split into two parts, each part focusing on one of our major goals.

Contribution 1: **Methodology for Server Efficiency Rating**

This contribution addresses `Goal A` and and research questions `RQ A.1` and `RQ A.2`. The contribution contains a measurement methodology and metrics for energy efficiency rating of servers that use multiple, specifically chosen workloads at different load levels for a full system characterization. As part of the contribution, we show how metrics are aggregated over the load levels and workloads in order to arrive at a final energy efficiency score that can, among other things, be used by regulators.

We evaluate the methodology and metric with regard to their reproducibility, fairness, and relevance. We investigate the power and performance variations of test results and show fairness of the metric through mathematical proof and a correlation analysis on a set of 385 servers. In addition, we evaluate relevance by showing the relationships that can be established between metric results and third-party applications. We show that our methodology helps decision makers and regulators make reliable decisions on the energy efficiency of servers.

The methodology has been implemented as part of the Server Efficiency Rating Tool (SERT) released by the Standard Performance Evaluation Corporation (SPEC), which was developed at the request of the U.S. EPA for use in the Energy Star program for servers. A test using the SERT 2.0, which implements many of the features introduced in this thesis, including our proposed metric, is required for acceptance in the Energy Star program. Many parts and analyses of the methodology have been published in the Proceedings of the ICPE 2015, ICPE 2016, and ICPE 2018 conferences (Kistowski et al., 2015c, Kistowski et al., 2016a, and Kistowski et al., 2018c).

Contribution 2: **Load Profiles for Energy Efficiency Measurement**

Expanding on the first contribution, we address research questions `RQ A.3` and `RQ A.4` using two sub-contributions. We create models and model extraction mechanisms for load profiles that vary over time. The models are designed to be used to define arbitrary dynamic load intensity profiles that can be leveraged for benchmarking purposes to evaluate the behavior of a system under different dynamic workload scenarios. In addition to the load profile models, we describe a modification to our methodology for evaluation of hierarchical load distribution. We also introduce load distribution policies that place workloads on computing resources in a hierarchical manner (i.e., first servers, then CPU sockets, then CPU cores).

We evaluate the load intensity models and demonstrate their ability to capture realistic load intensity profiles by automatically extracting load intensity models from a representative set of nine different real-world traces. Each extraction concludes in less than 0.2 seconds and our resulting models feature a median modeling error of 12.7% on average. An implementation of the models and extraction methods is available as part of the LIMBO toolkit, which has been reviewed and endorsed by the Standard Performance Evaluation Corporation, Research Group (SPEC RG). In addition to the load profile modeling evaluation, we evaluate the load distribution strategies on a range of systems and test a variety of combinations applying different load distribution strategies on the different levels of the execution hierarchy. We also show that our new load distribution strategy can save up to 10.7% of power consumption on a single server node. Publications regarding this contribution has been published at ACM TaaS (Kistowski et al., 2017a), SEAMS 2015 (Kistowski et al., 2015d), and MASCOTS 2015 (Kistowski et al., 2015b).

Contribution 3: **Workloads for Energy Efficiency Measurement**
We address `RQ A.5` in our third contribution by presenting workload creation methods and workloads. Firstly, we introduce an approach to create small-scale workloads that emulate the power consumption-relevant behavior of large scale workloads by approximating their CPU performance counter profile. We construct a Performance Event Trigger Framework (PET), designed to use performance counter measurements of third-party applications to construct a small-scale workload that emulates the third-party application's resource profile with the goal of reproducing its power consumption-relevant behavior. We also introduce TeaStore, a test and reference application for distributed systems. TeaStore can be used to evaluate power and performance model accuracy, elasticity of cloud auto-scalers, and the effectiveness of power saving mechanisms for distributed systems.

We validate PET by applying it to four different applications, ranging from small-scale test applications over standard benchmarks to a virtual network function. We show that PET is capable of emulating the power consumption behavior of realistic workloads with a mean deviation less than 10% and down to 0.19 W (1%). Additionally, we demonstrate the use of TeaStore for three target use cases. We extract performance models, demonstrate TeaStore's use in an auto-scaling environment and show that it offers placement options with different power consumption, energy efficiency, and performance optima. These contributions have been published at the MASCOTS 2017 and MASCOTS 2018 conferences (Schmitt et al., 2017a and Kistowski et al., 2018b).

Contribution 4: **Power Consumption Interpolation Methods**
Moving on to `Goal B`, we tackle research question `RQ B.1` by presenting a method for automated selection of interpolation strategies for performance and power characterization. We also introduce a configuration approach for polynomial interpolation functions of varying degrees that improves prediction accuracy for system power consumption for a given system utilization.

We evaluate our interpolation method's ability to predict the power consumption of an unmeasured load level. We show that interpolation features superior accuracy compared to regression in this domain. In comparison to regression, our automated interpolation method configuration and selection approach improves modeling accuracy by 43.6% if additional reference data is available and by 31.4% if it is not. This contribution has been published at the VALUETOOLS 2015 conference (Kistowski and Kounev, 2015).

Contribution 5: **Methods for Offline Prediction of Power Consumption**
Addressing `RQ B.2`, we introduce two offline power prediction mechanisms: Firstly, we present an approach for explicit modeling of the impact a virtualized environment has on power consumption. Secondly, we introduce a method to predict the power consumption of a software application. Both methods use results produced by the SERT implementation of our measurement methodology to predict the respective power consumption for servers that are otherwise not available to the person making the prediction.

We evaluate both methods by comparing power predictions against real-world systems. We predict the power consumption of multiple configurations of the Xen hypervisor and compare the predicted power consumption of the virtualized environment against our measurements. We also predict the power consumption of three different workloads on a target server. Our methods are able to predict power consumption reliably for multiple hypervisor configurations and for the target application workloads. Application workload power prediction features a mean average absolute error of 9.5%. This contribution have been published at EPEW 2015 (Kistowski et al., 2016b) and the ICPE 2019 conference (Kistowski et al., 2019a).

Contribution 6: **Method for Online Prediction of Power Consumption**
Finally, we address `RQ B.3` by proposing an end-to-end modeling approach for predicting the power consumption of component placements at run-time. The approach captures the individual components' power profiles and their interactions based on which it can predict the power consumption of a new component placement configuration. The model can also be used to predict the power consumption at load levels that were not yet observed on the running system.

We evaluate our models and prediction mechanism using two different web applications deployed in a heterogeneous environment with servers of different CPU hardware architectures. We predict the power consumption of different component placements and show that we can predict the power consumption of previously unobserved deployments with a mean absolute percentage error of 2.2%. We published this contribution at the ICAC 2018 conference (Kistowski et al., 2018a).

Our contributions can assist regulators in specifying energy efficiency criteria for servers and they help data center provisioners to make more informed decisions when selecting servers. Server operators can apply many of the methods presented here to better provision their servers at different times during a server's life-cycle. Hardware and software developers can use the proposed models and measurement methods for testing and for comparison of design alternatives. Finally, researchers can use the measurement methods of this thesis to evaluate their own work, as demonstrated in many chapters of the thesis itself. In addition, they can use the models and methodologies presented here as part of their own research. This can be demonstrated using the LIMBO framework and its corresponding load profile models, which have already been adopted and used by several follow-up works in other application domains, for example, by Herbst et al., 2015, Groenda and Stier, 2015, and Becker et al., 2017a.

1.6 Thesis Outline

This thesis is structured into four parts, excluding this introductory chapter (Chapter 1). Part I covers the foundations needed for understanding the thesis in Chapter 2 and describes the current state-of-the-art in Chapter 3.

Part II contains our contributions regarding measurement and rating of server energy efficiency and power. This part contains the contributions towards `Goal A`, which includes research questions `RQ A.1` through `RQ A.5`. We present our efficiency rating methodology in Chapter 4, our load intensity profile and load distribution models and methods in Chapter 5, and our workloads and workload creation methods in Chapter 6.

Next, Part III details our contributions regarding modeling and prediction of server power consumption, addressing `Goal B`, including research questions `RQ B.1` through `RQ B.3`. We present our power interpolation approach in Chapter 7, our offline power prediction methods in Chapter 8 and our online prediction method in Chapter 9.

Part IV evaluates all of our contributions. Firstly, we evaluate the contributions of Part II. We evaluate the measurement methodology regarding relevance, reproducibility, and fairness in Chapter 10, the load profile models and load distribution policies in Chapter 11 and our workloads in Chapter 12. We then evaluate the contributions of Part III. Chapter 13 investigates the accuracy of our interpolation method, Chapter 14 compares the predictions of our offline prediction methods against real-world measurements, and Chapter 15 applies our online prediction model using real-world systems and software applications.

Finally, we conclude the thesis and provide an outlook in Chapter 16.

Disclaimer

This thesis contains measurement results obtained with the following SPEC benchmarks and tools: SPECpower_ssj2008, SPEC SERT in versions 1 and 2, and SPEC ChauffeurWDK. Unless noted otherwise, these results must be considered non-compliant and have been obtained according to the SPEC fair-use policy for academic use (see Standard Performance Evaluation Corporation, 2018 for further details).

Part I

Foundations and Related Work

Chapter 2

Foundations

This chapter lays the foundation required for understanding the contributions of the thesis. We assume that the reader is familiar with basic measures of electricity, specifically with Energy, measured in Joules $[J]$ or Watthours $[Wh]$ and electrical power, measured in Watts $[W]$.

In this chapter, we first introduce the concept of *transactionality* that is used in the following work and discuss the established *performance metrics*. Next, we discuss *load profiles*, introducing open and closed workloads, including a definition of *load intensity*, and discussing basics of *time series* needed for our work on load profile changes over time. We then define the term *benchmark* and transition to discussing the criteria that a benchmark must meet in order to be considered a quality benchmark. These criteria are driving factors behind our design decisions regarding our power methodology and are being used to evaluate the quality of this methodology in Chapter 12. Of course, this methodology itself serves a foundation to the work of the following Chapters. We also discuss CPU Performance Counters as foundations to this work. They are used in many of our approaches and as part of several evaluations.

2.1 Transactions and Server Performance Metrics

All contributions in this thesis are designed for and around servers in a standard data center environment. The rating methodology rates the efficiency of these servers, the workloads are to be executed on them, and the prediction methods predict their power consumption. For all these use cases, we consider servers that serve a continuous stream of requests with a focus on the rate at which those requests are processed. Instead of either serving a request or not (or rather, serving the maximum rate of requests or running idle), the continuous stream of requests makes the servers under consideration appear to run at a relatively steady low load (meaning a low request rate below the server's maximum potential). This is in contrast to servers used in batch processing or High Performance Computing (HPC) environments, which tend to run at a high load level for long periods of time solving a single, highly complex problem.

We refer to the work units served by such a server as “transactions”. In this context, a transaction is any work unit that begins when a unit of work starts computation after being triggered by an external request and ends when a response is sent. In the case of performance benchmarking, no external user requiring a response exists. Consequently, a transaction is considered to be any unit of work for which a definite beginning and end can be specified.

We use two primary performance metrics for our transactional server performance tests:

- **Response Time** (Eq. 2.1)

  $$responsetime = t_{end} - t_{start} \tag{2.1}$$

  where t_{end} is the point in time at which a transaction concludes computation and t_{start} is the point in time, when it starts.

- **Throughput** (Eq. 2.2)

  $$throughput = \frac{\#transactions}{t} \tag{2.2}$$

  where t is the time during which the measurement was made and $\#transactions$ is the number of transactions that concluded during this time.

  Throughput is directly related to the arrival rate λ of transactions. The arrival rate is the rate of transactions arriving on the system. In theory, for systems that are not over-utilized, i.e., systems where an incoming transaction can begin execution immediately, the arrival rate equals the throughput.

2.2 Load Profiles

Our rating methodology and models use and capture load intensity in the form of user, job, or request arrival rates. Depending on the use-case, the load intensity may be steady or vary over time. For both cases, we employ an **open workload** view. Schroeder et al., 2006 define open workloads as workloads, in which new jobs arrive independently of job completions. Our workloads and workload models are designed around the open workload paradigm. They do not make any assumptions about the completion times of the work units. This decision is based on the assumption that users in a typical cloud environment

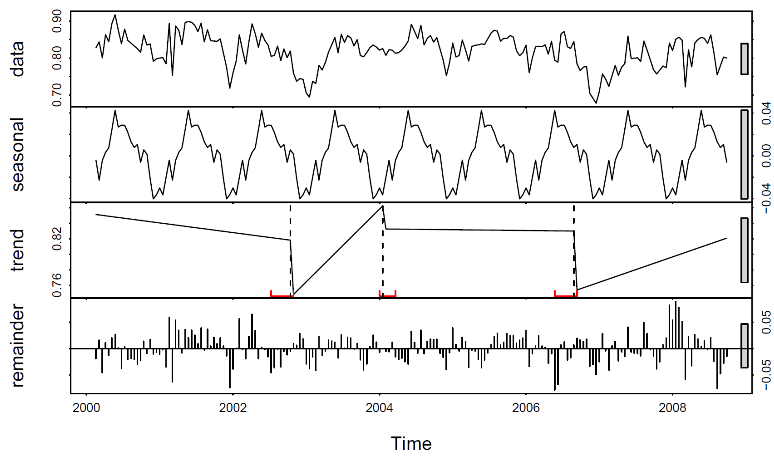


Figure 2.1: The decomposition of a time series into seasonal, trend, and remainder (Verbesselt et al., 2010).

are unaware of one another and access a software service without having any knowledge of other users' behavior.

For this thesis, we define **load intensity** as a function describing *arrival rates* of workload units (e.g., users, sessions, or requests) over time. We define the *arrival rate* $r(t)$ at time t as follows:

$$r(t) = R'(t)$$
$$\text{with } R(t) = |\{u_{t_0} | t_0 \leq t\}|$$

where $R(t)$ is the amount of *work units* u_{t_0} with *arrival time* t_0 that have arrived up until time t. R' is the derivative thereof.

In this thesis, we consider arrival rates of work units with the goal of creating or analyzing workloads. For varying arrival rates, analysis can be helped by decomposing the time series of the varying rates into its different parts. Time series decomposition is an established field of research and its primary concepts are used as a foundation of this work. Specifically, we use the approach of Breaks for Additive Season and Trends (BFAST) (Verbesselt et al., 2010) as a foundation.

BFAST decomposes time series into the following functions:

- **Seasonal**: an infinitely repeating function of the time series' seasonal characteristics (such as sin or cos functions)
- **Trend**: an additive trend that shows the change of the seasonal pattern over time

- **Remainder**: the remainder, which is assumed to consist of random noise

Figure 2.1 shows an example decomposition of a time series into its constituent parts. This sort of decomposition can be relevant for workload modeling and can be used to detect recurring patterns, trends, etc.

2.3 Benchmarks

This thesis introduces an energy efficiency rating methodology, which shares many aspects with energy efficiency benchmarks. In this section, we provide a definition of the term "benchmark" in the context of performance evaluation. Note that we differentiate between benchmarks with the purpose of product comparison and rating tools, which are intended for standardized measurements as part of a product development or evaluation process. We explain the differences between the three types of benchmarks: specification-based, kit-based, and hybrid. We also present the properties and criteria that any quality benchmark or rating tool must fulfill. For this, we focus on the properties that workloads of quality benchmarks must meet.

2.3.1 Definition of Benchmark

We define a benchmark as a "Standard tool for the competitive evaluation and comparison of competing systems or components according to specific characteristics, such as performance, dependability, or security".

This definition is a variation of a definition provided by Vieira et al., 2012 with a focus on the competitive aspects of benchmarks, as that is the primary purpose of standardized benchmarks as developed, for example, by the Standard Performance Evaluation Corporation (SPEC) and the Transaction Processing Performance Council (TPC).

In contrast, we define tools for the non-competitive system evaluation and comparison as *rating tools*. Rating tools are primarily intended for a standardized method of evaluation for research purposes, regulatory programs, or as part of a system improvement and development approach. Rating tools can also be standardized and should generally follow the same design and quality criteria as benchmarks. For example, the methodology proposed in this work has been implemented in SPEC's Server Efficiency Rating Tool (SERT), but follows many of the design guidelines for benchmarks.

2.3.2 Types of Benchmarks

Computer benchmarks typically fall into three general categories: specification-based, kit-based, and a hybrid between the two. *Specification-based benchmarks* describe functions that must be achieved, required input parameters and expected outcomes. The implementation to achieve the specification is left to the individual running the benchmark. *Kit-based benchmarks* provide the implementation as a required part of official benchmark execution. Any functional differences between products that are allowed to be used for the benchmark must be resolved ahead of time and the individual running the benchmark is typically not allowed to alter the execution path of the benchmark.

Specification-based benchmarks begin with a definition of a business problem to be simulated by the benchmark. The key criteria for this definition are the benchmark quality aspects discussed in Section 2.4. Specification-based benchmarks have the advantage of allowing innovative software to address the business problem of the benchmark by proving that the new implementation (Huppler and Johnson, 2014) complies with the specified requirements. On the other hand, specification-based benchmarks require substantial development prior to running the benchmark, and may have challenges proving that all the requirements of the benchmark are met.

Kit-based benchmarks may appear to restrict some innovative approaches to a business problem, but have the significant advantages of providing near "load and go" implementations that greatly reduce the cost and time required to run the benchmarks. For kit-based benchmarks, the "specification" is used as a design guide for the creation of the kit. For specification-based benchmarks, the specification is presented as a set of rules to be followed by a third party who will implement and run the benchmark. This allows for substantial flexibility in how the benchmark's business problem will be resolved - a principal advantage for specification-based benchmarks.

A hybrid of these may be necessary if the majority of the benchmark can be provided in a kit, but there is a desire to allow some functions to be implemented at the discretion of the individual running the benchmark.

2.4 Benchmark Quality Criteria

Benchmark designers must balance several, often conflicting, criteria in order to be successful. Several factors must be taken into consideration, and trade-offs between various design choices will influence the strengths and weaknesses of the benchmark and its workload. Since no single workload can be strong in all of these areas, there will always be a need for multiple workloads and benchmarks (Skadron et al., 2003).

It is important to understand the characteristics of a benchmark and workload to determine whether or not it is applicable for a particular situation. When developing a new benchmark, the goals should be defined so that choices between competing design criteria can be made in accordance with those goals to achieve the desired balance. Several researchers and industry participants have listed various desirable characteristics of benchmarks: Huppler, 2009; García-Castro and Gómez-Pérez, 2006; Skadron et al., 2003; Stefani et al., 2003; Henning, 2000; Gustafson and Snell, 1995 and Sim et al., 2003. The contents of the lists vary based on the perspective of the author and their choice of terminology and grouping of characteristics, but most of the concepts are similar. The key characteristics can be organized in the following groups, which will be discussed in more detail in the next sections:

- **Relevance**: How closely the benchmark behavior correlates to behaviors that are of interest to consumers of the results
- **Reproducibility**: The ability to consistently produce similar results when the benchmark is run with the same test configuration
- **Fairness**: Allowing different test configurations to compete on their merits with-out artificial limitations
- **Verifiability**: Providing confidence that a benchmark result is accurate
- **Usability**: Avoiding roadblocks for users to run the benchmark in their test environments

All benchmarks are subject to these same criteria, but each category includes additional issues that are specific to the individual benchmark, depending on the benchmark's goals.

2.4.1 Relevance

"Relevance" is a highly important characteristic of a benchmark. Even if the workload was perfect in every other regard, it will be of minimal use if it doesn't provide relevant information to its consumers. Yet relevance is also a characteristic of how the benchmark results are applied; benchmarks may be highly relevant for some scenarios and of minimal relevance for others. For the consumer of benchmark results, an assessment of a benchmark's relevance must be made in context of the planned use of those results. For the benchmark designer, relevance means determining the intended use of the benchmark and then designing the benchmark to be relevant for those areas. A general assessment of the relevance of a benchmark or workload involves two dimensions: the breadth of its applicability, and the degree to which the workload is relevant in that area. For example, an Extensible Markup Language (XML) parsing benchmark may be highly relevant as a measure of XML parsing performance, somewhat relevant as a measure of enterprise server application performance, and not at all relevant for graphics performance of 3D games. Conversely, a suite of CPU benchmarks such as SPEC CPU2006 may be moderately relevant for a wide range of computing environments. Generalizing these examples: benchmarks that are designed to be highly relevant in a specific area tend to have narrow applicability, while benchmarks that attempt to be applicable to a broader spectrum of uses tend to be less meaningful for any particular scenario (Huppler, 2009).

In this work, we consider scalability as an important aspect of relevance, particularly for server benchmarks. Most relevant benchmarks are multi-process and/or multi-threaded in order to be able to take advantage of the full resources of the server (Skadron et al., 2003). Achieving scalability in any application is difficult; for a benchmark, the challenges are often even greater because the benchmark is expected to run on a wide variety of systems with significant differences in available resources. Benchmark designers must also strike a careful balance between avoiding artificial limits to scaling and behaving like real applications (which often have scalability issues of their own).

2.4.2 Reproducibility

Reproducibility is the capability of the benchmark to produce the same results consistently for a particular test environment. It includes both run-to-run consistency and the ability for another tester to independently reproduce the results on another system. In practice, reproducibility can be measured using statistical variability measures accross benchmark results.

Ideally, a benchmark result is a function of the hardware and software configuration, so that the benchmark is a measure of the performance of that environment; if this were the case, the benchmark would have perfect consistency. In reality, the complexity inherent in a modern computer system introduces significant variability in the performance of an application. This variability is introduced by several factors, including things such as the timing of thread scheduling, dynamic compilation, physical disk layout, network contention, and user interaction with the system during the run (Huppler, 2009). Energy efficiency benchmarks often have additional sources of variability due to power management technologies dynamically making changes to system performance and temperature changes affecting power consumption.

Benchmarks can address this run-to-run variability by running for long enough periods of time to include representative samples of these variable behaviors. Some benchmarks require submission of multiple runs with scores that are near each other as evidence of consistency. Benchmarks also tend to run at steady state, unlike more typical applications which have variations in load due to factors such as the usage patterns of users.

The ability to reproduce results in another test environment is largely tied to the ability to build an equivalent environment. Industry standard benchmarks require results submissions to include a description of the test environment, typically including both hardware and software components as well as configuration options. Similarly, published research that includes benchmark results generally includes a description of the test environment that produced those results. However, in both of these cases, the description may not provide enough detail for an independent tester to be able to assemble an equivalent environment.

Hardware must be described in sufficient detail for another person to obtain identical hardware. Software versions must be stated so that it is possible to use the same versions when reproducing the result. Tuning and configuration options must be documented for firmware, operating system, and application software so that the same options can be used when re-running the test. TPC benchmarks require a certified auditor to audit results and ensure compliance with reporting requirements. SPEC uses a combination of automatic validation and committee review to establish compliance.

2.4.3 Fairness

Fairness ensures that systems can compete on their merits without artificial constraints. Because benchmarks always have some degree of artificiality, it is often necessary to place some constraints on test environments in order to avoid unrealistic configurations that take advantage of the simplistic nature of the benchmark.

Benchmark development requires compromises among multiple design goals; benchmarks developed by a consensus of experts are generally perceived as being more fair than a benchmark designed by a single company (García-Castro and Gómez-Pérez, 2006). While "design by committee" may not be the most efficient way to develop an application, it does require that compromises are made in such a way that multiple interested parties are able to agree that the final benchmark is fair. As a result, benchmarks produced by organizations such as SPEC and the TPC (both of which are comprised by members from companies in the industry as well as academic institutions and other interested parties) are generally regarded as fair measures of performance.

Benchmarks require a variety of hardware and software components to provide an environment suitable for running the benchmark. It is often necessary to place restrictions on what components may be used. Careful attention must be placed on these restrictions to ensure that the benchmark remains fair. Some restrictions must be made for technical reasons. For example, a benchmark implemented in Java requires a Java Virtual Machine (JVM) and an operating system and hardware that supports it. A benchmark that performs heavy disk IO may effectively require a certain number of disks to achieve acceptable IO rates, which would therefore limit the benchmark to hardware capable of supporting that number of disks.

Benchmark run rules often require hardware and software to meet some level of support or availability. While this restricts what components may be used, it is actually intended to promote fairness. Because benchmarks are by nature simplified applications, it is often possible to use simplified software to run them; this software may be quite fast because it lacks features that may be required by real applications. For example, enterprise servers typically require certain security features in their software which may not be directly exercised by benchmark applications; software that omitted these features may run faster than software that includes them, but this simplified software may not be usable for the customer base that the benchmark is targeted to. Rules regarding software support can be a particular challenge when using open source software, which is often supported primarily by the developer community rather than commercial support mechanisms.

These situations require a careful balance. Placing too many or inappropriate limits on the configuration may disallow results that are relevant to some legitimate situations. Placing too few restrictions can pollute the pool of published results and, in some cases, reduce the number of relevant results because vendors can't compete with the "inappropriate" submissions. Portability is an important aspect of fairness. Achieving portability with benchmarks written in Java is relatively simple; for C and C++, it can be more difficult (Henning, 2000).

Benchmark run rules often include stipulations on how results may be used. These requirements are intended to promote fairness when results are published and compared, and often include provisions that require certain basic information to be included any time that results are given. For example, SPECpower_ssj2008 requires that if a comparison is made for the power consumption of two systems at the 50% target load level, the performance of each system at the 50% load level as well as the overall ssj_ops/watt value must also be stated. SPEC has perhaps the most comprehensive fair use policy which further illustrates the types of fair use issues that benchmarks should consider when creating run rules (Standard Performance Evaluation Corporation, 2018).

In addition to run rules and portability, fairness is also an important aspect of a benchmark's final metric. The metric must present the benchmark's result in a way that does not artificially disadvantage certain systems or configurations.

2.4.4 Verifiability

Within the industry, benchmarks are typically run by vendors who have a vested interest in the results. In academia, results are subjected to peer review and interesting results will be repeated and built upon by other researchers. In both cases, it is important that benchmark results are verifiable so that the results can be deemed trustworthy.

Good benchmarks perform some amount of self-validation to ensure that the workload is running as expected, and that run rules are being followed. For example, a workload might include configuration options intended to allow researchers to change the behavior of the workload, but standard benchmarks typically limit these options to some set of compliant values which can be verified at runtime. Benchmarks may also perform some functional verification that the output of the test is correct; these tests could detect some cases where optimizations (e.g., experimental compiler options) are producing incorrect results.

Verifiability is simplified when configuration options are controlled by the benchmark, or when these details can be read by the benchmark. In this case, the benchmark can include the details with the results. Configuration details that must be documented by the user are less trustworthy since they could have been entered incorrectly.

One way to improve verifiability is to include more details in the results than are strictly necessary to produce the benchmark's metrics. Inconsistencies in this data could raise questions about the validity of the data. For example, a benchmark with a throughput metric might include response time information in addition to the transaction counts and elapsed time.

Of course, Verifiability ties in with Reproducibility, considering that easily reproducible results are also easy to verify. In addition, verifiability can be ensured using reporting and setup criteria and benchmark result review processes.

2.4.5 Usability

Most users of benchmarks are technically sophisticated, making ease of use less of a concern than it is for more consumer-focused applications. There are, however, several reasons why ease of use is important. One of the most important ease of use features for a benchmark is self-validation. This was already discussed in terms of making the benchmark verifiable. Self-validating workloads give the tester confidence that the workload is running properly.

Another aspect of ease of use is being able to build practical configurations for running the benchmark. For example, the current top TPC-C result has a system under test with over 100 distinct servers, over 700 disk drives and 11,000 SSD ash modules (with a total capacity of 1.76 petabytes), and a system cost of over $30 million USD. Of the 18 non-historical accepted TPC-C results published between January 1, 2010 and August 24, 2013, the median total system cost was $776,627 USD. These configurations aren't economical for most potential users (Huppler, 2009).

Accurate descriptions of the system hardware and software configuration are critical for reproducibility, but can be a challenge due to the complexity of these descriptions. Benchmarks can improve ease of use by providing tools to assist with this process.

2.4.6 Relevance of Benchmark Quality Criteria to this Work

Considering this work and, especially, our energy efficiency rating methodology, we rank relevance, reproducibility, and fairness as the most important characteristics. Verifiability can be partially ensured through reproducible results and is otherwise ensured through a review or regulation process. Usability, on the other hand, is an aspect of concrete benchmark implementations and UI and thus less relevant for the underlying methodology.

2.5 Performance Counters

We measure performance using CPU performance counters in several parts of this thesis. CPU counter measurements require system internal instrumentation, which is usually included in the CPU per default and must be enabled using software. We focus on CPU counters as a means of instrumentation, as the CPU is the hardware component with the highest variability in power consumption (Barroso and Holzle, 2007) and thus the component with the highest potential for power savings. In addition, CPU performance counters can report on DRAM activity (RAM being the component with the second highest power variability).

Hardware manufacturers implement performance counters in most processors to monitor a system's behavior by recording a variety of events, such as cache misses, branch mispredictions, memory accesses, and others. Performance counters can be read using either specialized software, performance monitoring utilities of the operating system, or by directly accessing model specific registers of the CPU. Most counters log events on a system wide basis but some allow reading event subsets on a per socket or per core basis. In general, performance counters can be partitioned into the following two groups(see *AMD64 Architecture Programmer's Manual Volume 2: System Programming* 2016 and *Intel® 64 and IA-32 Architectures Software Developer's Manual* 2016):

- *Occurrence event*, counting how often an event has been observed.
- *Duration event*, counting the accumulated clock ticks during the occurance of an event.

The operating system itself also provides event counters that can be accessed by a user. For example, the Linux `/proc/stat` file gives insight on how many hard- and software interrupts have been processed and the number of context switches performed. Most operating system performance counters can be accessed through the `proc` directory.

Monitoring performance events does have disadvantages (Weaver et al., 2013). Modern implementations of performance counters have inaccuracies due to a tendency to deviate. *Non-determinism*, which manifests as identical workloads resulting in different counter values, and *overcount*, where performance counters were increased multiple times for the same instruction, are the two causes of deviation.

Chapter 3

State-of-the-Art

We structure the state-of-the-art into five parts:

- We examine existing studies on *server* and *cpu power consumption* in Section 3.1. These studies show potential avenues of research, help motivate our work, and inspire some of our methods and evaluation methodologies throughout this thesis.
- Section 3.2 investigates existing *benchmarks* for power consumption and energy efficiency. It also examines existing *distributed test and reference applications* for servers that could be used for energy efficiency tests, especially regarding evaluation of energy efficiency management approaches. Finally, it describes the existing *power and efficiency metrics* that are used in or may be used by energy ratings. The related work of this section is relevant to contributions throughout this thesis, but especially relevant to the rating methodology, introduced in Chapter 4 and the workloads introduced in Chapter 6.
- The state-of-the-art in several additional concerns that relate to energy efficiency measuring and rating is explored in Section 3.2.1. It describes *load profiles and load models*, *load distribution methods and policies*, and additional instrumentation for measurements using *CPU performance counters*. These related works are relevant four our load modeling and distribution methods of Chapter 5.2 and some of the workloads of Chapter 6.
- We investigate existing models related to *offline power prediction* in Section 3.4. These models are relevant, considering that we introduce our own offline power prediction models in Chapter 8.
- Finally, we introduce the state-of-the-art that relates to *online power prediction* in Section 3.5. These models and formalisms expand on the state-of-the-art on offline prediction and are considered regarding our own online power prediction method, which is introduced in Chapter 9.

3.1 Experimental Studies on Server and CPU Power Consumption

A number of studies regarding the energy efficiency and power consumption of servers exist. These studies focus on different aspects, such as CPU power consumption or power consumption of distributed deployments. In the context of our work, they serve as motivation for our energy distribution methodology of Section 5.2 and online power prediction in Chapter 9. They demonstrate the potential for power savings and provide insights into the effects of existing power saving mechanisms. In addition to studies on server power consumption and efficiency, we investigate existing studies of processor power consumption and efficiency. These studies are relevant in the context of this work, as we introduce several processor-heavy workloads (especially in Section 6.1) and use CPU performance counters throughout our evaluation.

3.1.1 Experimental studies of server power management

Several studies of server power management, including the effects of workload consolidation have been conducted in the past: Nathuji and Schwan, 2007 present a study that explores the integration of power management policies, including workload consolidation in virtualized environments using micro-benchmarks. Srikantaiah et al., 2008 explore energy efficiency for workload consolidation on the server level for several workload mixes. Chen et al., 2013 analyze the power consumption of three different task types on a single physical server. Examined impact factors include the number of running Virtual Machines (VMs), task configuration parameters, and four different target load levels. Lefèvre and Orgerie, 2010 examine multiple balancing and unbalancing strategies for virtual machines in cloud environments. Finally, Jin et al., 2012 concentrate on the trade-off between the virtualization overhead and efficiency increase when using VMs to aggregate workloads on fewer physical machines with higher utilization. They also explore the influence of different workloads on efficiency.

To the best of our knowledge, these existing studies do not consider hierarchical compositions of multiple load distribution strategies at a high number of load levels (e.g., server level, CPU socket level, CPU core level). Our introduction of a hierarchical load distribution scheme in Section 5.2 demands such an analysis, which we provide in Section 11.2. We analyze the interdependencies of processor architecture, workload type, load level, and distribution strategy with the goal of achieving optimum energy efficiency for both homogeneous and heterogeneous workloads.

3.1.2 Experimental studies of CPU power management

A number of studies analyzing the power consumption of servers and processors exist. These studies analyze variations in power consumption for individual processors with focus on the major impact factors that can cause a difference in power consumption: Huang et al., 1995 and George et al., 1994 analyze CPU power consumption at the circuit level. They examine individual transistors and their integration with the goal of power characterization and simulation. However, when analyzing power consumption of most commercially available processors, this circuit level power consumption is usually considered as a black-box, as information on processor internals is commonly not available.

Bellosa, 2000 and Russell and Jacome, 1998 analyze power consumption depending on workload with a focus on the executed CPU instructions. They characterize CPU power based on performance counter data. Similarly, Contreras and Martonosi, 2005a build a power model using performance counters. To this end, they use industry standard benchmarks, such as SPEC CPU2006 Henning, 2000 for a thorough and representative analysis.

Processor power management exists at many system levels. All of these may impact power consumption. Gomaa et al., 2004 and Heo et al., 2003 examine the impact of the physical location where a task is executed inside the CPU on power consumption and heat generation. Other management techniques, such as Dynamic Voltage and Frequency Scaling (DVFS) (Schönherr et al., 2010 and Basmadjian and De Meer, 2012), also have significant impact on CPU power consumption and heat. CPU pinning can also affect the power consumption of a workload. Podzimek et al., 2015 analyze the effect of CPU pinning configurations for two virtualized processes on performance degradation through resource contention, they also measure power consumption during their experiments.

Studies of CPU power and power management are relevant to this work for two major reasons: The workloads in this thesis are designed to elicit different levels of power consumption from the tested servers. Some of these workloads (especially in Section 6.1) are CPU-bound and make implicit use of the knowledge gained in these studies. In addition, we use CPU performance counters extensively in our evaluation, considering that the related work has found many correlations between CPU counters and power related behavior.

3.2 Benchmarks, Test Applications, and Metrics

We examine existing *benchmarks and workloads* in two areas: (1) *energy efficiency* and (2) *distributed systems*. Most dedicated energy efficiency benchmarks, such as the ones of Rivoire et al., 2007b or Lange, 2009 are not designed with distributed deployment as a primary use-case. Yet, power management of distributed server systems is a topic addressed in many of our related works in Sections 3.1.1, 3.4, and 3.5, and our contributions in Chapters 6 and 9. As a result, we also examine existing benchmarks and reference applications for modern distributed server deployments, many of which are used in related work on energy-efficient management of servers. Finally, we also examine (3) the *metrics* that energy efficiency benchmarks and data center provisioners use to describe the energy efficiency of their systems.

3.2.1 Energy Efficiency Benchmarks

A great number of energy efficiency benchmarks exist. Many of these benchmarks have been created for end user devices, especially mobile devices, for which battery run-time is an important factor. Due to this reason, energy efficiency tests, such as Lochmann et al., 2017, characterize the different behavior patterns of users on those devices and observe the device states triggered by this behavior.

In contrast, server benchmarks focus less on the user, as end users do not directly use the system in the same manner as they would a mobile device. Common and widely used benchmarks, such as SPEC CPU (Henning, 2000) test the performance of server hardware components with respect to their functionality.

Server energy efficiency benchmarks run a stable load for a period of time and then compute energy efficiency as a function of the benchmark's performance and power consumption measured during the benchmark run. JouleSort (Rivoire et al., 2007b) is one of the earliest benchmarks of this type. It measures the energy efficiency when sorting large amounts of data. It always runs at maximum speed, but does feature different run levels with varying data sizes of the sorted arrays. SPECpower_ssj2008 (Lange, 2009) is a benchmark that emulates a transactional business application. It features ten different load levels, which scale with the application's throughput. It can be seen as the predecessor to this work. Finally, the TPC explicitly allows for energy measurement during the execution of all of their current benchmarks (Poess et al., 2010). These benchmarks are specifically designed for database performance testing and are often executed in highly distributed systems. In addition to the standard load levels that TPC benchmarks use for performance testing, a TPC-based energy efficiency test requires reporting of idle power consumption of the setup under test.

In contrast to the existing benchmarks, the rating methodology in Chapter 4 performs energy efficiency measurements for multiple workloads at multiple load levels. Different workloads may run at different load levels enabling a thorough characterization of the system under test. Also note that it is designed to enable ratings of servers used in production environments. Consequently, we do not require instrumentation of separate hardware modules within the server. This sets the methodology of this work apart from a significant class of related works, which do perform internal instrumentation, such as Economou et al., 2006b.

3.2.2 Distributed Software Workloads and Test Applications

Many distributed software benchmarks and reference application exist. Yet, with the emergence of modern trends like DevOps, the focus of research benchmarks has moved from fixed multi-tier application benchmarks, such as SPECjEnterprise 2010 (Standard Performance Evaluation Corporation (SPEC), 2010), towards more scalable micro-service applications. With this shift in how applications are developed, deployed and maintained, requirements for research benchmarking reference applications have changed. In this section, we examine the existing reference applications and workloads in regards to their ability to meet these new requirements. Section 6.2 introduces TeaStore as a new distributed reference application, based on the micro-service paradigm. In this examination of existing reference applications, we show the need for TeaStore as a new workload that can be used to examine the energy efficiency, performance, and other non-functional aspects of distributed system management.

Micro-services offer many degrees of freedom, such as placing your services in a public or private cloud (Ezhilchelvan and Mitrani, 2016), predictive elastic resource scaling (Gong et al., 2010), and auto-scaling (Chieu et al., 2009). These possible degrees of freedom have led micro-service architectures to be adopted in many areas of application, which, noteworthy in the context of this thesis, includes energy management (Bao et al., 2016). In addition to requiring these degrees of freedom, other requirements become necessary for a reference application to be used in research. Aderaldo et al., 2017 identify 15 requirements for micro-service research benchmarks. Table 3.1 lists these requirements and checks common benchmark applications for compliance. In terms of compliance to these criteria, TeaStore satisfies all criteria except *Alternate Versions* (R11) and *Community Usage & Interest* (R12) and is, in terms of requirements, identical to the Sock Shop (Weaveworks Inc., 2017). R12, in particular, cannot be satisfied by a newly proposed reference application. ACME Air (IBM, 2015), Spring Cloud Demo (Bastani, 2015) and MusicStore (.NET Foundation, 2017), also

Table 3.1: Micro-service benchmark introduced by Aderaldo et al., 2017 and our research benchmark requirements for TeaStore (TS) in comparison to ACME Air (AA), Spring Cloud Demo Apps (SCD), Shocks Shop (SoSh) and Music-Store (MS).

Micro-service Benchmark Requirement		**TS**	**AA**	**SCD**	**SoSh**	**MS**
R1	Explicit Topological View	✓		✓	✓	
R2	Pattern-based Architecture	✓	✓	✓	✓	✓
R3	Easy Access from a Version Control Repository	✓	✓	✓	✓	✓
R4	Support Continuous Integration	✓		✓	✓	
R5	Support for Automated Testing	✓		✓	✓	
R6	Support for Dependency Management	✓	✓	✓	✓	✓
R7	Support for Reusable Container Images	✓	✓	✓	✓	
R8	Support for Automated Deployment	✓			✓	
R9	Support for Container Orchestration	✓	✓		✓	
R10	Independence of Automation Technology	✓			✓	
R11	Alternate Versions		✓			✓
R12	Community Usage & Interest		✓			
Research Benchmark Requirement						
B1	Service must Stress System Under Test	✓	✓	(unknown)		✓
B2	Support for Different Load Behavior in Services	✓	✓	✓		
B3	Support for Different Load Generators	✓	✓	✓	✓	✓
B4	Load Profiles Publicly Available	✓				

compared in Table 3.1, satisfy fewer requirements. These criteria suit micro-service benchmarks, yet they do not cover the ability of an application to be used as a reference research benchmark application in research domains, such as resource management and software analysis and modeling. We therefore extended the requirements by four research benchmark requirements B1-B4 in Table 3.1. To research current performance challenges, an application must put actual load on the System Under Test (B1), without focusing on a single server component like memory or CPU load (B2). The application should also be able to be put under load by different load generators to fit a wide variety of benchmarking environments (B3) and the used load profiles should be public for repeatability (B4). The TeaStore satisfies all these criteria while the next best application, ACME Air, only misses publicly available load profiles.

With the changing requirements and the fast living development of modern web services, the available reference applications cannot keep up and thus cannot cover all requirements for micro-service research benchmarking. The Rice University Bidding System (RUBiS) was first released in 2002. It is an eBay-like bidding platform (*RUBiS User's Manual* 2008) that has been implemented using various technologies including PHP, EJB and Java HTTP servlets. The different available implementations allow for benchmarking of the underlying technologies (Cecchet et al., 2003), the specific implementations themselves (Cecchet et al., 2002), the impact of applying common design patterns (*RUBiS User's Manual* 2008) and the already mentioned predictive elastic resource scaling methods (Gong et al., 2010). RUBiS only has a single application service and an Apache HTTP load balancer (*RUBiS User's Manual* 2008). While RUBiS supports remote procedure calls across multiple hosts, it can only be scaled as a single service. In contrast, each service of TeaStore can be scaled individually. This allows for creating different resource usage characteristics through the deployment of TeaStore's services, consistent with the micro-service paradigm. Similar to RUBiS, the Dell DVD Store (Dell, Inc., 2011), released in 2001, also features multiple implementations. It is a single-service application that implements a simple web store for DVDs. The CloudStore application on the other hand, has a different focus. It is a book store built as a reference application for comparing cloud providers, cloud service architectures, and cloud deployment options. It implements the Transaction Performance Council's TPC-W specification. However, TPC-W is obsolete since 2005. Like RUBiS, CloudStore suffers from inherent scalability bottlenecks. It has been used for elasticity benchmarking, evaluating scalability metrics (see CloudScale Consortium, 2016, Lehrig et al., 2018, and Brataas et al., 2017) in order to test an infrastructure's ability to mitigate these bottlenecks. Just like RUBiS and the DVD Store, the CloudStore application lacks many degrees of freedom regarding deployment and configuration due to their single-service implementations.

A more modern and established benchmark both in research and in the commercial domains is the SPECjEnterprise2010 (Standard Performance Evaluation Corporation (SPEC), 2010) from the Standard Performance Evaluation Corporation (SPEC). SPECjEnterprise2010 implements a three tier web store with separate UI, business logic, and persistence components. It can be used to evaluate resource management techniques in a distributed setting (Willnecker et al., 2015b). Unfortunately, it lacks support for modern micro-service architectures due to its classic three tier architecture. It therefore can also not fulfill the micro-service requirements discussed earlier.

The Palladio Component Model (PCM)-Media Store is designed as a component-based application with components that can be deployed on different systems and therefore closer to a micro-service application than RUBiS, Dell DVD Store, CloudStore and SPECjEnterprise2010. Each component can be replaced to support different architectures. The Media Store is specifically developed for evaluating design-time performance modeling techniques (Happe et al., 2011). Also developed with a focus on design-time performance modeling is the Common Component Modeling Example (CoCoME). It was used in a Dagstuhl Research Seminar to compare different modeling approaches (Rausch et al., 2008). TeaStore, on the other hand, not only supports design-time modeling, but is also available with built-in instrumentation required for benchmarking, run-time model extraction, as well as elasticity and energy efficiency measurements.

Many other benchmark and test applications, like JPetStore (Oracle and Microsystems, 2005), PetClinic (Software, 2016), ACME Air (IBM, 2015), Spring Cloud Microservice Example (Bastani, 2015), Sock Shop (Weaveworks Inc., 2017) and MusicStore (.NET Foundation, 2017) are available. Yet, PetClinic, SCME, Sock Shop and MusicStore are primarily designed as demonstrators for specific technologies and not as research benchmark reference applications. Additionally, modern services such as Sock Shop are built with consistent performance in mind and do not pose the performance challenges that current research aims at.

TeaStore, presented in Section 6.2 offers a modern micro-service architecture for research benchmarks compared to other reference and benchmarking applications. It can also fullfill most micro-service benchmarking requirements shown in Table 3.1 without beeing a technology demonstrator like Sock Shop. TeaStore also offers different performance characteristics for each service and is not limited to a single use case like the PCM Media Store and CoCoME.

3.2.3 Energy Efficiency Metrics

Many metrics for energy efficiency of servers exist. Some, such as the Green Grid metrics PUE and DCiE (Belady et al., 2008) are designed to measure relative efficiency of an entire data center. These metrics usually take data center properties in addition to the server power consumption and/or performance into account. For example, the PUE considers the data center's infrastructure power. Similarly, the SWaP (Rivoire et al., 2007a) metric considers the physical space occupied by a server within a data center.

The primary energy efficiency metric of all of the energy efficiency benchmarks in Section 3.2.1 is a variation of performance per power. JouleSort (Rivoire et al., 2007b) divides the amount of sorted data by the energy spent (Joule) while sorting, whereas SPECpower_ssj2008 (Lange, 2009) divides the throughput (ops) by power consumption in Watt. The ASEE metric (Metri et al., 2012) generalizes this approach as an application specific energy efficiency metric, where an application's energy efficiency is its output (usually throughput) divided by its power consumption. The TPC benchmarks (Poess et al., 2010) also employ this metric, only in reverse (power consumption per performance).

The basic energy efficiency metric, as used in the mentioned benchmarks, works well as long as only a single application is tested at a single load level.

However, it results in multiple separate scores for multiple applications or multiple load levels. SPECpower_ssj2008 (Lange, 2009) addresses this by using the sum of the separate energy efficiency scores as its final score. This approach works as long as the tested load levels remain fixed and as long as the application under test does not change. Further characterization of the energy related behavior of devices over multiple load levels is provided by a class of metrics, referred to as energy proportionality metrics (see Schall et al., 2012 and Hsu and Poole, 2015). These metrics characterize how power consumption scales over load levels. However, they are power metrics only, usually missing the efficiency component.

The existing metrics do not aggregate energy efficiency for multiple workloads or load levels. SPECpower_ssj2008 (Lange, 2009) is the only exception, yet its method is limited to a single workload with a fixed pre-determined number of load levels. In contrast, the metric presented in Chapter 4 provides a single score for multiple workloads and with varying numbers of load levels.

3.3 Load Profiles, Load Distribution, and CPU Performance Counters

In addition to the benchmarks, workloads, and metrics described in the previous section, we investigate the state-of-the art in other areas relevant for efficiency measurement and rating: load profiles, load distribution, and additional instrumentation. Specifically, we focus on load profiles that vary over time, workload distribution on multiple layers within the computing infrastructure, and CPU performance counters. CPU performance counters receive attention in this work, as they can be used to analyze and characterize the workloads under consideration. As mentioned in Section 2.5, we focus on CPU, as it is the hardware component with the highest variability in power consumption (Barroso and Holzle, 2007) and thus the component with the highest potential for power savings.

3.3.1 Load Profiles

Regarding load profiles, several approaches to describe and generate workloads with variable intensity exist in literature. However, they differ from our approach in the following key aspects: Firstly, a set of approaches works purely statistically using independent random variables and therefore does not offer models describing load intensity changes over time. Secondly, approaches for workload or user behavior modeling model the structure of the actual units of work they dispatch or emphasize the behavior of users after their arrival on the system. In contrast, our models from Section 5.1 focus on the description of request or user arrivals, not user behavior and impact after arrival. This is done by combining both deterministic and statistical approaches, which goes beyond existing purely statistical modeling approaches. We group related work into at least one of the following categories:

- **User behavior models**: Hoorn et al., 2008, Roy et al., 2013, and Beitch et al., 2010 propose workload models that capture the behavior and tasks triggered by different types of users. Zakay and Feitelson, 2013 partition workloads according to the user types, and then sample workload traces for each user type to capture the user behavior. This differs from our approach in this thesis. The varying load profiles of this thesis focus on modeling user or work unit arrival processes. We note that models like the above can be easily combined with the approach introduced in Section 5.1 to further characterize the user behavior after a request has arrived at the system and a client session is started.

- **Resource demand focused modeling**, modeling the specific work units: These approaches focus on modeling the units of work processed by the system and estimating the system's resource demands. Casale et al., 2012 focus on modeling bursty workloads, whereas Barford and Crovella, 1998 focus on file distribution and popularity.

- **Statistical inter-arrival models**: These approaches capture the workload intensity using statistical distributions. Feitelson, 2002 creates a statistical model for parallel job schedulers. Li, 2010 models batch workloads for eScience grids and Menascé et al., 2003 as well as Reyes-Lecuona et al., 1999 analyze workloads at multiple levels, such as request and session level. These approaches use independent random variables to capture inter-arrival rate distributions. These random variables do not describe changes in load intensity behavior over time.

- **Regression techniques**, such as MARS (Friedman, 1991), M5 trees (Quinlan et al., 1992), or cubic forests (Kuhn et al., 2012): these techniques are capable of calibrating mathematical functions to fit a measured arrival rate trace. In contrast to explicit user arrival models, they do not convey the additional information of the types and composition of load intensity variation components.

In Section 5.1, we introduce a combined deterministic and statistical approach. This approach is expected to enable a better mapping between arrival rate variations and their respective time-stamps compared to the related work listed here. A composite piece-wise mathematical function, as used in this thesis, can capture load intensity profiles more effectively than independent random variables. This approach also enables a better understanding of a benchmark's behavior for users intending to apply it for benchmarking purposes. This primary purpose of being used in benchmarks is also one of the primary points that sets our approach apart from most of the related works, which are generally not intended for workload generation. The drawback of a deterministic model is the difficulty in capturing random behavior. For this reason, we include an optional random noise element, which deviates from the otherwise deterministic functions and enables a combined deterministic and statistical modeling approach.

3.3.2 Load Distribution

In general, load distribution is considered an important aspect influencing the energy efficiency of systems. Usually, it is considered in the context of workload consolidation. The primary idea behind workload consolidation being that work consolidated onto fewer resources should consume less power and increase efficiency. There are numerous different approaches to workload consolidation, which differ with respect to various aspects regarding the systems the load consolidation is being performed on. Some approaches account for temperature, some for device wear-and-tear, some account for resource provisioning and deprovisioning times, and so on.

Pinheiro et al., 2001 feature one of the first approaches to node-wise load distribution. It showed the promise of workload consolidation, although many of the mechanisms were still executed manually. Chen et al., 2005 introduce a workload distribution mechanism that accounts for power consumption, temperature, and device wear-and-tear. Quan et al., 2012 compare multiple resource allocation strategies, including strategies that use a Unified Modeling Language (UML)-based meta-modeling approach for power prediction. The approach by Kusic et al., 2008 allocates work using lookahead control. It has the goal of minimizing power consumption, while keeping quality-of-service (QoS) above a set threshold. Dorronsoro et al., 2014, Raghavendra et al., 2008, and Verma et al., 2008a feature hierarchical power management approaches. Load consolidation is, however, only managed on a per-machine or core-cluster level. CPU-level task consolidation is evaluated separately by Gomaa et al., 2004 and Heo et al., 2003. Both works focus on power density with the purpose of minimizing heat generation within micro-processors.

Load distribution algorithms are also found in some specialized application areas: For example, Bolla et al., 2014 distribute virtual network functions across a cluster with a focus on energy efficiency while also taking software defined networking into account.

The above mentioned load distribution mechanisms feature advanced decision making engines that decide when to change or reconfigure the current load distribution. The target load distribution is based on basic distribution strategies, primarily load consolidation. We evaluate these basic strategies in Section 11.2 and introduce a new one with the goal of offering additional options to power management decision engines in Section 5.2.

3.3.3 CPU Performance Counters

We explore CPU performance counters as a means to perform additional instrumentation of energy efficiency workloads under consideration. Our goal is the use of such counters to gain additional information about workloads' behavior regarding power consumption and the creation of workloads with specific characteristics, as described in Section 6.1.

Generally, performance counters are often used for software performance analysis or for compiler optimizations (Eyerman et al., 2006 or Cavazos et al., 2007). For example, power consumption aware thread scheduling based on performance counters is explored by Singh et al., 2009 and Bellosa, 2000. Having a framework that can reliably trigger performance events can be used for validation and testing such implementations.

Modeling power consumption based on performance counters is also a possible application. Bircher and John, 2012, Lewis et al., 2008 and Isci and Martonosi, 2003 develop models that estimate power consumption as a function of performance counters, dependent on the workload. Contreras and Martonosi, 2005b create a model based on performance counters but focus on embedded devices with specialized CPU and memory architecture. Kadayif et al., 2001 describe a tool based on performance counters for the UltraSPARC platform which provides energy estimations. These works show that performance counters can be used for power estimations. It is therefore expected that the proposed framework of Section 6.1, which emulates the power consumption-related behavior of a workload by triggering counter events, is a viable approach for power characterization that can help in model validation and in test cases for tooling and instrumentation.

Using performance counters in such a context requires understanding of their accuracy. Zaparanuks et al., 2009 study this accuracy of performance counter measurement facilities. Weaver et al., 2013 identify two major deviations of performance counters, whereas the overhead of common performance counter implementations is researched in a later work (Weaver, 2015). They show that the current PAPI interface has significant overhead.

3.4 Offline Power Prediction

Many modeling approaches for prediction of power consumption of server workloads exist. To illustrate the difference to our approach, we classify existing approaches in four non-exclusive categories: generic server power models, models for virtualized environments, cluster/cloud level models, and offline application power prediction models.

Generic **server power models** characterize or predict the power consumption of a single physical server. They have some overlap with offline power models (e.g., Gurumurthi et al., 2002), yet many are not specifically designed for offline prediction. Rivoire et al., 2008 provides an overview of generalized, generic full-system power models. These models can utilize a variety of methods, such as interpolation (Fan et al., 2007), regression (Lewis et al., 2008; Lee and Brooks, 2006), or others (Chen et al., 2011; Dhiman et al., 2010; Niu et al., 2010). They also vary regarding their purpose and, especially, regarding the type of data that they use to model the power consumption. For this work, we distinguish between hardware-based and workload-based power models. Hardware-based models, such as Lewis et al., 2008; Lee and Brooks, 2006 and Dhiman et al., 2010 are based on system-level data, such as architectural parameters, performance counters and utilization metrics. This data is often collected at run-time or during a calibration step from the system under consideration. Workload-based models, such as Economou et al., 2006a, are usually trained for specific workloads and use application-level parameters such as request arrival rates. In general, these models may be used to model systems that contain a virtualization layer, even though they do not explicitly model it. Some of these methods are used as inspiration for our explicit modeling of the virtualization layer's impact in Section 8.

Virtual machine power models, such as the ones by Choi et al., 2008, Kansal et al., 2010 and Bohra and Chaudhary, 2010, use instrumentation to additionally model the power impact of a virtual machine. These models use online data, gathered from the target system for detailed characterization. Thus, these models focus on immediate run-time decisions based on online measurements. In contrast, our approach in Section 8 enables power prediction before the target system is available for running the target load.

Cluster/cloud level models are often provided as part of larger power management decision engines (Verma et al., 2008a, Basmadjian et al., 2011 and Verma et al., 2008b). These models are created to support online decisions on workload placement within a larger group of servers. They capture the impact of potential decisions in terms of the effects and costs of online reconfigurations. We examine this class of works in greater detail in the following section (Section 3.5 on online power prediction).

Offline application power models, such as those by Basmadjian et al., 2011, Kahng et al., 2009, and Gurumurthi et al., 2002 predict the power consumption of a software application running on servers and are intended for use at design time. Such models can be architecture-based, see Basmadjian et al., 2011 and Stier et al., 2015, which expand upon software architecture models for design time power comparisons. These models are intended to compare software design alternatives for distributed systems and focus on the relative power prediction results with reduced absolute accuracy of single server predictions. In contrast, offline power models for hardware designers focus on maximum accuracy for single hardware components by requiring extremely detailed modeling of their properties. These models, such as the CPU model of Brooks et al., 2000 and Kahng et al., 2009, are usually intended for hardware system designers, analyzing their potential device architectures. Modeling approaches of such a granularity are also employed on a full server level by Gurumurthi et al., 2002, which allows for detailed modeling of full systems. In contrast to these models, our model learns purely from standard benchmark and application efficiency measurement, requiring no explicit modeling by the user.

Our power prediction mechnisms in Chapter 8 differ from the discussed related work in two key aspects:

1. They are designed to leverage available offline data that can be accessed even if the target system is not available. This allows for power prediction of a workload (optionally running within a virtualized environment) on an inaccessible system, for example, before having bought such a system.

2. They consider multiple levels of system load and enable predictions for different load levels.

3.5 Online Power Prediction

Regarding online power prediction, we survey related work in two areas: *Architectural performance models*, and *energy-aware power management*. The architectural performance models show some overlap with the cluster/cloud level models examined in the previous section. In addition, the *system-level power prediction models*, of the previous section remain relevant in the context of online power prediction.

Architectural performance models (e.g., Becker et al., 2009 and Huber et al., 2017) model the software architecture and infrastructure of a component-based system for performance prediction. They can be used for design-time performance prediction (Becker et al., 2009) or run-time prediction (Huber et al., 2017). These models can be combined or extended with power information on power consumption to enable power prediction, as done for example by Stier et al., 2015. An example of an architecture-level model for power prediction, based on fine granular modeling of the software architecture and underlying hardware infrastructure, can be found in the work of Basmadjian et al., 2011.

Architectural models normally require detailed information on the system's software architecture, resource demands, and hardware infrastructure. To reduce the modeling overhead, Huber et al., 2017 propose an approach to automatically extract architectural models at system run-time. This approach, however, has not yet been applied to power prediction.

Energy-aware power management techniques may employ generic power models or advanced online power prediction models (such as the one introduced in Chapter 9 of this thesis) to predict the power consumption of the system states that might result from management decisions. The typical goal is to minimize power consumption within certain Quality-of-Service (QoS) constraints (Beloglazov et al., 2012), or to maximize overall efficiency (Jung et al., 2010). The common technical mechanism by which management systems achieve this is VM migration (Beloglazov et al., 2012; Tian et al., 2012; Jung et al., 2010; Urgaonkar et al., 2010). The impact of the power management decisions is often estimated using established basic power prediction methods. For example, Beloglazov et al., 2012 use a utilization-based linear power model described by Rivoire et al., 2008, whereas Urgaonkar et al., 2010 use a quadratic power model. Other works construct their own model of the underlying systems, such as Tian et al., 2012, who model servers using stochastic Petri nets. Many of such power management models and/or their respective optimization methods also find application in related fields of research, such as smart homes (Mauser et al., 2017, 2014) or Smart Grids (Förderer et al., 2018).

We propose an end-to-end online power prediction model in Chapter 9. This model goes beyond single system power prediction models, as it learns the power consumption of a distributed application on multiple heterogeneous machines. It does so by learning relationships between component distributions and the power consumption of the machines. However, elements of the system-level models can be utilized in our power prediction scenario. Specifically, we leverage established regression models as part of our end-to-end modeling approach.

Part II

Measuring and Rating the Energy Efficiency of Servers

Chapter 4

Methodology for Server Efficiency Rating

Data centers and servers consume a significant amount of electrical power. As future data centers increase in size, it becomes increasingly important to be able to select servers based on their energy efficiency. Rating the efficiency of servers is challenging, as it depends on the software executed on a server and on its load profile. To account for this, a measurement and rating methodology for servers must make use of both realistic and varying workloads. Existing energy efficiency benchmarks either run standardized application loads at multiple load levels or multiple micro-benchmarks at full load. This does not enable a full analysis of system behavior as the energy efficiency for different kinds of software at low load levels remains unknown.

This chapter addresses this thesis' research questions **RQ A.1** and **RQ A.2** of `Goal A`. To address **RQ A.1**: "How to place, execute, and measure workloads in a reproducible and representative manner for energy efficiency rating?", it introduces a measurement methodology for energy efficiency rating of servers that use multiple, specifically chosen workloads at different load levels for a full system characterization. Addressing **RQ A.2**: "How to aggregate results of different multi-stage energy efficiency tests producing a fair energy efficiency metric?", it presents an energy efficiency metric for this methodology. The methodology and workloads have been implemented in the SPEC Server Efficiency Rating Tool (SERT). A significant contribution in this thesis is the evaluation of the methodology in Chapter 10, which shows the applicability and use of the measurement methodology specifically considering its reproducibility, fairness, and relevance. We evaluate the proposed metrics by investigating their energy efficiency rating on a set of 385 different servers.

4.1 Introduction

Improving the energy efficiency of data centers and servers requires the ability to measure and rate that efficiency. A comprehensive rating method can enable data center owners to purchase more efficient devices. It can also help service providers to select the most efficient servers for their specific applications. Finally, a reliable rating method makes it possible for regulatory government agencies to define standards and regulations specifying which devices are considered energy-efficient and which are not. To achieve these goals, the rating method must meet a number of criteria, which are mostly derived from the criteria of a good computer systems benchmark, as described in Chapter 2: It must be relevant, reproducible, fair, and verifiable.

Most servers in modern day data centers are not being utilized to their full capacity. Instead, servers are used to serve requests that arrive over time and are provisioned with additional capacity in order to be able to cope with variations in load, such as unexpected bursts. This leads to an average load somewhere between 10% and 50% (Barroso and Holzle, 2007). In addition, servers are used to run many kinds of different software. This is especially important for rating tools targeted for use by regulators, as a test running a single representative application can not show the many behaviors of the multitude of potential real-world server software setups.

Existing server-efficiency tests do not consider both of these issues. While some tests used for power and efficiency testing, such as JouleSort (Rivoire et al., 2007b), run multiple workloads in a suite, these tests are executed only at full load. In contrast, existing benchmarks that test at multiple load levels are usually application benchmarks that test an application or a set of applications for a specific domain, (e.g., Lange, 2009 and Poess et al., 2010).

This chapter presents a rating methodology for commodity servers including a measurement methodology and metrics for this methodology. It is designed to characterize and rate the energy efficiency of a System under Test (SUT) for multiple contexts and load profiles. To this end, the methodology executes multiple mini-workloads, called worklets, which are run at multiple load levels to showcase differences in system behavior regarding energy efficiency. We also introduce the metrics used for server efficiency calculation. We show how metrics are aggregated over the load levels and worklets in order to arrive at a final energy efficiency score that can, among other things, be used by regulators. The methodology is implemented in the SPEC SERT, which was developed at the request of the U.S. Environmental Protection Agency (U.S. EPA) for use in the Energy Star program for servers. A test using the SERT 2.0, which implements many of the features discussed in this work, is required for acceptance in the Energy Star program.

The goal of this work is the creation of a rating methodology that provides both insight into the behavior of application servers for many potential workloads and load levels, while at the same time still producing a single score energy efficiency metric that can be used by regulators to define criteria on efficiency certification.

The major contributions of this chapter are as follows:

1. A measurement methodology for measuring the power consumption and energy efficiency of servers at multiple load levels,
2. A collection of mini-workloads, called worklets, for analysis of energy efficiency behavior of different server subsystems,
3. Server energy efficiency metrics that account for multiple load levels and workloads and a final single-score metric that can be used for server rating.

We evaluate the methodology and metric in Chapter 10 considering their reproducibility, fairness, and relevance. We investigate the power and performance variations of test results and show fairness of the metric through mathematical proof and a correlation analysis on a set of 385 servers. Finally, we evaluate relevance by showing the relationships that can be established between metric results and third-party applications. We show that our methodology helps decision makers and regulators make reliable decisions on the energy efficiency of servers.

4.2 Server Power Rating Methodology

We present a rating methodology for the energy efficiency of commodity servers for multiple workloads and load levels. It covers device setup, constraints for measurement devices, and workload dispatch and calibration. It also covers the execution order and metrics for multiple workloads run as parts of a larger testing suite. In addition to describing the methodology, we also describe the workloads and mini-workloads (worklets) as shipped with our methodology implementation in the SPEC SERT.

4.2.1 Device and Software Setup

Our device and software setup is based on the setup of the SPECpower_ssj2008 (see Lange, 2009), which is an initial iteration of parts of this methodology, created by some of the methodology's co-authors. The System under Test (SUT) is at the center of the power measurement setup. It is a single physical system, which runs the workloads under test. The SUT's power consumption and its performance during testing are used to derive the energy efficiency score. As

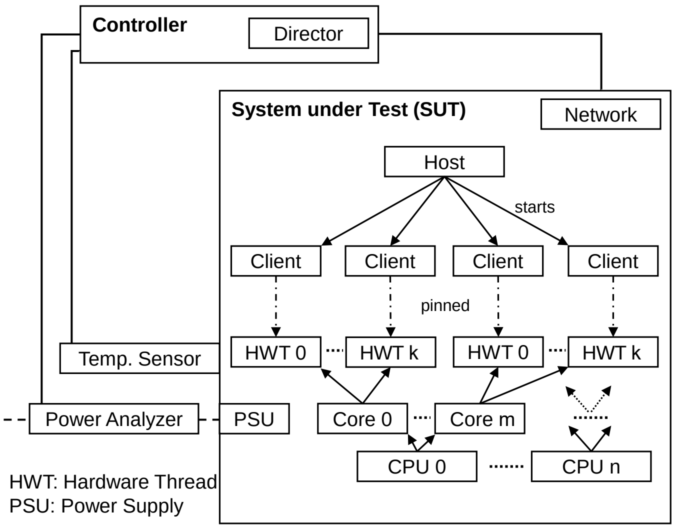


Figure 4.1: Server power measurement device setup.

the purpose of the test is rating the energy efficiency of the SUT device, it should not run any software in addition to the tests itself and the bare minimum of required software, such as the operating system and JVM. Performance metrics are gathered from the SUT using the testing software harness. Specifically, we refer to the actual test execution software on the SUT as the *host*. The host in turn spawns separate *clients* for each logical CPU (logical processor, often referred to as hardware thread). For most workloads, the transactional workload is executed sequentially on the clients. Parallelism is achieved by running multiple clients concurrently. Any computation apart from the actual workload, for example, metric calculation, is run on external devices. The *clients* collect the performance metrics of the workload they execute and forward this information to the host.

The workload is controlled by the *controller* system. The controller system decides which workload to run at which load level. It also collects all measurements both from the SUT (the host on the SUT), as well as external measurement devices and it calculates the metrics and scores. We refer to the collection of software managing all instances and measurement devices on the controller as the *director*. They are illustrated in Fig. 4.1.

Considering external sensors, we require at least one power analyzer and one temperature sensor. The power analyzer measures the power consumption of the entire SUT, whereas the temperature sensor can be used to verify the validity of measurements by assuring that all experiments are conducted under similar environmental conditions. External power and temperature instrumentation is used, as opposed to potential internal instrumentation, as it makes no assumptions about the internal structure of the SUT allowing for maximum portability of the methodology. In addition, it allows tight constraints on the accuracy of the power measurement devices. Specifically, power measurement devices must feature a maximum measurement uncertainty of 1% or better.

4.2.2 Workload and Worklets

The goal of the measurement methodology is the execution of different mini-workloads at multiple load levels each. We call those mini-workloads *worklets* and group them into worklet-collections, which we refer to as *workloads*. Specifically, a workload is a collection of worklets with a common testing goal. All worklets within a workload are designed to test a common resource by utilizing it in a specific fashion. They execute work units, referred to as *transactions*. Our implementation of the methodology in the SERT v2.0 features three separate workloads for CPU, Memory, and Storage. Each of these workloads consists of multiple worklets, which are executed at several load levels. The worklet code has been contributed by different members of the SPEC Open Systems Group (OSG) Power subcommittee.

In the following, we describe each of the workloads in detail. Each of the workloads was designed so that it primarily stresses the server subsystem after which it was named (CPU workload stresses CPU, Memory workload stresses memory, Storage workload stresses internal storage devices).

4.2.2.1 CPU

We define the CPU workload as the collection of all CPU worklets. SERT implements a total of seven different CPU worklets:

1. **Compress**: De-/compresses data using a modified Lempel-Ziv-Welch (LZW) method (Welch, 1984)
2. **CryptoAES**: Encrypts/decrypts data using the AES block cipher algorithms
3. **LU**: Computes the LU factorization of a dense matrix using partial pivoting
4. **SHA256**: Performs SHA-256 hashing transformations on a byte array

5. **SOR** (Jacobi Successive Over-Relaxation): Exercises typical access patterns in finite difference applications
6. **SORT** Sorts a randomized 64-bit integer array during each transaction
7. **Hybrid / SSJ**: Performs multiple different simultaneous transactions, simulating an enterprise application

4.2.2.2 Memory

The memory workload consists of worklets that have been designed to scale with installed memory. The primary memory characteristics being tested are bandwidth and capacity.

The memory worklets serve as the major exception to the load level and interval specification in Section 4.2.3. In contrast to other worklets, they do not scale via transaction rate, but instead scale with memory capacity. In addition, they do not use throughput as their performance metric, but they modify it to include bandwidth and/or capacity.

1. **Flood**: A sequential memory bandwidth test that exercises memory using arithmetic operations and copy instructions. Flood's performance score is a function of both the memory capacity and the bandwidth measured during testing.
2. **Capacity**: A memory capacity test that performs XML operations on a minimum and maximum data set. The final metric is a function of transaction rate and physical memory size.

4.2.2.3 Storage

We include a workload for testing storage in order to enable a well-rounded system test. Storage worklets test the server's internal storage devices.

1. **Random**: Reads and writes data to/from random file locations.
2. **Sequential**: Reads and writes data to/from file locations that are picked sequentially

4.2.2.4 Idle

Idle keeps the system in an idle state in order to measure the idle power consumption. Idle does not perform any work and has no performance. Consequently, it does not measure any efficiency metric, but only power consumption.

4.2.3 Worklet Dispatch and Load Levels

Worklets are designed to be executed at multiple load levels. We define these load levels using the transaction rate at which a worklet is run. Each worklet features transactions, which are the basic work units performed by the worklet (e.g., one array being compressed is one transaction for the Compress worklet). The maximum (100%) load level is defined as the load level at which as many transactions as possible are executed per second on the respective SUT. Lower load levels are achieved by deliberately waiting in between separate transactions in order to achieve lower performance and lower system utilization. To this end, the SUT must run each worklet in a calibration mode to determine the maximum transaction rate that the worklet can achieve. For each target load level (e.g., 100%, 75%, 50%, 25%), the director calculates the target transaction rate and derives the corresponding mean time from the start of one transaction to the start of the next transaction. During measurement, these delays are randomized using an exponential distribution that statistically converges to the desired transaction rate. The randomization ensures that different CPU cores are not always active at the exact same time. Please note, that load levels do not equal CPU utilization, which is a common misconception. A load level of 50% means that a worklet only runs 50% as many transactions as compared to full load, it does not necessarily mean that the CPU is utilized at 50%. Scaling via transactions, instead of CPU utlization, allows the methodology to be used in a broader setting, i.e., for non-CPU heavy workloads.

This generic load level calibration is used for all worklets except for the memory worklets. The memory worklets have been desigend to scale with memory capacity instead of transaction rate, providing an additional insight into system behavior.

4.2.3.1 Phases and Intervals

All worklets are run in three phases: A warm-up phase, a calibration phase, and a measurement phase. The warm-up phase runs the worklet at full load for a short period of time without performing any measurements. It ensures that transient effects that occur at the first execution of each worklet do not affect power and performance measurements. Common transient effects are caching effects and side-effects of the Java Just-In-Time (JIT) compiler. After warm-up, the worklet enters the calibration phase. During calibration, worklet transactions are executed as fast as possible. This means that for each execution unit (hardware thread) a transaction is scheduled exactly as the execution unit becomes available, without a waiting time after conclusion of the previous transaction. The result of the calibration phase is the maximum transaction rate of the worklet on the specific SUT. Finally, the measurement phase takes place. In the measurement phase, transactions are scheduled according to the load level specification (see Section 4.2.3).

Each phase is split into a configurable number of intervals, which serve different purposes, depending on the phase in question. Each interval is the period in time during which the actual work of the phase is being done. The durations of these intervals have been determined based on practices established in prior works (Lange, 2009) and then modified using results from internal tests. They are evaluated implicitly in our reproducibility analysis in Chapter 10. The worklet is put to sleep for ten seconds between intervals (and phases), allowing the hardware to cool down and become ready for the next interval. Each interval is also split into three time periods, a pre-measurement, measurement, and post-measurement period. The pre-measurement period allows for an additional warm-up for each separate interval, whereas the post-measurement period ensures that the worklet and hardware do not begin shutdown during measurement. In these periods, the worklets are already being executed at the target load level, yet no measurements are recorded. Per default the pre-measurement and a post-measurement period run for a duration of 15 seconds each. The measurement period performs the phase-specific work. It measures the maximum throughput during calibration and the current throughput, as well as the power consumption during the measurement phase. Per default, each measurement period in a measurement interval is executed for a duration of 120 seconds. Within this time, all transactions are logged and power measurements are reported at one second intervals. The interval's power consumption is the average over the 120 reported values.

Each phase runs its intervals in sequence. The type of sequence depends on the phase in question. The warm-up phase only runs a single interval of varying length, whereas the calibration phase runs multiple identical calibration intervals in sequence. The calibration result is the average throughput of those intervals. The measurement phase runs its intervals in the so-called graduated measurement sequence when adhering to the load level specification. The graduated measurement sequence executes the intervals at gradually diminishing target transaction rates. Fig. 4.2 illustrates an example calibration phase with a single calibration interval and a measurement phase with a graduated measurement sequence, containing three intervals.

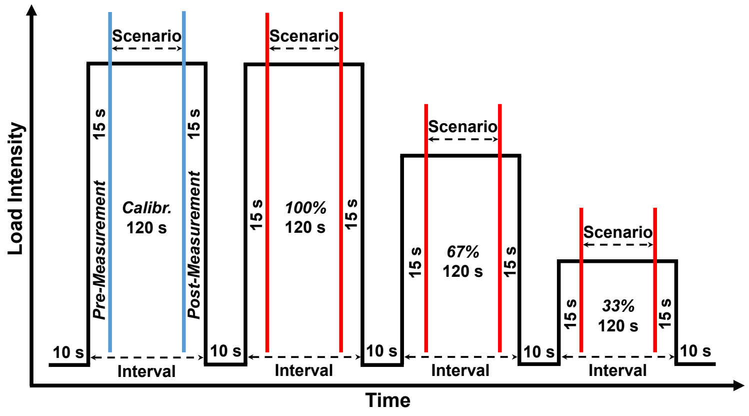


Figure 4.2: Intervals for calibration and measurement phase.

4.3 Energy-Efficiency Metrics

We introduce multiple energy efficiency metrics for measurements obtained using our methodology. The measurement methodology is designed to run multiple separate worklets at multiple load levels, which are then grouped into workloads. Metrics must provide insight into these stages of measurement and should also reflect the structure of the methodology. Our approach addresses this challenge by providing energy efficiency metrics at the following steps in the methodology. For these steps each efficiency metric is an aggregate of the efficiency at the subordinate levels of measurement. Separate energy efficiency metrics are calculated at these steps, as illustrated in Fig. 4.3:

1. Interval efficiency
2. Worklet efficiency (over all load levels)
3. Workload efficiency (for all worklets)
4. Total efficiency

The basic interval efficiency metric is the energy efficiency as a ratio of the performance over the power consumption during the interval. Worklet efficiency is an aggregate of all load level intervals that were measured for any given worklet. Workload efficiency is the aggregate energy efficiency of all the worklets grouped within the workload. Finally, we propose a final score

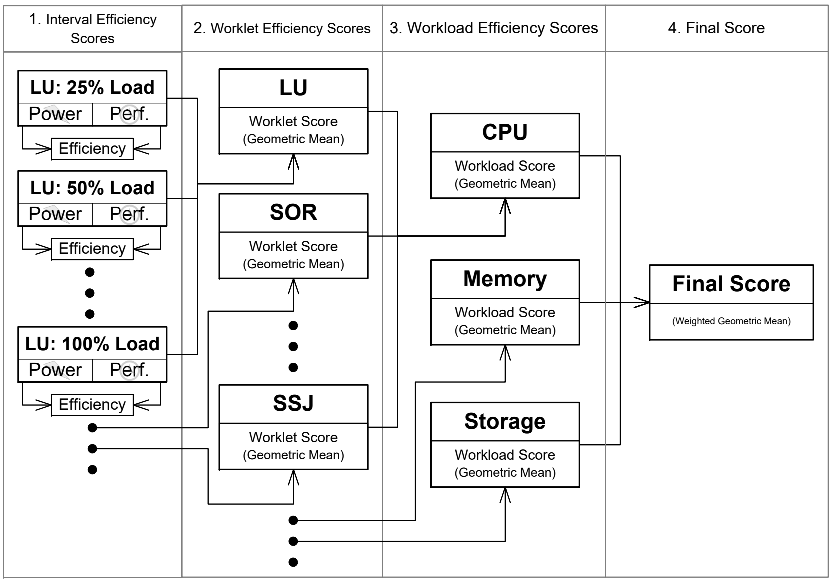

1. Interval Efficiency Scores
2. Worklet Efficiency Scores
3. Workload Efficiency Scores
4. Final Score
LU: 25% Load
Power
Perf.
Efficiency
LU: 50% Load
LU: 100% Load
LU
Worklet Score (Geometric Mean)
SOR
SSJ
CPU
Workload Score (Geometric Mean)
Memory
Storage
Final Score
(Weighted Geometric Mean)

Figure 4.3: Calculation order of energy efficiency metrics.

that provides a single number metric for the entire suite of workloads. The final score is primarily intended to be used by regulators when making server energy efficiency labeling decisions.

4.3.1 Worklet Performance Metrics

Our methodology uses throughput as its primary performance metric for the transactional workloads. Considering that throughput operates on different scales depending on the size of transactions, we normalize it using the throughput of a reference system. *Normalized throughput (η)* thus refers to the relative speedup of the SUT compared to the reference system (Eq. 4.1).

$$\eta = \frac{\tau}{\rho} \tag{4.1}$$

where η is the normalized throughput, τ the measured throughput, and ρ the reference throughput.

In addition to normalizing the average throughput, we also report its coefficient of variation. This enables users of the methodology to gage the stability of the load levels and enhances confidence in the results.

The memory worklets serve as an exception. Both worklets have been designed to scale with the available memory on the SUT. This is a direct result of the scalability requirement mentioned in Section 2.4. However, the performance score should also reward systems with greater memory capacity, considering that greater capacity allows for more caching, etc. in large server scale applications. Yet, scaling perfectly, the memory worklets can not test for this property. In addition, a test that is designed to use a certain amount of memory, more than some servers provide, might easily become outdated in the future as the amount of memory increases. To cope with this challenge, memory worklets include the memory capacity in their performance scores.

Flood uses throughput (memory bandwidth) of its memory accesses as its base performance. In contrast to all other worklets, its performance metric is not load level agnostic, as it does not scale using transaction rate, but using memory capacity. The Flood performance score for each load level is the measured bandwidth multiplied by the load level.

Capacity uses regular throughput, but multiplies it with the square root of the system's physical memory capacity, arriving at capacity performance κ (Eq. 4.2). This allows the capacity worklet to easily represent the system's capacity for any future system. The square root has been chosen, as memory DIMM sizes have continuously doubled in the past.

$$\kappa = \tau * \sqrt{\mu} \tag{4.2}$$

where κ is the capacity performance, τ the measured throughput, and μ the physical memory size.

4.3.2 Efficiency Metrics

We calculate energy efficiency separately for each load level / interval. During each interval, we measure power consumption and average throughput. We derive normalized throughput. Interval efficiency is then calculated by dividing normalized throughput by power consumption. Considering that normalized throughput is often significantly smaller than the power consumption in practice, and considering it has no upper bound, we multiply this score with a constant factor of 1000 (Eq. 4.3). This is a cosmetic factor that has been chosen in order to move the resulting score into a number range that is easier to read for a human reader.

$$\iota = \frac{\eta}{\phi} * 1000 \tag{4.3}$$

where ι is the interval efficiency, η the normalized throughput, and ϕ the power consumption.

Interval energy efficiency is similar to standard energy efficiency, as found in literature (Metri et al., 2012; Poess et al., 2010; Lange, 2009). The major difference that sets our version of this metric apart is the normalization of performance. Note that throughput divided by power consumption (in Watt) is mathematically identical to transactions per Joule, as used by Rivoire et al., 2007b. Based on the separate interval energy efficiency metrics, we iteratively calculate Worklet energy efficiency, then Workload energy efficiency, and, finally, total energy efficiency.

4.3.2.1 Worklet Energy Efficiency

The worklet energy efficiency score γ is calculated using the geometric mean of each worklet's separate interval scores (Eq. 4.4). We use the geometric mean over the arithmetic mean as it is known to preserve ratios (such as energy efficiency). The geometric mean is chosen in favor of the arithmetic mean or sum, as used in SPECpower_ssj2008 (Lange, 2009). The major difference between the geometric mean and arithmetic mean is that the arithmetic mean favors load levels with higher efficiency scores. Traditionally, higher load levels also feature higher efficiency scores on most systems. As a result, the arithmetic mean usually favors the results of high load levels. A change in energy efficiency at a higher load level has a greater impact on the final score than it would at a lower load level. The geometric mean, on the other hand, treats relative changes in efficiency equally at each load level.

$$\gamma = \exp\left(\frac{1}{n} * \sum_{i=1}^{n} \ln(\iota_i)\right) \tag{4.4}$$

where γ is the worklet efficiency, n the number of load levels per worklet, and ι_i the energy efficiency for load level i.

4.3.2.2 Workload Energy Efficiency

Workload efficiency ψ is calculated by aggregating the efficiency scores of all worklets within the workload using the geometric mean (Eq. 4.5). The worklet efficiency scores are the results of the score calculation in Eq. 4.4.

$$\psi = \exp\left(\frac{1}{m} * \sum_{i=1}^{m} \ln(\gamma_i)\right) \tag{4.5}$$

where ψ is the workload efficiency, m the number of worklets per workload and γ_i the energy-efficiency for each specific worklet.

4.3.2.3 Final Aggregate Energy Efficiency

The server energy-efficiency score σ is the final aggregate of the workload scores. It too is derived using the geometric mean. In contrast to the other geometric mean aggregates, the final score does not consider all workloads equally. Instead, it uses a weighted mean, putting a different focus on each of the workload scores. Weights may be chosen freely to enable a fair comparison for each specific use case (Eq. 4.6). For such specific use-cases, they can be found by investigating and comparing on-server resource usage, as done in Section 10.3, or using regression, similar to the approach of Chapter 8.

$$\sigma = \exp\left(\frac{1}{k} * \sum_{i=1}^{k} (w_i * \ln(\psi_i))\right) \tag{4.6}$$

where σ is the server efficiency, k the number of workloads, ψ_i the ith workload, and w_i the weight assigned to this workload.

Specific weights, used by the U.S. EPA with their adoption of the methodology's implementation in the SERT v2.0 are as follows:

- **High** CPU weight: 65%
- **Medium** Memory weight: 30%
- **Low** Storage weight: 5%

This specific weighting is targeted at regular data center compute nodes, resulting in a high CPU and medium memory weight that is intended to mirror a typical real-world compute workload's resource profile. Storage is weighted with a low 5% weight, as storage servers are not the target devices for this weighting.

4.4 Concluding Remarks

This chapter introduced a power rating methodology for servers. This methodology is our answer to **RQ A.1** ("How to place, execute, and measure workloads in a reproducible and representative manner for energy efficiency rating?"). It specifies SUT instrumentation, workload dispatch, a collection of reference workloads, measurement phases and intervals, and power and energy efficiency metrics. The metrics, in turn, are our answer to **RQ A.2** ("How to aggregate results of different multi-stage energy efficiency tests producing a fair energy efficiency metric?"). We evaluate this methodology in Chapter 10 for three major quality criteria: reproducibility, fairness, and relevance. The contributions of this chapter and the work towards these contributions has been published in the Proceedings of the ICPE 2015, ICPE 2016, and ICPE 2018 conferences (Kistowski et al., 2015c, Kistowski et al., 2016a, and Kistowski et al., 2018c).

Chapter 5

Advanced Load Profiles for Energy Efficiency Measurement

Today's system developers and operators face the challenge of creating software systems that make efficient use of dynamically allocated resources under highly variable and dynamic load profiles, while at the same time delivering reliable performance. Benchmarking of systems under these constraints is difficult, as state-of-the-art benchmarking frameworks provide only limited support for emulating such dynamic and highly variable load profiles for the creation of realistic workload scenarios. Load distribution plays a crucial role in this context. It can be used to improve energy efficiency of servers as (un-)balancing strategies can be leveraged to distribute the varying load over one or multiple systems. Ideally, load is distributed in a way in which resources are utilized at high performance, yet low overall power consumption. This can be achieved on multiple levels, from load distribution on single CPU cores to machine level load distribution on distributed systems.

With modern day server architectures providing load distribution opportunities at several layers, evaluating and answering the question of optimal load distribution has become non-trivial. Work has to be distributed hierarchically in a fashion that enables maximum energy efficiency at each level. Current load distribution approaches do not test this. Instead, they distribute load based on generalized assumptions about the energy efficiency of servers. These assumptions are based either on very machine-specific or highly generalized observations that may or may not hold true over a variety of systems and configurations. Similarly, standard benchmarks and benchmarking methodologies, including base methodology, introduced in Chapter 4, do not support varying loads over time, but typically confine themselves to workloads with constant or stepwise increasing loads. Alternatively, they support replaying of recorded load traces. Statistical load intensity descriptions also do not sufficiently capture concrete load profile variations over time.

This chapter presents two contributions that enable use of advanced load profiles for power and efficiency testing and research. Addressing **RQ A.3**: "How to model and create realistic, varying load profiles for energy efficiency testing?" of `Goal A`, we present a method for modeling varying load profiles over time. Addressing **RQ A.4**: "How to heterogeneously distribute load for different placements in energy efficiency testing?", we introduce workload distribution mechanisms for hierarchic distribution of workloads on the execution resources within servers or clusters.

Firstly, we present the *Descartes Load Intensity Model* (DLIM). DLIM provides a modeling formalism for describing load intensity variations over time. A DLIM instance can be used as a compact representation of a recorded load intensity trace, providing a powerful tool for benchmarking and performance analysis. As manually obtaining DLIM instances can be time consuming, we present an automated extraction method. Secondly, we introduce a modification to our methodology for evaluation of load distribution strategies. We use a modified version of the SPEC SERT implementation of our power methodology to measure the energy efficiency of a variety of hierarchical load distribution strategies on single and multi-node systems. We also introduce a new strategy and evaluate energy efficiency for homogeneous and heterogeneous workloads over different hardware configurations.

We evaluate both the load intensity models and load distributions in Chapter 11. DLIM model expressiveness is validated using the presented extraction method. Extracted DLIM instances exhibit a median modeling error of 12.4% on average over nine different real-world traces covering between two weeks and seven months. Additionally, extraction methods perform orders of magnitude faster than existing time series decomposition approaches. Our results analyzing the load distributions show that the selection of a load distribution strategy depends heavily on workload, system utilization, as well as hardware. Used in conjunction with existing strategies, our new load distribution strategy can reduce a single system's power consumption by up to 10.7%.

5.1 Load Profiles with Varying Load Intensity over Time

Today's cloud and web-based IT services need to handle large numbers of concurrent users under highly variable and dynamic load intensities. Customers access services independently of each other and expect a stable Quality-of-Service (QoS). In this context, any knowledge about a service's load intensity profile becomes a crucial information for managing the underlying IT resource landscape. Load profiles with large amounts of concurrent users are typically strongly influenced by deterministic patterns due to human habits, trends, calendar effects, and events. The performance evaluation of systems under these dynamic conditions poses new challenges. Our power benchmarking methodology from Chapter 4, as well as benchmarking frameworks such as Faban[1], Rain (see Beitch et al., 2010), and JMeter (see Halili, 2008) allow request injection rates to be configured either to constant values, or to stepwise increasing rates (e.g., for stress tests) or variable rates based on recorded workload traces. The challenges arising when applying open workload models for benchmarking are not thoroughly addressed by these frameworks, as varying load intensity profiles are a common observation in real world systems and require consideration in benchmarking.

In this section, we address **RQ A.3**: "How to model and create realistic, varying load profiles for energy efficiency testing?" by introducing the *Descartes Load Intensity Model* (DLIM). DLIM offers a fine-grained, structured and accessible Meta-Object Facility (MOF)-based meta-model to describe load intensity profiles by editing and combining piece-wise mathematical functions. At a higher abstraction level, we propose the *high-level Descartes Load Intensity Model* (hl-DLIM) to support the description of load variations using a small set of parameters to characterize seasonal patterns, trends, as well as bursts and noise. DLIM can be used to define an arbitrary dynamic load intensity profile that can be leveraged for benchmarking purposes to evaluate the behavior of a system under different dynamic workload scenarios (e.g., bursty workloads, seasonal load spikes). This is useful in many use-cases, for example, for both online and off-line evaluation of the quality of system adaptation mechanisms such as elastic resource provisioning techniques in modern cloud environments. DLIM and hl-DLIM are both MOF-based meta-models for load intensity description. This allows the use of tools and techniques provided for model driven development in conjunction with DLIM. As such, we offer a model-to-model transformation from hl-DLIM to the detailed DLIM. In contrast to pure regression approaches, DLIM offers the advantage of classifying load intensity variations by type, as they are fitted to certain model elements. As a result, models include additional type information, which is useful when analyzing or modifying load intensity variations. Further DLIM-based applications and developments in the fields of benchmarking and system resource management at run-time are enabled by providing automatic model extraction processes. Specifically, we envision the use of automatically extracted load intensity profiles as part of autonomic and self-aware system management by employing them in the following contexts:

- **Benchmarking and Analysis**: Using the load intensity model, we can

[1]Faban `http://faban.org`

create, extract, and modify load profiles to create testing profiles specifically designed to elicit target system behavior for testing. This is the prime use-case for which DLIM is used in this work.

- **Load Forecasting**: Automatically extracted load intensity profiles can be used to forecast the changes in arrival rates at run-time. This, in turn, enables pro-active resource management and system adaptation. A description of DLIM's use in this context can be found in the thesis of Herbst, 2018.
- **Anomaly Detection**: A load profile model can serve as a baseline for the online detection of anomalous load behavior, such as unplanned bursts.
- **Load Classification**: The compact information about load behavior contained within the model can be used to classify profile categories, enabling dynamic distinction between user or application types based on profile characteristics.

A load intensity profile, represented as a DLIM model instance, can be created either manually by the user or it can be extracted from a request arrival trace obtained by monitoring a real-life production system. Given a trace, the trace can be represented in a mathematical form as a compact DLIM model instance. The latter captures the major properties of the trace (e.g., burstiness, seasonality, patterns and trends) and can be used at any time to automatically generate an equivalent trace (exhibiting the same properties). Furthermore, starting with an extracted model instance, the instance can be easily modified to reflect a given target dynamic load scenario under investigation, for example, changing the frequency of bursts or adding a given trend behavior. Load intensity profiles, represented as DLIM model instances, can be used in a variety of application scenarios, for example, for emulating request/job arrivals in benchmarking experiments, or for analyzing mathematical properties of real-life workloads (e.g., burstiness, seasonality). In the latter case, load intensity profiles provide valuable information for performance modeling and capacity planning studies. Manual construction and maintenance of DLIM model instances becomes infeasible in complex scenarios or when using it at run-time. hl-DLIM addresses this by providing a more concise way for load intensity profile description. This enables the quick and easy creation of an initial model instance. For the easy creation of model instances based on real-world data, we introduce and validate an automated DLIM model extraction method, which we call the *Simple DLIM Extraction Method* (s-DLIM) for DLIM instances. It is based on the idea of time series decomposition as taken in Breaks for Additive Season and Trends (BFAST, introduced by Verbesselt et al., 2010). Additional DLIM extraction methods that enhance DLIM's use in different contexts, such as in load forecasting, are found in the thesis of Herbst, 2018.

Chapter 11, Section 11.1 evaluates DLIM's expressivness and highlights as its major benefits the new capabilities to accurately automatically extract load intensity models (12,7% median modeling error on average) from a representative set of nine different real-world traces. Each extraction completes in less than 0.2 seconds. These results demonstrate and validate the capability of DLIM to capture realistic load intensity profiles. An implementation of the models and extraction methods is available as part of the LIMBO toolkit[2]. We use this implementation as part of our work in Chapters 6 and 9, which demonstrates the usefulness of our method.

5.1.1 Descartes Load Intensity Model

The Descartes Load Intensity Model (DLIM) describes the profile of arrival rates over time with a focus on describing variations and capturing characteristic load intensity behaviors. At its core, it uses piece-wise mathematical functions to approximate the varying rates with support for periodicity and flexibility to adapt and incorporate unplanned events. The DLIM meta-model is visualized in Fig. 5.1.

Being a composition of piece-wise mathematical functions, DLIM uses a *Sequence* of functions as its central element. Functions may be added or multiplied with one another using mathematical operators. The result of this approach is a sequence of function-trees, which describe the arrival rate behavior during a time period as defined by the containing *Sequence*. Specifically, a *Sequence* carries a number of *TimeDependentFunctionContainers*, which describe the duration of each interval and are executed in sequence. The containers, in turn, contain the actual mathematical functions describing the arrival rates. The *Sequence*'s execution repeats as many times as indicated by the *terminateAfterLoops* attribute. Alternatively, the sequence repeats for the time indicated by the *terminateAfterTime* attribute. If both are set, we calculate the final duration as the minimum of either the looping time or the specific *terminateAfterTime* attribute (Eq. 5.1).

$$\begin{aligned} finalDuration =& min(terminateAfterLoops * loopDuration, \\ & terminateAfterTime) \end{aligned} \tag{5.1}$$

[2]LIMBO `http://descartes.tools/limbo`

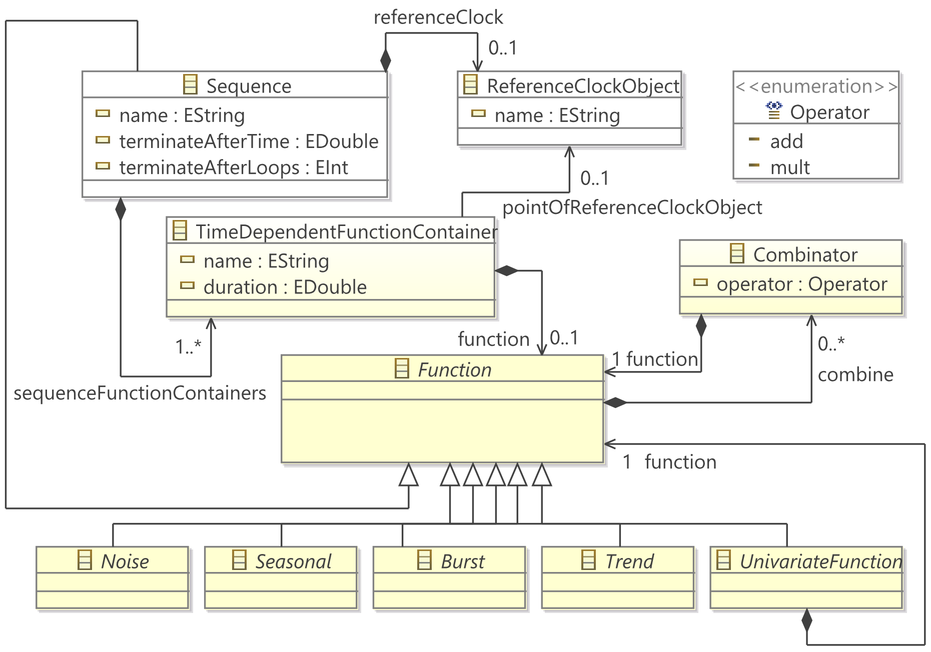


Figure 5.1: The Descartes Load Intensity Meta-Model without the child implementations of the abstract *Noise*, *Burst*, *Seasonal*, and *Trend*.

Infinite sequences are not allowed in order to guarantee termination. Any *Function* can be combined with other *Functions* using a *Combinator*, which results in a *TimeDependentFunctionContainer* carrying an entire tree of functions. The *TimeDependentFunctionContainer* describes its arrival rates for a set *duration*, after which the next *TimeDependentFunctionContainer* in the parent *Sequence's* list is processed.

Function is the abstract parent class to mathematical functions contained within a *TimeDependentFunctionContainer*. Being abstract, it cannot be instantiated. Instead a number of non-abstract children are provided that can be used as *Functions*. The most notable concrete *Function* is the *Sequence*, which effectively means that any *TimeDependentFunctionContainer* may hold a *Sequence* in its *Function* tree. This contained *Sequence* must be unique, preventing cyclical containment dependencies. The other concrete *Functions* fall into one of the following categories (each represented by their abstract super-class): *Seasonal*, *Burst*, *Noise*, or *Trend*.

A *Function* holds a list of *Combinators*. A *Combinator* allows the combination of this *Function*'s arrival rates with the arrival rates generated by other concurrently running *Functions*. The *Combinator* contains operators such as $+$ and $*$. Any *Function* contained within a *Combinator* is defined for the exact same duration as its containing parent *Function*.

5.1.2 High-level DLIM

DLIM offers a convenient way of structuring and ordering functions for the description of load intensity profiles. However, from the standpoint of a human user, it provides limited abstract knowledge about those variations, as the tree of composite piece-wise mathematical functions may be difficult to grasp. hl-DLIM addresses this issue by providing means to capture load intensity variations with a limited number of parameters. These parameters can then be used to quickly define and characterize a load intensity model. Inspired by the time-series decomposition approach in BFAST (Verbesselt et al., 2010), hl-DLIM is split into a *Seasonal* and *Trend* part. Additionally, as hl-DLIM is intended for modeling load profiles, it features a *Burst* and a *Noise* part. In contrast to DLIM, it is designed to model a subset of the most common load variation profiles in favor of better readability.

The *Seasonal* part describes the underlying composite function that repeats after every seasonal duration (e.g., every day in a month long load intensity description). hl-DLIM describes the seasonal part using the following parameters (as shown in Fig. 5.2): *period*, *number of peaks*, *base arrival rate level*, *first peak arrival rate*, and *last peak arrival rate*. Additional peak arrival rates are derived using linear interpolation.

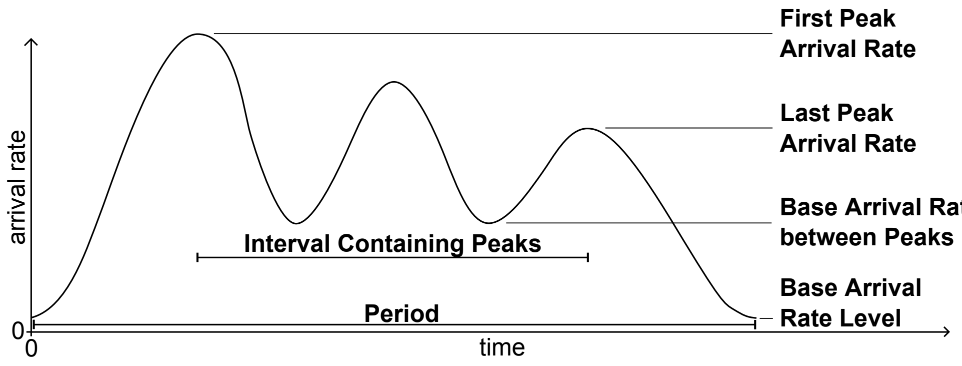


Figure 5.2: hl-DLIM *Seasonal* part.

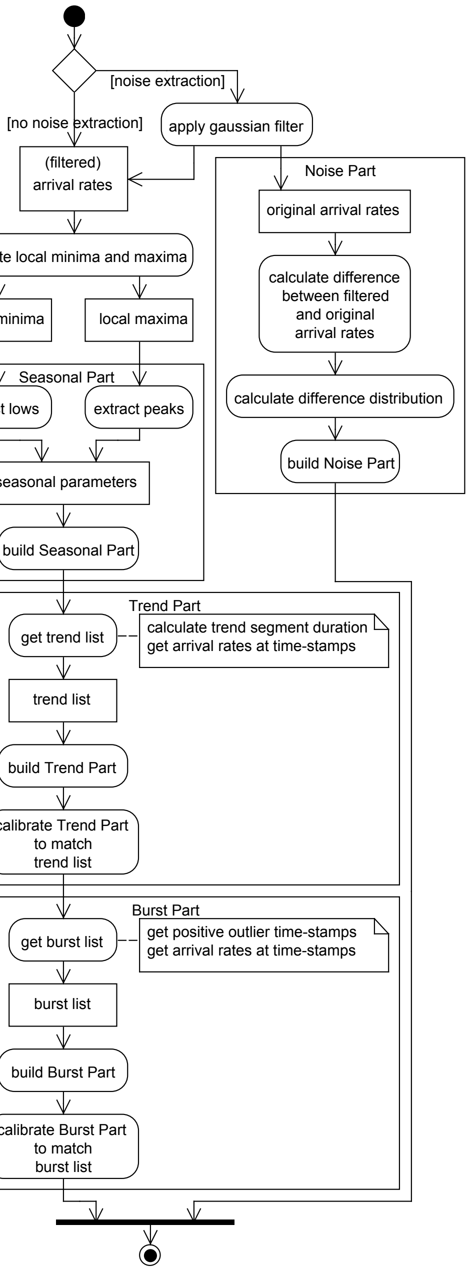

[noise extraction]
[no noise extraction]
apply gaussian filter
(filtered) arrival rates
calculate local minima and maxima
local minima
local maxima
Seasonal Part
extract lows
extract peaks
seasonal parameters
build Seasonal Part
Noise Part
original arrival rates
calculate difference between filtered and original arrival rates
calculate difference distribution
build Noise Part
Trend Part
get trend list
calculate trend segment duration
get arrival rates at time-stamps
trend list
build Trend Part
calibrate Trend Part to match trend list
Burst Part
get burst list
get positive outlier time-stamps
get arrival rates at time-stamps
burst list
build Burst Part
calibrate Burst Part to match burst list

Figure 5.3: Activity diagram of the *Simple DLIM Extraction Method.* (s-DLIM)

The *Trend* part describes an overarching function that captures the overall change in the load intensity over multiple seasonal periods. It does so by constructing a list of equi-length *Trend* segments. These segments describe the respective arrival rates at their start and end points. The arrival rate of the *Seasonal* part is then interpolated to match the segment's arrival rate. In contrast to the *Trend* within BFAST, the hl-DLIM *Trend* can interact with the *Seasonal* part either by addition or multiplication. A multiplicative trend holds promise for the domain of arrival rate modeling, as existing traces seem to indicate a greater *Trend* impact on peaks in comparison to its impact on local minima. The *Trend* part is defined using the following parameters: *number of seasonal periods within one trend* (i.e., the length of a single trend segment), *operator* (addition or multiplication), and the *list of seasonal arrival rate peaks*. The latter defines the arrival rate at the beginning and end of the *Trend* segments. The *Trend* segment functions are defined so that they always begin and end at the largest *Seasonal Peak*. The concrete *Trend* function, interpolating between each segment's start and end point, can be either a *linear*, *exponential*, *logarithmic*, or *sin* function. As a result, the values contained in this list define the resulting maximum peak after applying the *Trend* at the corresponding point in time. The point in time at which each arrival rate in this list is defined is always the time of the largest peak in a *Seasonal* iteration.

The *Burst* part allows the definition of recurring bursts, which are added onto the existing *Seasonal* and *Trend* behavior. It is defined using the following parameters: *first burst offset*, *inter burst period*, *burst peak arrival rate*, and *burst width*.

The *Noise* part allows the addition of a uniform distributed noise function defined by its *Minimum Noise Rate* and *Maximum Noise Rate*. Other Noise distributions can easily be added to DLIM instances, which are obtained from hl-DLIM instances via a model-to-model transformation.

5.1.3 Model Instance Extraction

In this section, we present the *Simple DLIM Extraction Method* (s-DLIM). It extracts a DLIM instance. This process (and its resulting DLIM instance) are inspired by the time-series decomposition approach used in BFAST (Verbesselt et al., 2010). s-DLIM extracts a repeating *Seasonal Part* and a non-repeating *Trend Part*. The non-repeating *Trend Part* contains a list of *Trend* segments of fixed size that interpolate between their start and end arrival rate value. The *Trend* list extends throughout the entire time duration for which the extracted model is defined. Additionally, a *Burst Part* and an optional *Noise Part* are extracted. s-DLIM is visualized in Fig. 5.3.

5.1.3.1 Extracting an s-DLIM Instance

The following sections describe the extraction of the different model parts by s-DLIM.

Extracting the Seasonal Part The *Seasonal Part* of the arrival rate trace is modeled using a *Sequence* of *TimeDependentFunctionContainers* and their *Functions*. Each *Function* interpolates the corresponding peaks and lows within each seasonal period. As a result, the following data needs to be derived in order to build the *Seasonal Part*:

- Duration of the seasonal period
- Arrival rate peaks and their time-stamps
- Arrival rate lows and their time-stamps
- Function type used to interpolate between peaks and lows.

The seasonal period (length) is set by the user. It is usually selected using expert knowledge about the trace. A trace that extends over multiple days and contains daily patterns, for example, features a period of 24 hours. The peaks and lows are automatically determined by using a local minimum/maximum search on the arrival rates within the trace. The local arrival rate minima and maxima and their corresponding time-stamps within a seasonal period constitute the peaks and lows. Since the trace usually contains multiple seasonal periods, the respective median arrival rate value is selected for each local maximum and minimum within the *Seasonal Part*. Selecting the median instead of the mean reduces the impact of outliers on the extracted value. As a result, the derived functions interpolate first between the first median low and the first median peak, then between the first median peak and the second median low, and so on. The last low must be of the same arrival rate as the first low in order for the *Seasonal Part* to repeat seamlessly. The type of the interpolating function (linear, exponential, logarithmic, sin) can be selected by the user, depending on his needs. According to our experience, the sin interpolation usually results in a good model fit. The *Seasonal Part* extraction is illustrated in Algorithm 5.1.

Extracting the Trend Part The *Trend Part* consist of a series of functions (trend segments) that are either added or multiplied onto the *Seasonal Part*. Each trend segment begins at the maximum peak of the *Seasonal Part* and ends at the maximum peak of the *Seasonal Part* in a later *Seasonal* iteration.

Algorithm 5.1: Extracting the *Seasonal* part.

```
   Data: duration: seasonal period duration,
   LIST: list of tuples $\vec{t} = \begin{pmatrix} arrivalRate \\ timeStamp \end{pmatrix}$,
1  rootSequence: root Sequence of the DLIM instance
2  Function extractSeasonalPart()
3      MIN ← getLocalMinima (LIST)
4      MAX ← getLocalMaxima (LIST)
5      peakNum ← median(number of peaks within each Seasonal iteration)
6      for $i \leftarrow 0$ to peakNum $- 1$ do
7          peak [i].arrivalRate ← median(arrival rate of all $i_{th}$ peaks $\in$ MAX
             within each seasonal iteration)
8          peak [i].timeStamp ← median(time stamp of all $i_{th}$ peaks $\in$ MAX
             within each seasonal iteration)
           /* In seasonal iterations with more than peakNum peaks,
              the $i_{th}$ peak is selected, so that peaks are evenly
              spaced throughout that seasonal iteration.           */
9          peak [i] ← $\begin{pmatrix} \text{peak}[i].arrivalRate \\ \text{peak}[i].timeStamp \end{pmatrix}$;
10     for $i \leftarrow 0$ to peakNum $- 1$ do
11         low [i].arrivalRate ← median(arrival rate of all $i_{th}$ lows $\in$ MIN within
             each seasonal iteration)
12         low [i].timeStamp ← median(time stamp of all $i_{th}$ lows $\in$ MIN within
             each seasonal iteration)
           /* In seasonal iterations with more than peakNum lows,
              the $i_{th}$ low is selected, so that lows are evenly
              spaced throughout that seasonal iteration.           */
13         low [i] ← $\begin{pmatrix} \text{low}[i].arrivalRate \\ \text{low}[i].timeStamp \end{pmatrix}$;
14     for $i \leftarrow 0$ to peakNum $- 1$ do
15         interpolatingFunction ← DLIM Function starting at low [i], ending at
             peak [i]
16         rootSequence.append(interpolatingFunction)
```

Algorithm 5.2: Extracting the *Trend* part using s-DLIM.

```
   Data: duration: seasonal period duration,
   LIST: list of tuples $\vec{t} = \begin{pmatrix} arrivalRate \\ timeStamp \end{pmatrix}$,
1  MAX: list of local maxima in LIST,
2  trendSequence: root Sequence of all Trend segments
3  Function extractTrendPart()
4      largestPeakOffset ← offset of peak with largest arrival rate within a
         seasonal iteration
5      largestPeakArrivalRate ← arrival rate of peak with largest arrival rate
         within a seasonal iteration
6      iterations ← LIST.$lastTuple.timeStamp$/duration
7      for $i \leftarrow 0$ to iterations do
8          a ← nearestTuple(MAX, $i *$ duration + largestPeakOffset)
9          trendPoint [i] = a/largestPeakArrivalRate
10     trendSequence.append(constant trendPoint [0] with duration
         largestPeakOffset)
11     for $i \leftarrow 0$ to iterations do
12         interpolatingFunction ← DLIM Function starting at trendPoint [i],
             ending at trendPoint [i+1]
13         trendSequence.append(interpolatingFunction with duration duration)
14     trendSequence.append(constant trendPoint [iterations] with duration
         (duration − largestPeakOffset))
15 Function nearestTuple(tuple list L, time)
16     returns the tuple $\vec{t} = \begin{pmatrix} arrivalRate \\ timeStamp \end{pmatrix} \in \mathbf{L}$ with minimal
       $d \leftarrow |\mathbf{L}.timeStamp - time|$
```

This minimizes errors with trend calibration. The trend extraction calibrates the trend in a way that the model output arrival rate at the trend segment's beginning (or end) equals the trace's actual arrival rate at the respective point in time. The shape of the trend function (linear, exponential, logarithmic, sin) is predefined as a sin-shape, but can be changed on demand.

s-DLIM extracts a list of equal-length trend segments. These segments have a user defined duration that is a multiple of the seasonal period. Like the seasonal period, the trend segment duration is also selected using meta-knowledge about

Algorithm 5.3: Calculating the *Noise* distribution.

```
Data: LIST: list of read tuples $\vec{t} = \begin{pmatrix} arrivalRate \\ timeStamp \end{pmatrix}$;
1 Function calculatNoiseDistribution()
2     FILTERED_LIST ← applyGaussianFilter(LIST)
3     for $i \leftarrow 0$ to $|\text{LIST}| - 1$ do
4         difference[i] ← LIST [i].arrivalRate - FILTERED_LIST [i].arrivalRate
5     distribution ← normal distribution with mean(difference) and
      standardDistribution(difference)
```

the trace. The segments are then calibrated at their beginning and end to match the arrival rates in the trace. The s-DLIM *Trend Part* extraction is displayed in Algorithm 5.2.

Extracting the Burst Part Extracting *bursts* is a matter of finding the points in time at which significant outliers from the previously extracted *Seasonal* and *Trend* parts are observed in the trace. Once a burst is found, it is added to the root *Sequence* and then calibrated to match the arrival rate from the trace. Finding a burst requires the arrival rate in the trace to differ significantly from the predicted value based on the *Seasonal* and *Trend* parts. In order to eliminate false positives due to *Seasonal Parts* that are offset time wise, the *Seasonal Part* used for the reference model in the burst recognition activity differs from the actual extracted *Seasonal Part*. The difference is that the *Seasonal Part* used in the burst recognition activity does not interpolate between the peaks and lows of the original arrival rate trace. Instead it interpolates only between the peaks. This removes false positives due to seasonal periods that are slightly offset in time, however, it also eliminates bursts that do not exceed the current seasonal peak. This trade-off is considered acceptable, since time wise offset seasonal periods are commonly observed.

Extracting the Noise Part The *Noise Part* extraction consists of two steps: Noise reduction and the calculation of the noise distribution. The idea behind our approach is to first reduce the noise observed within the arrival rates contained in the trace, and then reconstruct the reduced noise by calculating the difference between the original trace and the filtered one. Having filtered the noise, the extraction of the *Seasonal Part*, *Trend Part*, and *Burst Part* are then performed on the filtered trace. This has a significant impact on the extraction accuracy of these parts, and thus on the overall accuracy of the extracted model instance.

Noise is reduced via the application of a one-dimensional Gaussian filter on the read arrival rates. A Gaussian filter has a kernel based on the Gaussian distribution, it thus has the following form (as defined by Blinchikoff and Zverev, 1986):

$$G(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}}$$

We choose the kernel width depending on the *Seasonal* period (duration of a single seasonal iteration) and the expected number of peaks (local maxima) within a *Seasonal* period:

$$KernelWidth = \frac{SeasonalPeriod}{ExpectedMax\#SeasonalPeaks}$$

A Gaussian filter's kernel width is defined as:

$$KernelWidth = 6 \cdot \sigma - 1$$

As a result, the standard deviation is:

$$\sigma = \frac{\frac{SeasonalPeriod}{ExpectedMax\#SeasonalPeaks}+1}{6}$$

The *Noise Part* is modeled as a normally distributed random variable. This variable is added to the DLIM instance's root *Sequence*. The normal distribution's mean and standard deviation are calculated as the mean and standard deviation of the differences between the filtered arrival rate trace. This is illustrated in Algorithm 5.3.

Summarizing, we address **RQ A.3** ("How to model and create realistic, varying load profiles for energy efficiency testing?") by introducing the DLIM modeling mechanism for load intensity profiles. DLIM is designed to enable modeling of seasonal patterns, trends, bursts, and noise within arrival rate profiles over time. We also provide an automated extraction mechanism, called s-DLIM, which extracts DLIM model instances from arrival rate traces. The models and extractors are implemented in the open-source LIMBO toolkit and have been extensively used in other works, for example, by Herbst et al., 2015, Groenda and Stier, 2015, and Becker et al., 2017a. We use DLIM to create load profiles for power benchmarking in Chapters 12 and 15. The work on the models and approaches of this section has been published, among others, at ACM TAAS (Kistowski et al., 2017a) and SEAMS 2015 (Kistowski et al., 2015d).

5.2 Hierarchical Load Distribution

The rising power consumption of servers and data centers over the last decades leads to an increasing pressure on hardware vendors to design systems with a high energy efficiency. Equally, software developers are tasked with the design and development of energy efficient applications.

Servers are rarely fully idle; but instead, they often serve requests that arrive at low frequencies leading to a typical load at a low-resource utilization level, as described by Barroso and Holzle, 2007. As a result, software can be designed to distribute its load on target systems in a manner that reduces power consumption. Typically, work is consolidated on as few systems as possible, allowing unused systems to enter power saving states. Designing such workload consolidation mechanisms and policies is challenging, as load intensity varies dynamically (see Section 5.1). Additionally, power consumption, performance, and energy efficiency characteristics change depending on workload, utilization, and machine hardware and software configuration. Finally, newer server processors, such as Intel's Haswell generation processors, feature advanced core-level power management mechanisms, which may warrant the application of workload consolidation on a processor core level.

Current load distribution approaches balance load based on generalized assumptions about the energy efficiency of servers. Most existing power consolidation approaches, such as the ones described by Pinheiro et al., 2001 and Chen et al., 2005, consolidate as much work as possible on each machine until a pre-configured threshold of performance degradation is met. These approaches assume optimal energy efficiency at full machine level utilization. Similar assumptions are made for many existing power management solutions. These approaches also do not attempt to fully maximize energy efficiency as they only minimize power consumption within specified performance constraints. They do not maximize the tradeoff between performance and power.

Hierarchical power management solutions explore the possibilities of triggering power saving states and mechanisms of hardware components. Raghavendra et al., 2008 and Verma et al., 2008a, for example, take the effects of CPU Dynamic Voltage and Frequency Scaling (DVFS) into account. They do not, however, make use of lower level load distribution, such as core-wise load distribution.

In this section, we address **RQ A.4**: "How to heterogeneously distribute load for different placements in energy efficiency testing?". We describe a modification of our power methodology to measure the energy efficiency of a variety of load distribution strategies on single and multi-node systems. Specifically, we modify the SERT implementation of the methodology. We specify the target load intensity (transaction arrival rate) and target workload on a client by client basis for each load level, with the clients being pinned to specific CPU cores. Using this approach we can emulate any load distribution policy for transactional workloads, allowing the evaluation of energy efficiency for homogeneous and heterogeneous workloads over different hardware and software configurations. In addition, we introduce load distribution policies, which we apply using this modification.

We describe the load distribution policies and our modifications to the measurement methodology. The analysis and policy evaluation is found in Chapter 11, Section 11.2. The goal of this part of the thesis is to gain insight in how different load balancing and unbalancing strategies affect overall power consumption and energy efficiency. The core contributions if this section and Section 11.2 are:

1. We introduce a modification to the workload placement of our measurement methodology to allow for hierarchical load distribution.

2. We investigate power consumption and energy efficiency of load balancing and consolidation policies at multiple utilization levels on a range of hardware configurations, including different architectures.

3. We explore the power consumption and energy efficiency of hierarchical load distribution on four levels: logical CPUs, physical CPUs, separate CPU sockets, and machine level load distribution for distributed multi-node systems.

4. We demonstrate that the effectiveness of load distribution policies depends on multiple factors, including workload type and load level for both homogeneous and heterogeneous workloads.

5. We combine existing strategies to create a new load distribution strategy using partial consolidation to target maximum energy efficiency across as many execution units as possible.

We evaluate the load distribution strategies in Chapter 11, Section 11.2 on a range of systems with increasing complexity: a one-socket system, two dual-socket systems of different architectures and two dual-socket systems running the workload in a multi-node configuration. For each of these systems, we test a variety of combinations applying different load distribution strategies on the different levels of the execution hierarchy (nodes, sockets, physical CPU cores, logical CPU cores).

We show that the selection of a single most energy-efficient strategy is only possible on smaller or older systems. For other systems, the most energy efficient load distribution strategy depends on workload type and load level. For some loads, the most efficient strategy is not always the commonly assumed one, for example, full load consolidation on a multi-node level can have a smaller energy efficiency than other load distributions. We also show that our new strategy can save up to 10.7% of power consumption on a single server node.

5.2.1 Load Distribution of Worklets

We modify our power rating methodology of Chapter 4 and its implementation in the SERT to specify the target load percentage on a client-by-client basis. The goal of this modification is the creation of target load distributions for load distribution testing. The methodology's clients (the basic execution schedulers, pinned to a single hardware thread) are modified to use separate different transaction inter-arrival time distributions, instead of using the a global distribution. Each client receives a separate load level specification, which defines a client-specific target load level for each global load level. Clients must use this specification to override the global load level the beginning of each interval in order to derive their own inter-arrival rate distribution. As clients are bound to a specific SMT unit (hardware thread) and physical CPU, this specification allows the stressing of each specific core. The operating system ensures that the client to CPU mapping remains identical over the separate experiments. Separate configurations can be deployed on each host (in our case, server), allowing different load distribution behaviors for each host.

We define the following policies for load distribution. Each policy can be applied individually on every level of the load distribution hierarchy (e.g., servers, sockets). This enables the creation of a strategy compositon by selecting one of these policies for each hierarchy level:

Balanced: Transaction counts are set to be equal for all clients, resulting in a balanced load across all systems.

Consolidated: Transactions are consolidated on as few clients as possible. As a result, all clients, with the exception of one, are either idle or at full utilization.

Energy Efficient Consolidation: This new strategy, introduced in this thesis, keeps as many clients as possible at the point of maximum energy efficiency. Specifically, we consolidate load on clients, with the upper load boundary for each client being the predetermined (calibrated) point of maximum energy efficiency for the given workload. Only once all clients have reached this point, do we start to increase client load with rising global load. At this point, the extra load is still consolidated on as few clients as possible. This strategy is identical to the consolidation strategy if maximum energy efficiency for the given workload is achieved at full utilization.

Any of these strategies can be applied at any level of the load distribution hierarchy. The following hierarchical execution units are targeted: full servers, CPU sockets, CPU cores, and logical CPU cores (also called hyperthreading or SMT units).

To enable the execution of heterogeneous workloads within our methodology, we also allow separate specification of the worklet executed on each *client* (see Section 4.2.1) in addition to the global default worklet. At the beginning of each scenario, the client will check if it has been assigned a worklet other than the global one. In this case, it replaces the global worklet with its own local worklet.

Specifying different worklets has an adverse effect on calibration. Recall that our methodology measures the maximum transaction rates of worklets on a SUT during calibration and uses this to calculate the target load level. However, co-locating different worklets on parallel resources requires an adaptation to calibration. This adapted calibration should, however, be conducted with the separate homogeneous workloads, in order to eliminate the influence of inter-worklet interference. We address this using separate calibration runs for each worklet using our default methodology. Each *client's* 100% target load level is then set to the calibration result of the corresponding *client* and workload during the calibration run. Note that we only run a single workload per *client*. Consequently, we have a maximum granularity of one workload per hardware thread. Workloads do not get mixed within the hardware threads. We also consider the possibility of turning affinity of *clients* off. In this case, clients are no longer bound to specific cores and may be redeployed by the operating system at runtime.

Summarizing, we address **RQ A.4** ("How to heterogeneously distribute load for different placements in energy efficiency testing?"). We describe a modification to our power measurement methodology that can be used to achieve uneven load distributions on different levels of a server system or cluster. We introduce load distribution policies to be tested using this modification, including one new policy that can be tested using this methodology adaptation. A thorough analysis using multiple different systems can be found in Chapter 11, Section 11.2. The load distribution strategies and their analysis have been published at the MASCOTS 2015 conference (Kistowski et al., 2015b).

5.3 Concluding Remarks

This chapter addressed **RQ A.3** ("How to model and create realistic, varying load profiles for energy efficiency testing?") and **RQ A.4** ("How to heterogeneously distribute load for different placements in energy efficiency testing?"). It addressed **RQ A.3** by introducing the DLIM modeling mechanism for load intensity profiles and **RQ A.4** by introducing hierarchical load distribution policies together with a modification to our power rating methodology that enables hierarchical load distribution.

DLIM is designed to enable modeling of seasonal patterns, trends, bursts, and noise within arrival rate profiles over time. We also provide an automated extraction mechanism, called s-DLIM, which extracts DLIM model instances from arrival rate traces. The models and extractors are implemented in the open-source LIMBO toolkit and have been extensively used in other works, for example, by Herbst et al., 2015, Groenda and Stier, 2015, and Becker et al., 2017a. LIMBO has also been endorsed as a peer-reviewed tool by the Standard Performance Evaluation Corporation, Research Group (SPEC RG). We use DLIM to create load profiles for power benchmarking in Chapters 12 and 15. The work on the models and approaches of this section has been published, among others, at ACM TAAS (Kistowski et al., 2017a) and SEAMS 2015 (Kistowski et al., 2015d).

Our modification to the power measurement methodology can be used to achieve uneven load distributions on different levels of a server system or cluster. We introduce load distribution policies, including one new policy that can be tested using this methodology adaptation. A thorough analysis using multiple different systems can be found in Chapter 11, Section 11.2. The load distribution strategies and their analysis have been published at the MASCOTS 2015 conference (Kistowski et al., 2015b).

Chapter 6

Advanced Workloads for Energy Efficiency Measurement

The accurate measurement of a server's power consumption when running realistic workloads, which go beyond our mini-workloads of Chapter 4, enables characterization of its energy efficiency and helps to make better provisioning and workload placement decisions. In addition, modern distributed and network applications offer complex performance behavior and many degrees of freedom regarding deployment and configuration, which affect energy efficiency. However, measuring energy efficiency and power consumption of server applications has become challenging as applications are often distributed or require work intensive configuration, setup, and specialized load drivers for reproducible testing. As a result, it may not be feasible to perform tests using the actual workload that is to be deployed on a server. Even then, existing production software is often inaccessible for researchers or closed off to instrumentation. Adding to this problem, existing distributed application testing and benchmarking frameworks are either designed for specific testing scenarios, or they do not offer the necessary degrees of freedom needed for energy efficiency research.

Addressing **RQ A.5**: "How to create reference workloads for complex test setups in distributed scenarios?", we introduce a workload creation approach and one distributed reference application. Firstly, we present a Performance Event Trigger Framework (PET) to create small-scale workloads that emulate the power consumption-relevant behavior of an application by deliberately triggering specific power relevant performance counter events. These workloads can then be easily deployed on a target server for fast and efficient power characterization. Secondly, we introduce TeaStore, a state-of-the-art microservice-based test and reference application. TeaStore offers services with different performance characteristics and many degrees of freedom regarding deployment and configuration to be used as a benchmarking framework for researchers. TeaStore allows evaluating performance modeling and resource management techniques; it also offers instrumented variants to enable extensive run-time analysis. We evaluate PET's workload creation approach and the TeaStore reference workload in Chapter 12. Specifically, we validate PET by approximating the power consumption behavior of four different workloads at multiple load levels. We show that our approach is capable of producing small-scale workloads that reflect the power consumption behavior of their reference applications over multiple load levels with a minimum error of less than 1%. Regarding TeaStore, we demonstrate its use in three contexts: performance modeling, cloud resource management, and energy efficiency analysis. Our experiments show that TeaStore can be used for evaluating novel approaches in these contexts and also motivates further research in the areas of performance modeling and resource management.

6.1 Performance Event Trigger Framework

Reducing the power consumption and improving the energy efficiency of servers requires the ability to accurately and reliably evaluate their efficiency when running the applications they are expected to execute during normal operation. Evaluation results can help to improve energy efficiency by enabling, among other things, better server purchasing decisions, better application placement decisions, and more efficient configuration of hardware or software.

However, performing accurate measurements with real world applications is difficult. Real world applications are often distributed and load is driven by external requests. This makes the setup and configuration complex, time consuming, and difficult, especially since it can be hard to evaluate individual software components on their respective servers in isolation.

Additionally, power measurements using external power meters require stable loads for a period of time in order to provide accurate measurements (Lange and Tricker, 2011). Achieving stable loads for real world applications requires specialized external load drivers and may also limit the configurations for which a test can be run. As a result, it is often not feasible to run tests using the actual application under consideration.

In this section, we address **RQ A.5**: "How to create reference workloads for complex test setups in distributed scenarios?" with a workload creation mechanism. Specifically, we introduce an approach to create small-scale workloads that emulate the power consumption-relevant behavior of large scale workloads by approximating their CPU performance counter profile. We analyze performance counters and their relation to power consumption to construct PET. It is designed to use performance counter measurements of third-party applications to construct a small-scale workload that emulates the third-party application's resource profile with the goal of reproducing its power consumption-relevant behavior.

The small-scale workload created by PET can be easily executed and measured on a target server platform. It is locally executable, not part of any larger distributed workload, requires no additional configuration and is designed to be a transactional workload that can be run at different load levels for server and workload characterization.

The major contributions of this section are:

- We characterize and analyze CPU performance counters with respect to their relevance for server power consumption modeling.
- We show that emulating a workload's performance counter profile can lead to a similar or even identical power profile.
- We propose a light-weight approach for emulating a third party workload's power consumption-relevant behavior, enabling easier benchmarking of large scale applications.

We validate our approach in Chapter 12.1 by applying it to four different applications. They range from small-scale test applications to industry standard benchmarks. We also evaluate the ability to accurately emulate the power consumption-relevant behavior of a virtual network function, which, considering its heavy I/O load, would intuitively be expected to be difficult to model using CPU performance events. We analyze the ability of the PET framework to accurately reproduce the original workloads' performance counters and power characteristics and compare multiple performance counter triggering options. We show that PET is capable of emulating the power consumption behavior of realistic workloads with a mean deviation down to 0.19 W (1%).

6.1.1 General PET Approach

The primary goal of our approach is the emulation of the power consumption-relevant behavior on a System under Test (SUT) by leveraging performance counters as a basis for the emulation. To this end, performance counters with relevance to power consumption will be identified by using a real world reference workload in the form of a deep packet inspection (DPI) firewall. This firewall is chosen as a reference as it produces significant CPU load, but is also bound to other hardware components due to its intensive use of network I/O.

The firewall is measured at ten load levels (using our measurement methodology from Chapter 4), ranging from 10% to 100% in 10% steps. During the measurement, performance counters and power consumption are recorded once per second and relevant counters are identified by correlating increase in counter events with the increase of power consumption over the different load levels using the Pearson correlation coefficient. A performance counter is considered relevant if its correlation with power consumption exceeds 0.8. It is assumed that some performance counters are collinear, causing other counters to be triggered as side-effects. Instead of not implementing counters causing side effects in PET, we chose to compensate side effects with composition mechanisms. Counters with dependencies or those requiring active management, such as CPU frequency, are excluded.

After the performance counters with relevance to power consumption are identified, we implement event triggers designed to generate the respective performance events on demand by executing suitable artificial workloads. The accuracy of a concrete event trigger can be influenced by implementation details. To account for this, several competing trigger implementations are analyzed for each event under consideration. The implementations capable of reaching the defined target counter values with the best accuracy are selected and incorporated into the PET framework. We also analyze potential side effects of our implementations by measuring the influence of event trigger implementations on other counters. Side effects must be considered given that some performance counters are not independent of each other. Recorded side effects are used to determine interactions/influences between performance events and their potential impact regarding the final power consumption emulation. We also consider background noise, which if not considered may lead to erroneous assumptions regarding the event trigger accuracy and side effects. Consequently, we measure the counter background noise on the SUT in an idle state for 120 s.

The system on which the measurements are performed is an HP DL20 Gen9 with 16 GiB of memory and a Xeon E3-1230v5 at 3.4 GHz. As Operating System (OS), we use a Debian Linux with kernel version 3.16.0-4-amd64. The background noise on the SUT using other OS's and software stacks must be measured once and made available to PET before deployment. Use of different hardware should not influence PET. We assume that each component uses a minimum amount of energy to be active and is consuming a specific amount of energy for each event occurrence. Assuming that the difference in active idle energy between different hardware components is negligibly small, influences can be reduced to how often a component is used. However, transferability of our approach might be limited if the target system's architecture differs significantly from the original one, for example, if it uses an additional CPU caching layer that did not exist on the original system. In addition, we assume that the systems under consideration share the same instruction set (e.g., we do not intend to test ARM systems with workloads created on an amd64 system).

PET creates the final workload for power consumption behavior using one of three composition mechanisms. The mechanisms differ by how they account for the performance counter side effects that are caused by the concrete trigger implementations and possible collinearities. Combinations of mechanisms are not used. The first *naive* method ignores side effects and operates only on the target event count recorded for the workload. The second composition mechanism *accumulates* the side effects as shown in Equation 6.1.

$$s_x = \sum_{i=1}^{n} e_{x,i} v_i \quad v_{s,x} = \begin{cases} v_x - s_x & \text{if } s_x \leq v_x \\ 0 & \text{if } s_x > v_x \end{cases} \tag{6.1}$$

We consider a given performance event x with the initial goal of computing the total number s_x of its occurrences as a side effect. x occurs $e_{x,i}$ times as a side-effect to each single triggering event of another target performance event i. We can calculate the total number of x's occurrences as a side event by multiplying it with the number of times v_i each ith event is triggered and then calculating the sum of these occurrences as a side effect over all i.

Next, we compute the number of times $v_{s,x}$ that we must trigger the performance event x considering that it is already being triggered s_x times as a side effect. To this end, we subtract s_x from v_x, which is the number of times we would have triggered x had it not been for the side effects. If s_x is larger than v_x, $v_{s,x}$ is set to zero, resulting in PET not deliberately triggering this event.

The third composition method uses *simulated annealing*, as described by Henderson et al., 2003. As previously, the goal is to calculate the total number of times we must trigger each event, considering that triggering it produces side-effects that affect the number of times we must trigger other events. We use simulated annealing, as performance counters can have non-linear dependencies to a degree that makes an analytic solution unfeasible. Simulated annealing is preferred to regular hill climbing as it is less prone to be caught in local minima or maxima. As energy function, a modified mean squared error function $f(\omega)$ for the solution ω is used, as shown in Equation 6.2. $\hat{\omega}_i$ is the target value for the i-th operation (the number of times the ith trigger must be run), ω_i the current value and $\omega_{s,i}$ represents the side effects imposed by the current configuration. It is calculated using Eq. 6.1, substituting ω_i for v_i.

$$f(\omega) = \frac{1}{n} \sum_{i=1}^{n} \left(\hat{\omega}_i - \omega_i - \omega_{s,i}\right)^2 \tag{6.2}$$

Finally, we prune event triggers from the composition. In cases where some performance counters reach values close to the system's background noise, it may be beneficial to remove them from the composition under the assumption that they have a low or negative impact on the system's power consumption. We make this assumption and prune the trigger if at least one of the following is observed considering the performance counter in question:

- it is overcounting by at least one order of magnitude, taking into account the median over all load levels,
- it features a correlation with power consumption of less than 0.9 in the reference measurement,
- it features a median reference value below the background noise.

Not pruning transactions may lead to inaccuracies in both the power consumption as well as the target performance in the power measurement methodology. The power methodology is based on reaching target load levels by injecting respective target throughputs, which can only be done if the sizes of transactions within the workload are sufficiently small, so that steady state can be reached during measurement. However, triggers can only trigger an integer number of performance events. PET's goal is to preserve the ratio between different performance events for each transaction within the workload. Consequently, triggering events that occur in very small numbers causes all other events to have to be triggered in relatively high numbers, in turn causing transactions to grow in size. This can cause instability in power measurements as large transactions are run at low frequencies, making it hard to ensure steady state during measurements.

6.1.2 Performance Counter Relevance to Power

In this first step of our approach, we analyze performance counters with respect to their relevance for modeling a system's power consumption profile. The goal of this analysis is the selection of the most power relevant counters for inclusion in the PET framework.

Counters that are implemented in PET and that have a high correlation with system power include L3 cache misses (*L3MISS*), L2 cache misses (*L2MISS*), memory reads (*READ*), memory writes (*WRITE*), instructions retired (*INST*), interrupts (*irq*), and context switches (*ctxt*). Despite lower correlation (< 0.8), *L3HIT*s and *L2HIT*s are selected, considering that cache misses do correlate with power consumption. If a memory access misses L2, it could either hit or miss L3. It therefore seems reasonable to include *L3HIT* as L2 misses could directly generate L3 hits if needed. As the cache's content needs to be controlled to reliably trigger cache misses, *L2HIT* is also included in PET despite a correlation of 0.727. Hardware interrupts (*irq*) and the number of context switches (*ctxt*) are supported by PET due to their high correlation with power consumption. The *user* counter is not selected despite a correlation of 0.874 as the test harness specifies the number of processes used for workload generation. The *softirq* counter is not selected despite its correlation of 0.993 because software interrupts are handled only after a system call or a hardware interrupt. As one of our main goals is the approximation of large scale workloads, network I/O, producing hardware interrupts with a correlation of 0.997 is deemed sufficient. The *processes* counter has a good correlation (-0.984) but is not selected as the implementation for context switches (*ctxt*) creates threads which will interfere with this counter. Correlation on the remaining events is considered too low to be relevant or the event has a dependency on an already implemented counter.

6.1.3 Event Trigger Implementation

Triggers to generate performance events can be implemented in multiple ways. The following paragraphs describe our different implementations for triggering counters.

6.1.3.1 Cache and Memory Access

To trigger cache hits and misses, an array of at least twice the L3 cache size and up to 2048 MiB is used. For memory accesses (L3 miss), a pointer is cycled through the array. L3 hits and L2 misses are generated by a pointer cycling through a subarray of four times the size of the L2 cache. L2 hits are generated by accessing the same pointer used for L2 misses, but without moving it prior to a hit access. As the CPU will always move data with the size of a cache line, the array is traversed in a step size of 2, 4 and 6 times the cache line to determine if the step size has an influence on accuracy. A random number is added to each step size to account for the influence of hardware prefetching. We implement the cache and memory event triggers using three different instruction sets, each. We implement the triggers using C, non-temporal SIMD intrinsics, and Assembler (ASM). In addition, read, write and copy functionality are implemented and tested with process owned memory, shared memory and mapped kernel memory marked as uncachable via the page attribute table. In case of the uncachable memory, only fixed memory locations are used. We evaluate the effects of the different implementations in the first part of PET's evaluation in Section 12.1.1.

6.1.3.2 Instructions Retired

Retired instructions are implemented by causing instructions to be executed on the CPU. For efficiency, we trigger multiple instructions per performance trigger execution by looping over an instruction adding a constant value to a temporary variable. As a result, the corresponding counter is incremented by the amount of assembler instructions executed in this loop for each call to the event trigger.

6.1.3.3 Context Switches

To trigger a context switch, a new process or thread has to be created. Another option would be to suspend the currently running thread so the OS can switch to already existing processes or threads. Suspending the workload to trigger context switches has two disadvantages. The first reason to not further investigate is the scheduling interval which limits the amount of context switches in a given time-frame. Secondly, the possibility exists that no thread has to be switched in during suspension, making this solution random and unreliable. Therefore a thread is created with an empty workload that is switched in and immediately joined to be switched out. Consequently, context switches are always triggered in groups of two.

6.1.3.4 Interrupts

We use the Boost open source library to program the advanced programmable interrupt controller to throw a local timer interrupt after a deadline is reached. The deadline is set to the minimum value after which interrupts can be observed.

Summarizing, PET addresses **RQ A.5** ("How to create reference workloads for complex test setups in distributed scenarios?") in regards to creation of workloads for specialized resource (CPU) profiles. To create PET, we determined relevant performance counters to trigger in order to emulate the power profile of a real-world server application. PET implements these triggers with several alternate trigger implementations. We evaluate PET's implementation and performance counter set in Chapter 12.1. The work on PET has been published at the MASCOTS 2017 conference (Schmitt et al., 2017a).

6.2 TeaStore: A Micro-Service Reference Application

Modern distributed component and/or service-based applications have complex performance characteristics, as the constituent services feature different bottlenecks that may even change over time, depending on the usage profile. However, these applications also offer many degrees of freedom, which are intended to help deal with these challenges. They can be deployed in various ways and configured using different settings and software stacks. These degrees of freedom can be used at design-time, deployment-time, and at run-time for continuous system optimization. Current research employs many methods of analysis, modeling, and optimization that utilize these degrees of freedom at different points of the software life-cycle to tackle the challenging performance behavior (Ilyushkin et al., 2017; Becker et al., 2009). More generally, the goal of such research is the improvement of a running system's non-functional properties and may include dependability (Lee and Iyer, 1995; Littlewood and Strigini, 1995) or energy efficiency (Beloglazov et al., 2012; Basmadjian et al., 2011).

Verifying, comparing, and evaluating the results of such research is difficult. To enable practical evaluation, researchers need a software application (1) that they can deploy as reference and (2) that offers realistic degrees of freedom. The reference application must also feature sufficient complexity regarding performance behavior to warrant optimizing it in the first place. Finding such an application and performing the necessary experiments is often difficult. The software in question should be open source, available for instrumentation, and should produce results that enable analysis and comparison of research findings, all while being indicative of how the evaluated research would affect applications in production use.

Real world distributed software is usually proprietary and cannot be used for experimentation. It is often inaccessible and lacks the potential for instrumentation. In addition, evaluations conducted using such software are difficult to reproduce and compare, as the software used remains inaccessible for other researchers. Existing test and reference software, on the other hand, is usually created for specific testing scenarios (Happe et al., 2011). It is often designed specifically for evaluating a single contribution, which makes comparisons difficult. Other existing and broadly used test software does not offer the necessary degrees of freedom and is often manually adapted (Willnecker et al., 2015b). Some of the most widely used test and reference applications, such as RUBiS (*RUBiS User's Manual* 2008) or Dell DVD Store (Dell, Inc., 2011), are outdated and therefore not representative of modern real world applications. Newer distributed reference applications, such as Sock Shop (Weaveworks Inc., 2017), are built for maximum scalability and consistent performance and do not pose the performance challenges that current research aims at.

In this section, we address **RQ A.5**: "How to create reference workloads for complex test setups in distributed scenarios?" with a micro-service reference application. Specifically, we introduce TeaStore[1][2], a micro-services-based test and reference application that can be used as a benchmarking framework by researchers. It is designed to provide multiple degrees of freedom that researchers can vary when evaluating their work. TeaStore consists of five different services, each featuring unique performance characteristics and bottlenecks. Due to these varying performance characteristics and its distributed nature, TeaStore may also be used as a software for testing and evaluation of software performance models and model extraction techniques. It is designed to be scalable and to support both distributed and local deployments. In addition, its architecture supports run-time scalability as services and service instances can be added, removed, and replicated at run-time. The services' different resource usage profiles enable performance and efficiency optimization with non-trivial service placement and resource provisioning decisions.

To summarize, we envision the use of TeaStore in the following research areas, among others:

1. Evaluation of software performance modeling approaches, model extractors, and model learners.

2. Evaluation of run-time software performance management techniques such as auto-scaling and service placement algorithms.

3. Evaluation of software energy efficiency, power models, and optimization techniques.

We demonstrate the applicability of TeaStore as a test application and benchmarking framework in Chapter 12, Section 12.2 by using it as a reference software in experiments that show its applicability in each of the three motivating research areas. We show that TeaStore can be used as a reference scenario for performance modeling by creating a simple performance model to predict the application performance for different deployment options. This example model also illustrates the limitations of simplified software performance models for predicting the performance of complex distributed applications, while highlighting open research challenges. In addition, we show TeaStore's

[1]TeaStore on GitHub: `https://github.com/DescartesResearch/TeaStore/`
[2]TeaStore on DockerHub: `https://hub.docker.com/u/descartesresearch/`

elastic run-time scalability by running it using a state-of-the-art baseline auto-scaler. We show that the baseline auto-scaler can scale TeaStore elastically at run-time, while also demonstrating the limitations of conventional auto-scalers for complex applications. Finally, we examine the energy efficiency and power consumption when scaling TeaStore over multiple physical hosts. We show that distribution and placement decisions lead to different power and energy efficiency behavior, which can be used to evaluate energy optimization methods.

6.2.1 TeaStore Description

TeaStore is an online store for tea and tea related utilities. Its products are sorted into categories. For online shopping, the store supports an overview of products including preview images for each category and featuring a configurable number of products per page. All pages of TeaStore show an overview header bar and include the category menu and page footer. As main content, it shows the products for the selected category, including shortened product information and the preview image. Depending on the number of products shown per page, the user has the option to cycle through multiple pages of the category view.

Each product can be viewed on a separate product page containing detailed information, a large image, and advertisements for other store items. Besides the regular header, footer, and category list, this page includes a detailed image of the product (provided by the Image Provider Service), a description, and price. The page also contains an advertisement panel suggesting three products that the user might be interested in. The advertised products are provided by the Recommender Service and are selected depending on the viewed product.

All products can be placed in a shopping cart and users can proceed to order the current shopping cart. The user can choose to modify the shopping cart at any time. The shopping cart page lists all products currently included in the cart together with some product information and the quantity. The shopping cart view also displays product advertisements, which are, again, provided by the separate Recommender service and selected depending on the shopping cart's contents.

To order, the user must supply personal information about the billing address and payment details. After confirmation by the user, the current shopping cart is stored in the order history database through the Persistence service. The store also supports user authentication and login. Registered users can view their order history after login.

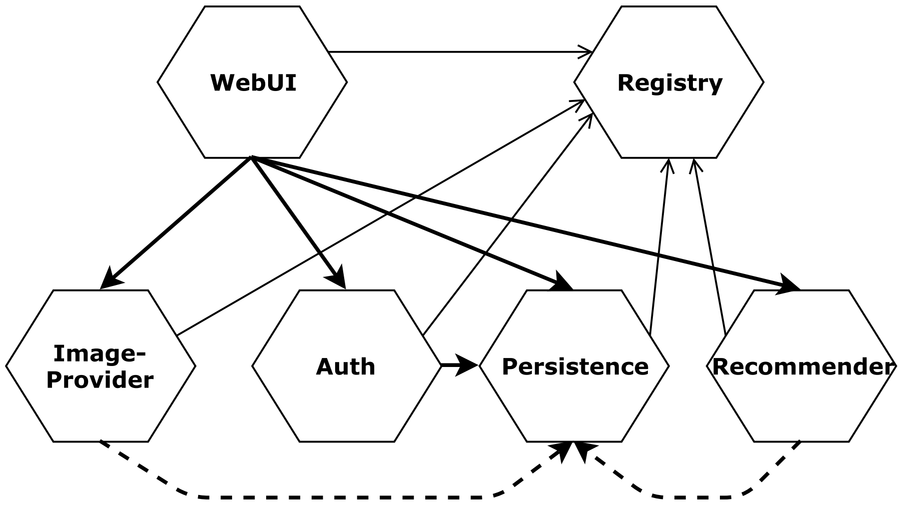


Figure 6.1: TeaStore architecture.

In addition to regular operations, TeaStore's user interface provides an overview of all running service instances and an option to regenerate the database. In case a specific database setup or size is necessary, it can be regenerated with user defined parameters. These include the number of categories, number of products per category, number of users, and maximum orders per user history. The service overview and database regeneration are not intended to be run during an experiment run, but separately on experiment setup.

All functionality is contained within the five primary micro-services and the Registry service.

6.2.2 Architecture

TeaStore consists of five distinct services and a Registry service as shown in Figure 6.1. All services communicate with the Registry. Additionally, the WebUI service issues calls to the Image Provider, Authentication, Persistence and Recommender services.

The Image Provider and Recommender service both connect to a provided interface at the Persistence service. However, this is only necessary on startup (dashed lines). The Image Provider must generate an image for each product, whereas the Recommender needs the current order history as training data. Once running, only the Authentication and the WebUI access, modify, and create data using the Persistence service.

All services communicate via Representational State Transfer (REST) calls, as REST has established itself as the de-facto industry standard in the micro-service domain. The services are deployed as web-services on Apache Tomcat. Yet, the services can be deployed on any Java application server able to run web-services packaged as `war` files. As an alternative to deploying the `war` files, we provide convenient Docker images containing the entire Tomcat stack. Each service is packaged in its own `war` file or Docker image.

TeaStore uses the client-side load balancer Ribbon[3] to allow replication of instances of one service type. Ribbon distributes REST calls among running instances of a service. Instead of using Netflix Eureka[4], TeaStore uses its own registry that supplies service instances with target instances of a specified target specific service type. To enable this, all running instances register and unregister at the registry, which can be queried for all running instances of a service. This allows for dynamic addition and removal of service instances during run-time. Each service also sends heartbeats to the registry. In case a service is overloaded or crashed and therefore fails to send heartbeat messages, it is removed from the list of available instances. Subsequently, it will not receive further requests from other services. This mechanism ensures good error recovery and minimizes the amount of requests sent to unavailable service instances that would otherwise generate request timeouts.

As TeaStore is primarily a benchmarking and testing application, it is open source and suitable for instrumentation using available monitoring solutions. Pre-instrumented Docker images for each service that include the Kieker[5] monitoring application (Hoorn et al., 2012, 2009) as well as a central trace repository service, are already available. We use Kieker, as it requires no source code instrumentation and the instrumentation can be adapated at runtime. However, as TeaStore is open source, other monitoring solutions, such as Prometheus [6] or Logstash [7] can also be utilized.

Generally, all requests to the WebUI by a user or load generator are handled in a similar fashion. The WebUI always retrieves information from the Persistence service. If all information is available, images for presentation are fetched from the Image Provider and embedded into the page. Finally a Java Server Page (JSP) is compiled and returned. This behavior ensures that even non-graphical browsers and simple load generators that otherwise would not fetch

[3]Netflix Ribbon: `https://github.com/Netflix/ribbon`
[4]Netflix Eureka: `https://github.com/Netflix/eureka`
[5]Kieker APM: `http://kieker-monitoring.net/`
[6]Prometheus: `https://prometheus.io/`
[7]Logstash: `https://www.elastic.co/products/logstash`

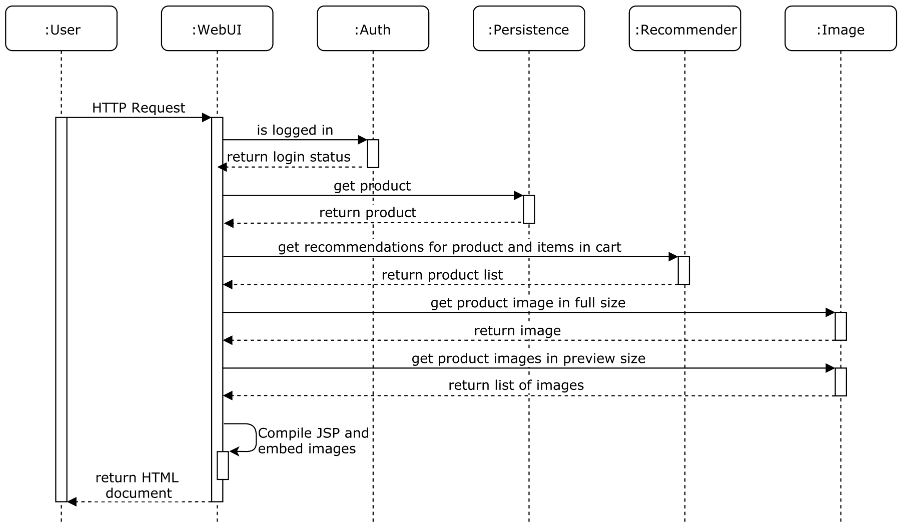


Figure 6.2: Service calls when requesting product page.

images from a regular site cause image I/O in TeaStore, ensuring comparability regardless of the load generation method.

Figure 6.2 shows the service calls for a user request for a product information page. After receiving the Hypertext Transfer Protocol (HTTP) request, the WebUI checks the user's login status by calling the Auth service. Next, it queries the Persistence service for the corresponding product information, based on a unique identifier. Afterwards, the WebUI requests advertisement options for the current product from the Recommender service, which generates a recommendation based on the learned historical order data. The call to the Recommender service takes the current login status into account. Specifically, a logged in user receives personalized recommendations, whereas an anonymous user is served recommendations based on general item popularity. Having received all product information, the WebUI queries the image provider to supply a full size image of the product shown in detail and preview images for the recommendations. The image data is embedded in the Hypertext Markup Language (HTML) response as base-64 encoded strings.

6.2.3 Services

TeaStore consists of five services, in addition to a registry necessary for service discovery and load balancing. In case monitoring is enabled, a trace repository service can be used to collect the monitoring traces centrally.

6.2.3.1 WebUI

This service provides the user interface, compiling and serving JSPs. All data, available categories, their products, product recommendations and images, are retrieved from the Image Provider and Persistence service instances. The WebUI service performs preliminary validity checking on user inputs before passing the inputs to the Persistence service. The WebUI focuses purely on presentation and web front-end operations. However, the performance of the WebUI depends on the page that has to be rendered as each page contains at least one picture in different formats.

6.2.3.2 Image Provider

The Image Provider serves images of different image sizes to the WebUI when being queried. It optimizes image sizes depending on the target size in the presentation view. The Image Provider uses an internal cache and returns the image with the target size from the cache if available. If the image is not available for this size, the image provider uses the largest available image for the category or product, scales it to the target size, and enters it into the cache. It uses a least frequently used cache, reducing resource demand on frequently accessed data. Through the 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.

6.2.3.3 Authentication

This service is responsible for the verification of both the login and the session data of a user. The session data is validated using SHA-512 hashes. For login verification, the BCrypt algorithm is used. The session data includes information about the current shopping cart content, the user's login status and old orders. Thus, the performance of the hashing for the session data depends on number of articles in the cart and number of old orders. Furthermore, as all session data is passed to the client, the Authentication itself manages to remain stateless and does not need additional information on startup.

6.2.3.4 Recommender

The Recommender service uses a rating algorithm to recommend products for the user to purchase. The recommendations are based on items other customers bought, on the products in a user's current shopping cart, and on the product the user is viewing at the time. The initial Recommender instance usually uses the automatically generated data-set, as provided by the Persistence service at initial startup, for training. Any additional Recommender instance queries existing Recommender service instances for their training data-set and uses only those purchases for training. This way, all Recommenders stay coherent, recommending identical products for the same input. In addition, using identical training input also ensures that different instances of the Recommender service exhibit the same performance characteristics, which is important for many benchmarking and modeling contexts. The Recommender service queries the Persistence service only once on startup.

For recommending, different algorithm implementations exhibiting different performance behaviors are available. Next to a fallback algorithm based on overall item-popularity, two variants of Slope One (Lemire and Maclachlan, 2005) and one order-based nearest-neighbor approach are currently implemented. One variant of Slope One calculates the predicted rating matrix beforehand and keeps it in the memory (memory-intensive), wheras the other one calculates every row if needed, but discards all results after each recommendation step (CPU-intensive).

6.2.3.5 Persistence

The Persistence service provides access and caching for the store's relational database. Products, their categories, purchases, and registered store users are stored in a relational Structured Query Language (SQL) database. The Persistence service uses caching to decrease response times and to reduce the load on the database itself for improved scalability. The cache is kept coherent across multiple Persistence service instances. We use the EclipseLink JPA implementation as a black-box cache. All data inside the database itself is generated at the first start of the initial persistence instance. By using a persistence service in separation from the actual database, we improve scalability by providing a replicable caching service. However, the performance of the database accesses depends on the content in the database that is changed or can be repopulated during the operation of the store.

6.2.3.6 Registry

The registry is not part of the TeaStore application under test but is a necessary support service. It keeps track of all running service instances, their IP addresses or host names and port numbers under which the services are available. All service instances send keep-alive messages to the registry after registration.

If a service unregisters or no keep-alive message is received within a fixed time frame, the service is removed from the list of available service instances. All services can query the list of service instances for a specified service type in order to distribute their outgoing requests between running target instances.

TraceRepository The services are configured with optional Kieker monitoring[8] (Hoorn et al., 2012, 2009). With monitoring enabled, each service instance collects information about utilization, response times and call paths. Collecting these monitoring traces manually is only feasible for small deployments, however, TeaStore deployments can include hundreds of service instances. Therefore, we offer a central trace repository, which consists of an AMQP server coupled with a graphical web interface. All service instances send their logs to the AMQP server. The web interface collects them and makes them available for download. The trace repository does not only reduce the effort required to acquire the monitoring traces, but also enables online analysis such as online resource demand estimation (Spinner et al., 2014). Kieker traces are also available for use with tools other than Kieker's own tooling, as they can be automatically transformed to Open Execution Trace Exchange (OPEN.xtrace) traces, an open source trace format enabling interoperability between software performance engineering approaches (Okanović et al., 2016).

Summarizing, TeaStore addresses **RQ A.5** ("How to create workloads for specialized tests, such as distributed tests or tests with certain resource usage profiles?") regarding tests of distributed applications. It is a micro-service testing and reference application that can be used for evaluation of research in several domains, including energy efficiency of distributed service placements, consisting of five primary services. We show TeaStore's use in research, using three use-cases in Chapter 12, Section 12.2. TeaStore has been published at the MASCOTS 2018 conference (Kistowski et al., 2018b) and is available as an open-source application on GitHub and DockerHub.

6.3 Concluding Remarks

This chapter addressed **RQ A.5** ("How to create workloads for specialized tests, such as distributed tests or tests with certain resource usage profiles?"). In regards to creation of workloads for specialized resource (CPU) profiles this question is addressed with the Performance Event Trigger Framework (PET). We describe PET's approach and how to find relevant performance

[8]Kieker setup: `https://github.com/DescartesResearch/TeaStore/wiki/Testing-and-Benchmarking`

counters to trigger in order to emulate the power profile of a real-world server application. PET's implementation and performance counter set are evaluated in Chapter 12.1. The work on PET has been published at the MASCOTS 2017 conference (Schmitt et al., 2017a).

Regarding tests of distributed applications, we address **RQ A.5** by introducing TeaStore, a testing and reference application that can be used for evaluation of research in several domains, including energy efficiency of distributed service placements. TeaStore is a micro-service application that consists of five primary services. We show TeaStore's use in research, using three use-cases in Chapter 12, Section 12.2. TeaStore has been published at the MASCOTS 2018 conference (Kistowski et al., 2018b) and is available as an open-source application on GitHub and DockerHub.

Part III

Modeling the Energy Efficiency of Servers

Chapter 7

Interpolating Power Consumption

Performance and power scale non-linearly with device utilization, making characterization and prediction of energy efficiency at a given load level a challenging issue. A common approach to address this problem is the creation of power or performance state tables for a pre-measured subset of all possible system states. Approaches to determine performance and power for a state not included in the measured subset use simple estimation methods, such as nearest neighbor interpolation, or define state switching rules. This leads to a loss in accuracy regarding estimation of these unmeasured system states. In this chapter, we address **RQ B.1** of `Goal B`: "How to obtain information about load level power consumption for a load level not covered by the measurements made when applying the methodology?". We compare different interpolation functions and automatically configure and select functions for a given domain or measurement set. We evaluate our approach in Chapter 13 by comparing interpolation of measurement data subsets against power and performance measurements on a commodity server. We show that for non-extrapolating models interpolation is significantly more accurate than regression, with our automatically configured interpolation function improving modeling accuracy up to 43.6%.

7.1 Introduction

Computational devices have to run at a great range of device utilization levels with power and performance scaling non-linearly over the different load levels, as power saving mechanisms, such as Dynamic Voltage and Frequency Scaling (DVFS) are being used. Being able to characterize and predict the power consumption and performance at a target load level improves the quality of management decisions and helps to improve overall energy efficiency. A common and highly accurate method for the characterization of power and performance of a workload on a target system at a given load level is the measurement of said characteristics. Measurement results are then stored in a table for the set of different measured load levels and used for subsequent decisions.

These stored measurement results contain power and performance values for a subset of possible system states. Determining the power and performance of the system at other states remains challenging, yet necessary for accurate management decisions. Currently, several approaches to estimate power consumption at these states exits: Some tools, such as the ones by Apte and Doshi, 2014 and Verma et al., 2008a only consider the existing pre-defined states and determine the current state either by nearest neighbor interpolation or through other rule-based mechanisms. Another approach is the training of models based on measured data. Models range from simple models, such as the linear power model described by Tu et al., 2013 or Bohrer et al., 2002 and variations thereof (Fan et al., 2007), to more complex regression models, which also take additional system properties into account (Nagasaka et al., 2010; Lewis et al., 2008; Lee and Brooks, 2006). Trained models can be used for the prediction of non-measured power states on the original system. They also offer the advantage of allowing the transfer of knowledge to other systems beyond the training system. To do so, they only require a re-training of a smaller subset of model parameters. The major drawback to the modeling approach is the introduction of the model's error. Regression models, in particular, even introduce errors when modeling the pre-measured data points.

In comparison to approximation, interpolation increases prediction accuracy, as it does not sacrifice or approximate any of the pre-measured results. A great number of different interpolation methods exist. Depending on the system and the power or performance metric under observation, a different interpolation method may be optimal. In addition, some interpolation methods can be configured with varying degrees of freedom.

This chapter presents a method for automated selection of interpolation and configuration strategies for performance and power characterization with the goal of minimizing prediction errors for unmeasured performance and power. The major contributions of this chapter are as follows:

1. We present an approach for automated selection of an interpolation strategy for a given set of performance or power measurements.

2. We introduce a method for configuration of interpolation strategies.

3. We propose a composition of piece-wise polynomial interpolators of varying degrees for the interpolation of a system's power over utilization function.

The methodology presented in this chapter has been implemented in a freely available open-source library[1].

We evaluate our method in Chapter 13 based on power and performance measurements using ten of the workloads of the SERT implementation of our power rating methodology in Chapter 4. Measurements are performed for 100 load levels per workload. Prediction accuracy is then evaluated regarding the methods' ability to predict power and performance for all load levels based on a sub-set of the measurements. We show that for bounded problem spaces interpolation features superior accuracy compared to regression. On our set of measurement data, our automated interpolation method configuration and selection approach improves modeling accuracy by 43.6% in comparison to regression if additional reference data is available and by 31.4% if it is not.

7.2 Interpolation Functions

Hoschek and Lasser, 1993 define scattered data interpolation as the reconstruction of a continuous function $f(x)$ from n different sample points $\{(x_1, f_1), (x_2, f_2), ..., (x_n, f_n)\}$. In this thesis, we consider 2-dimensional functions, where $f(x)$ is our power or performance metric and the input metric x the corresponding system metric (usually utilization). For these purposes, we consider the following interpolation functions:

- **Nearest Neighbor Interpolation**: $f(x) = f(x_i)$ with $x_i \in \{x_1, ..., x_n\}$ being the nearest neighbor to x, meaning that $\forall x_j \in \{x_1, ..., x_n\} : |x - x_i| \leq |x - x_j|$.

- **Linear Interpolation**: Given the two nearest neighbors of x, x_i and x_{i+1}, with $x_i \leq x$ and $x_{i+1} > x$ (Eq. 7.1):

$$f(x) = f(x_i) + (f(x_{i+1}) - f(x_i))\frac{x - x_i}{x_{i+1} - x_i} \tag{7.1}$$

- **Shepard Interpolation** (Shepard, 1968): $f(x) = f(x_i)$ if $x = x_i$, otherwise (Eq. 7.2):

$$f(x) = \frac{\sum_{i=1}^{n} w_i(x) f(x_i)}{\sum_{i=1}^{n} w_i(x)} \tag{7.2}$$

with $w_i(x) = \frac{1}{|x - x_i|^p}$.

[1]Library: `http://descartes.tools/interpolation`

Parameter p is freely configurable and usually selected based on experience. We select it using our auto-configuration approach.

- **Polynomial Interpolation**: $f(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_0 x^0$, with the coefficients a_i being the solution of the following system of equations (Eq. 7.3):

$$\begin{pmatrix} x_0^n & x_0^{n-1} & \cdots & x_0^0 \\ x_1^n & x_1^{n-1} & \cdots & x_1^0 \\ \vdots & \vdots & \ddots & \vdots \\ x_n^n & x_n^{n-1} & \cdots & x_n^0 \end{pmatrix} \begin{pmatrix} a_n \\ a_{n-q} \\ \vdots \\ a_0 \end{pmatrix} = \begin{pmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix} \tag{7.3}$$

  Polynomial interpolation of large datasets is prone to oscillation, also known as Runge's Phenomenon, see Shen et al., 2012. To avoid this, we do not only perform polynomial interpolation on the entire dataset. We also split the set into subsets of size m and interpolate these subsets using polynomial functions of degree $m - 1$. The composite function of these piece-wise polynomials constitutes the interpolation function for the entire dataset.

- **Spline Interpolation**: A special type of piece-wise polynomial interpolation, which guarantees that the overall function remains continuous in all interpolated data points. The contributions of this thesis use the cubic spline implementation of the Apache Commons Math library, described by Burden and Faires, 1989.

7.3 Determining Interpolation Accuracy

We determine the accuracy of an interpolation function using one of these two methods: Interpolation against a reference dataset or cross validation. For single cases, the latter of these two options is the more common one, as interpolation accuracy improves with additional interpolated data. As a result, all available data is included in the interpolated data, leaving no additional data for referencing. However, a reference dataset is most useful when determining the optimal interpolation method for a given problem domain (in our case, a class of power or performance functions). In such a case, the optimal interpolation function for the entire class of problems can be determined using an (independent) reference dataset, e.g., a reference dataset measured on a separate machine.

If a reference dataset R containing the tuples (x_i, y_i) is available, we calculate a set of absolute errors E with $E = \{e_1, ..., e_n\}$ for interpolation function f as in Eq. 7.4:

$$\forall (x_i, y_i) \in R : e_i = |f(x_i) - y_i| \tag{7.4}$$

If no reference dataset exists, we calculate the set of absolute errors E via cross-validation on the interpolated dataset I containing the tuples (x_i, y_i), with $|I| = n$. We create a set of cross-validation-sets V_i as shown in Eq. 7.5:

$$\forall i \in \{2, ..., n-1\} : V_i = I \setminus \{(x_i, y_i)\} \tag{7.5}$$

With f_i being the interpolation function constructed using V_i, we calculate the cross-validation errors as in Eq. 7.6:

$$\forall i \in \{2, ..., n-1\} : e_{i-1} = |f_i(x_i) - y_i| \tag{7.6}$$

Our implementation allows the metric for calculation of the final aggregate error of the error set E to be passed using a functional expression. In this contribution, we use the arithmetic mean (MAE) and median.

7.4 Interpolation Selection and Configuration

We allow selection of the best interpolation function for the problem's domain using an independent reference dataset containing a larger set of data points, which describes a similar problem as the dataset to be interpolated. This is usually the case if both datasets describe power per load level measurements, yet they were measured for different workloads on different machines. As the function to be interpolated is of the same type, we can use the independent reference dataset for interpolation method and configuration space detection. In such a case, we create a subset of the same size as the set to be interpolated, containing datapoints with the closest possible input values to the input values of the independent reference dataset (x-axis values in a 2D function). Then we select the best configuration and interpolation methods for this subset by comparing the aggregate modeling error of the potential interpolation methods. The function with the minimum aggregate error, calculated using the selected aggregate error metric (see Section 7.3), is selected as the final interpolation function. For functions with a configurable parameter (degree of freedom), this parameter must be auto-configured first. Finally, we transfer the selected method and configuration to the actual set to be interpolated.

In some cases, reference data is not available and selection of a single pre-configured interpolation method is not possible, either due to a lack of sufficient domain knowledge or because of the problem domain's nature. In this case, we select the best interpolation function for a given dataset by calculating the cross-validation error. As specified in Section 7.3, we compute different cross-validation datasets, each with one data point removed. At least one data point must be removed for cross-validation, as the self-prediction error of an interpolation function is always 0. Consequently, cross-validation using the full dataset is not possible. We evaluate the interpolation method's ability to predict the missing data point for each of the cross-validation datasets. The function with the minimum aggregate error over all cross-validation datasets is selected as the final interpolation function.

7.4.1 Interpolation Function Configuration

Among the existing interpolation functions used in this chapter, two function types feature a configurable degree of freedom. Shepard interpolation features the parameter p, which is usually configured using a positive single digit integer number, yet is theoretically infinite. Piece-wise polynomial interpolation, on the other hand, can only range between polynomials of degree 1 and polynomials of degree $dataset\_size - 1$.

We select the final configuration parameter using a hill-climbing approach, as the parameters have a specified minimum value in all of our cases. To apply hill-climbing, each parameter must have an initial parameter instance p_0 and a function h so that $p_i = h(p_{i-1})$. With $f(p_i)$ being the parametrized interpolation function and $e(f(p_i))$ being its error metric and using an initial infinite error, we iterate over the parameters p_i in an ascending order (using h) as long as $e(f(p_{i+1})) \leq e(f(p_i))$.

7.4.2 Break Detection for Polynomial Interpolation

To improve interpolation accuracy for performance and power measurements, we introduce a new approach to parametrization of piece-wise polynomials. This approach is designed to minimize the interpolation error due to state changes caused by device power management. These state changes cause non-continuous behavior in a power or performance function. Consequently, it pays to introduce breaks at these points when interpolating polynomials.

Breakpoints are detected at the data points featuring the greatest difference between their power/performance and their successor's power/performance. Meaning that given a list $(y_0, \ldots y_m)$ of data points to interpolate and given a set of n breakpoint indices B, with $|B| = n$, $n < k$, the following has to hold true: $\forall i \in B : |y_{i+1} - y_i| > |y_{j+1} - y_j|$, with $j \notin B$.

We use these break points for piece-wise polynomial interpolation by interpolating polynomials over the subsets defined within breakpoint boundaries.

An example of such a state change as mentioned above is the activation of a processor's turbo mechanism, which leads to a significant sudden increase in performance and power draw at high utilization. The effect is reproducible on newer processors and can be observed at the sudden increase of power consumption around the 80% load level mark in Fig. 7.1, which shows the power consumption for the seven CPU workloads of the SPEC SERT suite over a range of target load levels. Each of the workloads was sampled at 50 load levels ranging from 2% load to 100% device load.

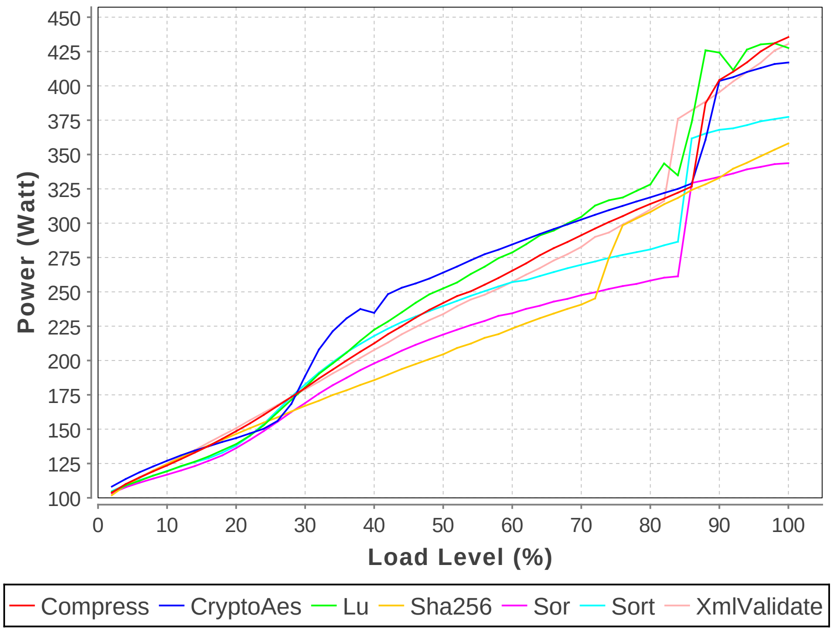


Figure 7.1: Power consumption of SERT CPU worklets on Fujitsu PRIMERGY RX300S7 system.

The amount of breakpoints to be applied remains a freely configurable parameter. It can be determined and set using the hill-climbing approach introduced in Section 7.4.1, making the dynamically split polynomial interpolation the third function with a configurable degree of freedom in this paper.

7.5 Concluding Remarks

This chapter addressed **RQ B.1** ("How to obtain information about load level power consumption for a load level not covered by the measurements made when applying the methodology?") using several interpolation methods for power and energy efficiency. It presents an automated selection and configuration mechanism that is either able to select good interpolation methods based on reference-datasets or based on no additional information using cross-validation. We also present a break-point detection mechanism for piece-wise polynomial interpolation using varying degrees. The contributions of this chapter have been implemented in an open-source interpolation library. It is used as part of our work in Chapter 9. They have also been published at the VALUETOOLS 2015 conference (Kistowski and Kounev, 2015).

Chapter 8
Offline Prediction of Power Consumption

Energy efficiency and power consumption of data centers can be improved through intelligent placement of workloads on specific servers. Placing specific target applications on servers on which they run in an energy-efficient manner is a first step in this regard. To enable this placement, virtualization is commonly employed as it allows for dynamic reallocation of work and abstraction from the concrete server hardware. However, estimating the power consumption of an application on a not-yet-available server is difficult, as nameplate power values are generally overestimations and do not capture the power consumption for the multiple potential load levels. Offline power models are able to predict the consumption of hardware components accurately, but are usually intended for system design, requiring very specific and detailed knowledge about the system under consideration. In addition, existing power prediction for servers is limited to non-virtualized contexts or it does not take multiple load levels into account. These approaches also fail to leverage publicly available data on server efficiency and instead require experiments to be conducted on the target system. This makes them unwieldy when making decision regarding systems that are not yet available to the decision maker.

In this chapter, we address **RQ B.2**: "How to predict the power consumption of a concrete target application or software stack (such as a hypervisor) for a server for which the methodology's results are available?" by introducing two offline power prediction methods. Both use the readily available data provided by the SPEC SERT implementation of our power rating methodology to predict the power consumption of workloads on a target server that is otherwise unavailable for testing. Firstly, we address the research question regarding power prediction of hypervisors by predicting the power consumption of SERT workloads for different load levels in virtualized environments. Secondly, we address the question regarding third-party software applications by predicting the power consumption of such applications for multiple load levels.

We evaluate our approaches in Chapter 14. We compare predicted results of our hypervisor power prediction method against measurements in multiple virtualized environment configurations on a target server that differs significantly from the reference system used for experimentation. We show that power consumption of CPU and storage loads can be reliably predicted with a prediction error of less than 15% across all tested virtualized environment configurations. Finally, we evaluate the accuracy of our third-party application power prediction by predicting the power consumption of three applications on different physical servers. Our method is able to achieve an average prediction error of 9.49%.

8.1 Offline Power Prediction for Virtualized Environments

The use of virtualized environments is a common approach to tackle the issue of increasing data center power consumption. Such environments abstract the concrete hardware on which work is executed, thus offering more placement options. They can be used to consolidate work on a smaller number of servers or to dynamically change resource allocations at run-time. This may lead to increased energy efficiency despite the initial power and performance overhead caused by virtualization.

Prediction or estimation of hypervisor power consumption is a prerequisite for intelligent server purchasing decisions with the goal to optimize energy efficiency, as they are used on a majority of servers, especially in the cloud domain. Prediction techniques may be useful at various times. For example, they may help at run-time to predict the effects of workload migration; they may help at deployment time to determine the adequate resource allocation for a new application; they may help when designing a new execution environment to optimize its energy efficiency. The latter is especially challenging as the devices under consideration may not be available yet, preventing instrumentation of these devices; finally, cloud providers need a method for predicting the power consumption of hypervisors for not-yet-available servers. After all, the hypervisor executed on the server is one of the few workload-relevant points of information known to a cloud provider when purchasing new devices, whereas the actual workload depends on the customer and changes regularly at run-time.

Prediction of power consumption and energy efficiency of servers must also be capable of dealing with multiple load levels on the devices under consideration. Fan et al., 2007 state that servers usually operate in a load range between 10% and 50%. Consequently, while the prediction of power consumption at maximum load may be necessary and useful for capacity planning it offers little information about the actual power draw at run-time.

Existing methods for predicting power consumption that explicitly model virtualized environments are usually designed for run-time prediction using online data obtained during system execution. These methods use this run-time data to assist management decisions based on current and past system behavior. The major drawback of such an approach is its inability to provide information about systems that are not yet accessible, meaning that they cannot be used for impact assessment of a virtualized environment during the early stages of a cluster's or cloud's design.

Models that explicitly model the impact of a virtualized environment on the overall energy efficiency and power consumption also do not express differences in target system load levels. They support modeling the power impact of the virtualized environment at maximum load, whereas online models allow for assessment of load levels based on past run-time measurements.

In this section, we address **RQ B.2**: "How to predict the power consumption of a concrete target application or software stack (such as a hypervisor) for a server for which the methodology's results are available?" in the context of power prediction of hypervisors. We introduce an approach for explicit modeling of the impact a virtualized environment has on power consumption. The approach enables the prediction of this impact on two target load levels of 50% and 100%, the former being more relevant for predicting run-time power and the latter being more useful for capacity planning. The method is intended for off-line power prediction enabling its use before the target device is accessible. To this end, we utilize the data produced by the SPEC SERT implementation of our methodology, used by the U.S. EPA in the Energy Star standard (EPA, 2013).

By using the available SERT results of a target device and performing measurements in a virtualized and non-virtualized environment on an available reference machine, we can predict the impact of virtualization on the power consumption of the inaccessible target device.

The major contributions of this section are as follows:

1. We measure the power consumption of virtualized environments using the SPEC SERT, showing how standard benchmarks can be used to characterize the energy efficiency of servers running in such environments.

2. We present an approach for predicting the power consumption of multiple workloads running in a virtualized environment. This approach uses data that can be obtained without having direct access to the target server and enables prediction at two target load levels.

We evaluate our approach in Chapter 14, Section 14.1 by predicting the power consumption of multiple configurations of the Xen hypervisor (Chisnall, 2008). We measure virtualized and non-virtualized power consumption using the SPEC SERT, both on a reference (base) machine and on the target system. We predict power consumption of the target system based on measured results of the reference machine and the non-virtualized SERT results of the target system, as they would be measured for the U.S. EPA Energy Star submission. We then compare the predicted power consumption of the virtualized environment against our measurements on the target machine.

In the evaluation, we show that our approach is able to predict power consumption reliably for multiple hypervisor configurations at both load levels of 50% and 100%. We also show that it features great accuracy for the prediction of CPU-heavy SERT workloads' power consumption.

8.1.1 Measuring Power Consumption and Energy Efficiency

We measure power consumption and energy efficiency using SPEC's SERT implementation of our measurement methodology in Chapter 4. The methodology (and the SERT) have been designed to be executed on an operating system running directly on the SUT's hardware. The only additional abstraction layer is a JVM used to execute Java-based workloads. In addition to using SERT in non-virtualized environments, we also apply it in virtualized environments, as described in detail in Section 8.1.1.1.

We consider all of SERT's worklets, with *Capacity* serving in a double role. It is implemented as the regular *Capacity* worklet as part of the Memory workload and separately in the CPU workload under the name of *XMLvalidate*. *XMLvalidate* is configured to use less memory, never exceeding the memory capacity of the SUT and scale with the transaction rate, identical to the scaling behavior of all other CPU worklets.

8.1.1.1 Measuring Power on Virtualized Systems

We measure energy efficiency of virtualized systems by deploying our workloads within VMs on the SUT. We execute the workloads' transactions in parallel on all of the SUT's active VMs. Since all VMs share the same physical host, we expect the sum of the VMs transaction rates to be similar to the transaction rates on the non-virtualized SUT (or the SUT with exactly one active VM), although some performance impact by the virtualization itself is to be expected.

Virtual machines on the SUT may be over-provisioned, meaning that in the case of multiple VMs, each VM is allowed to utilize more of the physical resources than would be available if all VMs on the system were at full load. We account for this by also using a configuration that allows each VM to utilize all processors on the SUT (CPU-share of 100%). This means that, although each VM is theoretically capable of fully utilizing all available CPUs, it may not be able to do so in practice as other VMs may be occupying some CPU time.

Table 8.1: Virtual environment configurations.

Config.	# VMs	CPU-share
1	1	100%
2	2	100% (over-provisioning)
3	2	50% (equal sharing)

We measure and predict power consumption and energy efficiency both for virtualized environments that are configured to allow over-provisioning of processing time for the virtual machines, as well as a configuration that shares the available processors equally and does not allow for over-provisioning. Specifically, we use the three configurations shown in Table 8.1.

Fig. 8.1 shows that our general measurement methodology of achieving load levels by scaling the target throughput works as expected in virtualized environments. Throughput scales linearly over the different load levels with minor differences between the configurations. For the XMLValidate workload, over-provisioning seems to impact performance resulting in a slightly lower throughput, yet the workload still scales as expected over the load levels.

8.1.2 Prediction Approach

We predict the power consumption of a virtualized environment with a specified configuration on a given target system by using measurements from a reference (base) system. Again, the idea behind our prediction approach is to enable the prediction of power consumption of a virtualized environment on a target machine that is not available to the person making the prediction. Because of this, the only data that we can assume to be available to this person is a standard non-virtualized SERT result of the target machine. The general idea behind our prediction approach is that we train a model to capture how the virtualized environment in its given configuration affects the power characteristics of a system. This training is done using the reference system. Once training is complete, the learned model can be applied to the SERT result of the target system, providing a prediction and characterization of the power impact of the virtualized environment.

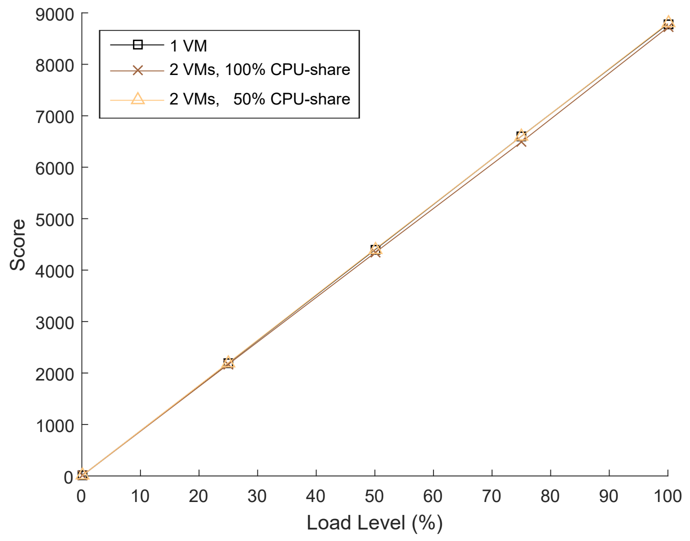


Figure 8.1: Throughput-scaling of the different VM configurations for XMLvalidate workload.

To this end, three sets of data must be available. A non-virtualized SERT result of the reference system, a SERT result of the virtualized environment on the reference system, and a SERT result of the non-virtualized target system. The reference system must be available to the person making the prediction, as specific measurements for the target environment must be performed on this system. The SERT results for the systems running without a virtualized environment may be obtained from third party sources (such as the system manufacturer). The non-virtualized results for the reference system can also be obtained through measurements by the person making the prediction.

We use multiple linear regression for prediction of the virtualized environment's impact on power consumption. Using the data obtained for the reference system, we construct a system of equations as our model. In this model, we train the linear coefficients for the regressor variables, which represent the power consumption of the workloads as measured in the non-virtualized environment. For each workload, the power consumption in the virtualized environment serves as response variables.

As stated in Section 8.1.1, we use 12 workloads in our model, providing us with 12 independent regressor variables: Idle, Compress, CryptoAES, LU, SOR, XMLValidate, Sort, SHA256, HDD Sequential, HDD Random, SSJ, and

Flood. Each of these workloads (except Idle) is measured at at least two load levels (50% and 100%). We use the workloads at their respective load levels to construct a system of equations as follows (Eq. 8.1):

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \tag{8.1}$$

where

$$\boldsymbol{y} = \begin{bmatrix} Idle \\ Compress_{100\%} \\ CryptoAES_{100\%} \\ \vdots \\ Flood_{100\%} \\ Idle \\ Compress_{50\%} \\ CryptoAES_{50\%} \\ \vdots \\ Flood_{50\%} \end{bmatrix}, \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{24} \end{bmatrix}, \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_{24} \end{bmatrix},$$

and $\boldsymbol{X}$ being the control matrix that contains the regressor variables. $\boldsymbol{X}$ is constructed as in Eq. 8.2.

$$\boldsymbol{X} = \begin{bmatrix} 0 & Compress_{100\%} & CryptoAES_{100\%} & \dots & Flood_{100\%} \\ Idle & 0 & CryptoAES_{100\%} & \dots & Flood_{100\%} \\ Idle & Compress_{100\%} & 0 & \dots & Flood_{100\%} \\ \vdots & & & \ddots & \vdots \\ Idle & Compress_{100\%} & CryptoAES_{100\%} & \dots & 0 \\ 0 & Compress_{50\%} & CryptoAES_{50\%} & \dots & Flood_{50\%} \\ Idle & 0 & CryptoAES_{50\%} & \dots & Flood_{50\%} \\ Idle & Compress_{50\%} & 0 & \dots & Flood_{50\%} \\ \vdots & & & \ddots & \vdots \\ Idle & Compress_{50\%} & CryptoAES_{50\%} & \dots & 0 \end{bmatrix} \tag{8.2}$$

However, creating a regression model where each workload is predicted using all other workloads is not optimal. Some workloads' power characteristics differ significantly from other workloads, as they scale differently with load levels or do not scale at all (Idle). This can lead to inaccurate predictions if such workloads are included in an equation for the prediction of another workload for which they are not suited.

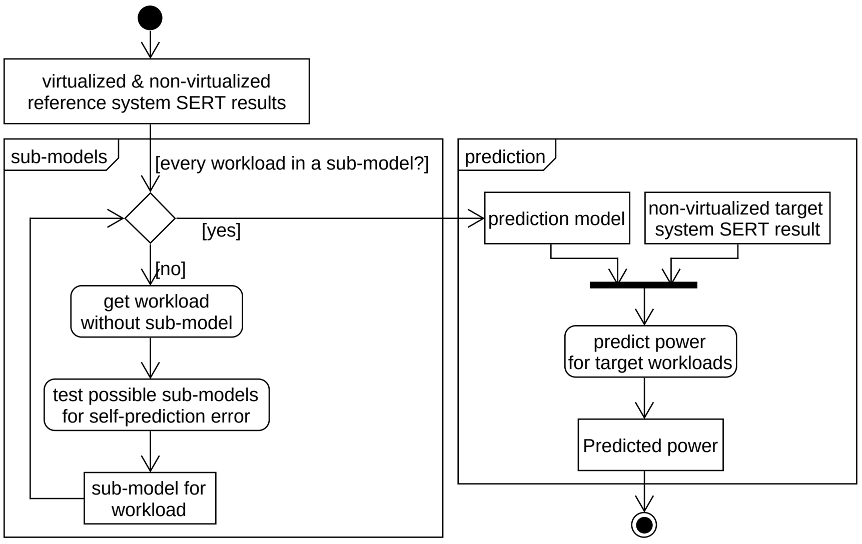

virtualized & non-virtualized reference system SERT results
sub-models
[every workload in a sub-model?]
[yes]
[no]
get workload without sub-model
test possible sub-models for self-prediction error
sub-model for workload
prediction
prediction model
non-virtualized target system SERT result
predict power for target workloads
Predicted power

Figure 8.2: Power prediction approach outline.

We introduce a heuristic for pruning workloads to avoid the use of workloads that may lead to a decrease in prediction accuracy. Specifically, we use the self-prediction error. The self-prediction error is the model's error when predicting the response variables it was trained with. We then create models that only contain subsets of the overall workloads. These sub-models are optimized towards their self-prediction error. This self-prediction error can be minimized either for a single target workload or for all workloads within the sub-model. For a single workload, the self-prediction error is the absolute difference between the predicted and measured power consumption. For all workloads in the sub-model the self-prediction error is the result of Eq. 8.3.

$$\text{self-prediction error} = \frac{1}{n} \cdot \sum_{i=1}^{n} \left| \frac{\text{predicted Watt}_i - \text{actual Watt}_i}{\text{actual Watt}_i} \right| \tag{8.3}$$

where i is are the indexes of the n workloads in the sub-model.

For each workload there are $2^{12} = 4096$ possible sub-models, as each workload may or may not be used in the sub-model. Removal of the empty sub-model and sub-models that include only one workload, results in a total of 4083 potential models. Fortunately, as the self-prediction error only depends on the reference system, these combinations must only be tested once and the same sub-models may then be reused for prediction of other target systems.

The sub-models are used for prediction of workload power consumption on the target system. For this prediction, they receive the result of a SERT run on the non-virtualized target system and use this to predict the power consumption the workloads would exhibit on the target system if they ran in the virtualized environment tested on the reference system. An outline of the entire approach is shown in Fig. 8.2.

Summarizing, we address **RQ B.2** ("How to predict the power consumption of a concrete target application or software stack (such as a hypervisor) for a server for which the methodology's results are available?") regarding power prediction of the software stack. To this end, we describe an offline prediction approach for the power consumption of virtual machine hypervisors. This approach uses results of the SPEC SERT to train regression and predict the SERT result of a virtualized system. We evaluate the approach's accuracy in Chapter 14.1. The prediction method of this section has been published at EPEW 2015 (Kistowski et al., 2016b).

8.2 Offline Power Prediction for Target Applications

Server providers rely on educated guesses and experience when trying to choose the most efficient and least consuming servers for their specific application. Nameplate information about the considered devices, as supplied by the vendor, is usually an overestimation and inaccurate. In addition, standard benchmarks do not reflect the target application to be run on the servers in question. The ability to accurately asses the power consumption of the servers under consideration for their intended application would enable planning and provisioning of energy efficient server landscapes.

Existing power models are either very generic and do not consider the workload, or they are assumed to be trained at run-time. Models that predict power for unavailable systems and components, such as those of Brooks et al., 2000 and Kahng et al., 2009, are primarily intended for server and hardware component design. They require very detailed knowledge on system internals, which would be available to a system designer, but not necessarily to someone intending to buy such a system.

This section addresses **RQ B.2**: "How to predict the power consumption of a concrete target application or software stack (such as a hypervisor) for a server for which the methodology's results are available?" regarding power prediction of software applications. It proposes a prediction method for power consumption of servers based on standard rating tool results. The prediction method uses publicly available rating data to predict the power consumption of servers under consideration for a target application. Specifically, the method uses data available from the SERT implementation of our power methodology of Chapter 4, which is required to run for U.S. E.P.A. Energy Star labeling (EPA, 2013). Using our method, a server provisioner can predict the power consumption of a future server running a target application even without access to it. Of course, this also implies that the server provisioner does not have to perform any measurements on the server under consideration.

Predicting the power consumption of servers using standard rating tool results is difficult, as this result set is relatively small. To address this issue, we consider multiple prediction formalisms and introduce a parameter optimization method. We also investigate the use of interpolation to generate additional data points for training and prediction.

The goal of this section is to provide accurate prediction of server power consumption for a target workload at multiple load levels, based on the SERT results of the target server. We envision the result of this prediction, containing information on the concrete workload under consideration and multiple load levels, to be of use for planners and potential server customers when making their decisions. In a nutshell, the core contributions are:

1. We introduce an offline prediction method for server power consumption based on standard rating tool results.
2. We present a parameter optimization method for automated tuning of prediction formalisms.
3. We investigate and show the use of interpolation methods to create additional training and prediction data for the power prediction domain.

We evaluate our offline prediction method in Chapter 14, Section 14.2 by predicting the power consumption of three target workloads on separate physical servers. We evaluate the prediction accuracy the effects of parameter optimization and interpolation. We show that our method can predict the power consumption of applications for an unavailable server using only standard rating tool results with a mean average absolute error of 9.5%, measured using real-world, physical servers and workloads. We also show that our combination of interpolation and parameter optimization methods greatly aids in achieving accurate predictions in a domain with little training and prediction data.

8.2.1 Challenges when using SERT for Offline Power Prediction

As explained in Chapter 4, SERT features seven CPU worklets in version 2.0. Each worklet is measured at four load levels (25%, 50%, 75%, 100%), with the exception of SSJ. SSJ is measured at eight load levels instead of the four levels of the other CPU worklets. In addition to the CPU worklets, the two storage worklets are run at two load levels (50% and 100%).

In addition, SERT features two memory worklets: *Flood* and *Capacity*. Flood tests the SUT's memory bandwidth, whereas Capacity tests its capacity. However, the memory worklets do not use the same load level scaling mechanism as the other worklets. Each features two load levels that differ in the amount of memory reserved for the worklet instead of scaling with the transaction rate. Because of this, we expect these worklets not be as useful in training power models as the storage and CPU worklets.

The major challenge in training models based on SERT results is twofold: *1)* The amount of data points per worklet is relatively small and *2)* it varies between the worklets. The load levels can be configured in theory, but standard compliant runs use the load levels described in this section. As we intend our model to be used with publicly available results, we must deal with the small and varying load level numbers.

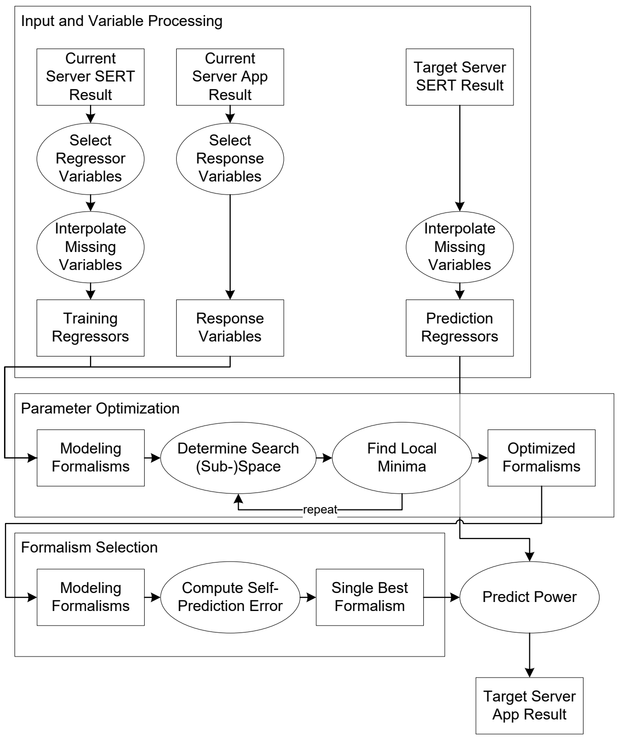

Input and Variable Processing
Current Server SERT Result
Current Server App Result
Target Server SERT Result
Select Regressor Variables
Select Response Variables
Interpolate Missing Variables
Interpolate Missing Variables
Training Regressors
Response Variables
Prediction Regressors
Parameter Optimization
Modeling Formalisms
Determine Search (Sub-)Space
Find Local Minima
Optimized Formalisms
repeat
Formalism Selection
Modeling Formalisms
Compute Self-Prediction Error
Single Best Formalism
Predict Power
Target Server App Result

Figure 8.3: Outline of power prediction approach.

8.3 Offline Power Prediction

The goal of our offline power prediction approach is the prediction of a *target application* on a *target server* not yet available to the operator currently running the application. The general idea behind the prediction approach is as follows: The operator has a *current server* on which the *target application* in question is being executed or upon which it can be deployed for testing. The operator measures the performance and power consumption of the target application for multiple load levels on this server. The prediction also requires a SERT result for the current server, which can be measured or obtained using a public database. Finally, it also requires a SERT result for the *target* server, which can be obtained from the vendor or a public database.

Predicting the power consumption of the target application for multiple potential target devices can help decision makers when deciding with which device to provision their cluster or data center. In addition, this approach could also be used in a general software placement context. An outline of the overall approach is illustrated in Figure 8.3. The approach is modular, allowing to use a regression method from a pool of methods. Specifically we consider the following methods:

- *Regression Tree (CART)*: the maximum number of leaf nodes in a tree (`max nodes`) and the number of instances in a node before splitting (`node size`)
- *Random Forest and Gradient Tree Boost*: the number of trees in the forest (`num trees`)
- *Gaussian Process Regression*: the `shrinkage` regularization parameter of the gaussian processe's kernel and the `kernel width`

8.3.1 Regressor and Response Variables

In general, power prediction is a prediction problem on a continuous scale and can thus be posed as a regression problem. These problem statements can be solved by various regression and/or classification algorithms. In general, regression problems have the following form (Eq. 8.4):

$$\boldsymbol{Y} \approx f(\boldsymbol{X}, \boldsymbol{\beta}) \tag{8.4}$$

where $\boldsymbol{y}$ is the vector of *response* variables (also called *dependent* variables), $\boldsymbol{X}$ the set of *regressor* variables (also called *independent* variables), and $\boldsymbol{\beta}$ the set of regression *parameters* to be trained. In this work, we map our domain specific measurement results to generic regressor variables and response variables, allowing for a range of regression and classification models to be used, as opposed to limiting ourselves to a single model.

8.3.1.1 Target Application Power Response Variable for Training

We measure the power consumption of the target application at multiple load levels on the *current server*. We then construct the training response vector from these power consumption results. As an example, the measurements at four load levels (100%, 75%, 50%, 25%), the training response vector y would be constructed as shown in Equation 8.5:

$$\boldsymbol{y} = \begin{bmatrix} power(App_{100\%}) \\ power(App_{75\%}) \\ power(App_{50\%}) \\ power(App_{25\%}) \end{bmatrix}. \tag{8.5}$$

Note that the number of load levels may have great impact on the prediction accuracy. Specifically, not all load levels measured for the target application may have measurement counterparts for all worklets in the SERT (and thus in our set X of independent regressor variables). We tackle this issue using interpolation in Section 8.3.3. As explained in Chapter 4, the load levels are derived from the workload's throughput. Consequently, this approach is applicable to any application with a measurable throughput. For many applications, this would be a request rate (e.g., HTTP requests per second for web applications).

8.3.1.2 SERT Results as Regressor Variables

The matrix X of independent regressor variables is constructed from the per-load level measurement results of the separate worklets. Specifically, we construct a vector from a single measurement metric, such as power or performance over the load levels of a worklet. We consider the following metrics for construction of our vectors:

- **Performance**: The average throughput of the worklet at the specified load level (in s^{-1}).
- **Power Consumption**: The average power consumption of the SUT when running the worklet at the specified load level (in W).

We do not consider the measured temperature, as it is measured as a control metric at the SUT inlet and thus independent from the system state. Equation 8.6 shows an example regressor variable matrix X that uses the load level percentages and power consumption of several worklets with four load levels.

$$\boldsymbol{X} = \begin{bmatrix} 1 & 0.75 & 0.5 & 0.25 \\ pwr(Com._{100\%}) & \ldots & \ldots & pwr(Com._{25\%}) \\ pwr(AES_{100\%}) & \ldots & \ldots & pwr(AES_{25\%}) \\ \vdots & \vdots & \vdots & \vdots \\ pwr(SSJ_{100\%}) & \ldots & \ldots & pwr(SSJ_{25\%}) \end{bmatrix}. \tag{8.6}$$

Again, the number of measured load levels is important and affects accuracy. Yet, as explained in Section 8.2.1, the worklets within SERT are measured with different load level counts. However, many regression methods require training vectors of equal size. In our case, this would imply measurements with the same load level counts for all worklets. We address this issue threefold by *(1)* discarding load levels, *(2)* discarding worklets, and *(3)* interpolation (see Section 8.3.3). We discard some worklets that do not fit into the load level schema (such as *Capacity*) or feature too few load levels for accurate interpolation (*Flood* and the storage worklets). In addition, we can add the power consumption of the *Idle* worklet as the power consumption of each worklet at 0% load.

8.3.1.3 System Power as Expected Output

The models predict system power consumption for each of the load levels of the application under consideration running on the target server. The number of load levels is implicitly specified by the size of the training response vector y.

8.3.2 Prediction Formalisms under Consideration

We consider four underlying prediction formalisms for power prediction: regression trees in three variations and Gaussian mixture models. These formalisms can be used as part of our power prediction framework. For regression trees, we use Regression trees (CART) (Breiman et al., 1984), gradient tree boosting (Friedman, 2001), and random forests (Breiman, 2001). In general, regression trees are built using binary recursive partitioning, which means repeated iterative splitting of data into partitions or branches.

Gaussian Mixture Models are stochastic models based on a a mix of probability distributions. They capture a target distribution using superposition of multiple Gaussian distributions, adjusting their means and covariances. Gaussian mixture models are a candidate for power prediction in this work, as they have been used to model power consumption of server environments in the past (Dhiman et al., 2010).

8.3.3 Interpolating Measurement Results

We require the same amount of measurements for each worklet used to construct the regressor matrix X. However, SERT's default settings measure different worklets with different load level counts. We consider two options to tackle this issue: zero padding and interpolation. Both methods are intended to create artificial results at the missing load levels for the respective worklets.

Zero padding simply fills the gaps in load levels for worklets with too few levels with results containing the value 0. This method is primarily useful if these results are needed for mathematical correctness, but otherwise discarded later in the prediction process (i.e., when the prediction method in question does not use them). Otherwise, the expectation is that zero padding enables the prediction to run, but with negative effects on prediction accuracy.

Interpolation creates missing results based on the neighboring load levels' results. In general, it is the reconstruction of a continuous function $f(x)$ from n different sample points $\{(x_1, f_1), (x_2, f_2), ..., (x_n, f_n)\}$. We investigate the use of three interpolation methods, which are explained in detail in Chapter 7:

- *Nearest Neighbor Interpolation*
- *Linear Interpolation*
- The *cross-validation interpolation* approach of Chapter 7, which selects an interpolation method from a range of polynomial and scattered interpolation methods using cross validation. Note that this approach requires a minimum of three points of data in a set to be interpolated. This is an issue in our use-case, as some worklets only feature two datapoints. We have to discard these worklets when using this interpolation, whereas nearest neighbor and linear interpolation are capable of working on the smaller sets.

8.3.4 Self-Prediction Accuracy

Our power prediction model uses parameter optimization and cross-validation of the different underlying modeling formalisms in order to increase the prediction's accuracy. During this optimization phase, each potential optimization candidate is evaluated and either discarded or adopted. We use the model's self-prediction error for this run-time evaluation. We then predict the training target application power consumption values and compare the resulting errors. We allow the use of several error metrics for this comparison and consider two, specifically:

1. *Root Mean Squared Error*:

$$e_{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(y'_i - y_i)^2}{n}}.$$

2. *Mean Absolute Percentage Error*:

$$e_{MAPE} = \frac{100}{n}\sum_{i=1}^{n}\frac{\|y'_i - y_i\|}{y_i}.$$

8.3.5 Parameter Modeling and Optimization

The prediction formalisms used by our offline prediction approach feature several different formalism-specific parameter settings that can affect the prediction's accuracy. Specifically, we consider the following parameters:

We optimize these parameters automatically using a method inspired by Noorshams et al., 2013. We create a generic parameter model, which sets the optimization exploration space for each paramter by assigning it a *minimum* and *maximum* value and an initial exploration *step* size. We then apply an iterative local search for each parameter. We split the parameter's search space between its minimum and maximum values into $k + 1$ eqal parts, resulting from the parameter's step size. We then calculate the self-prediction error of the model with the parameter under consideration for each of the potential values. We then create local search spaces around each local minimum, and iteratively explore those search spaces with a halved step size. This process repeats until a maximum search depth d is achieved, at which point the smallest local minimum is picked. We perform the parameter value search for each potential parameter. This is then repeated iteratively for all parameters for n iterations, where n the number of parameters per default.

Summarizing, we address **RQ B.2** ("How to predict the power consumption of a concrete target application or software stack (such as a hypervisor) for a server for which the methodology's results are available?") regarding power prediction of a target appplication. To this end, we describe an offline prediction approach for the power consumption of such applications. This approach uses results of the SPEC SERT to train regression models including interpolation and parameter optimization. We evaluate approach's accuracy in Chapter 14.2. The prediction method of this section has been published at ICPE 2019 (Kistowski et al., 2019a).

8.4 Concluding Remarks

This chapter addressed **RQ B.2** ("How to predict the power consumption of a concrete target application or software stack (such as a hypervisor) for a server for which the methodology's results are available?"). We address the question regarding power prediction of the software stack and regarding third-party software applications. To this end, we describe two offline prediction approaches able to predict the power consumption of otherwise unavailable servers. The first approach predicts the power consumption of virtual machine hypervisors, whereas the second predicts application power consumption. Both use results of the SPEC SERT to train their respective regression models.

We evaluate the accuracy of both approaches in Chapter 14. The work in this chapter has been published at EPEW 2015 (Kistowski et al., 2016b) and ICPE 2019 (Kistowski et al., 2019a).

Chapter 9

Online Prediction of Power Consumption

The power consumption of servers in data centers depends greatly on the software running on each server and how it interacts with the hardware. As a result, different deployments of distributed software components on heterogeneous servers can lead to significant differences in power consumption depending on which component is deployed on which server and depending on what workloads the components are exposed to. As workloads and load intensity change, components may be re-deployed or exchanged in order to reduce the power consumption for the current load profile. Deciding which component to place on which server during run-time remains difficult as the power consumption that would result from such a placement remains unknown. Existing work on component deployment optimization at run-time focuses on maximizing performance. Work that considers power consumption does so in the context of static design time placement decisions.

In this chapter, we address **RQ B.3** of `Goal B`: "How to use power measurements or pre-measured results at run-time to predict the power consumption of software component placements?". We introduce a model to predict the power consumption of component placements at run-time based on the load and power profile collected for a running distributed application in a heterogeneous environment. In addition, we present a model, which enables the use of our approach without dedicated power measurement devices, predicting power consumption based on load intensity and performance counters. We show that we can predict the power consumption of two different distributed web applications with a mean absolute percentage error of 2.2%. In addition, we can predict the power consumption of a system at a previously unobserved load level and component distribution with an error of 1.2%.

9.1 Introduction

The power consumed by servers in data centers depends on the specific software running on each server and on how this software interacts with the underlying hardware. Differences in the software design, specific implementation, and the hardware on which it runs can result in different power profiles, as can be observed in our rating methodology evaluation in Chapter 12. As a result, the deployment of distributed software components on heterogeneous servers can lead to significant differences in power consumption depending on which component is deployed on which server. In addition, the power consumption depends on each component's workload, which usually varies over time, especially in common transactional enterprise applications, such as the web or database applications considered in Chapter 11.1. For each software component, the power consumption may scale differently as the load increases. Therefore, the most efficient component placement on the servers under consideration may change over time.

As the load changes over time, components (or services) may be re-deployed to different devices in order to reduce the power consumption for the current load profile. They may also be replaced by components with the same interface but with a different implementation that provides better efficiency for the current load level. In addition, modern architectures allow for load balancing between multiple deployment instances of each component, allowing for a great number of potential deployment options. For example, components or (micro-) services may be replicated, exchanged, or moved to different devices. The power consumption of such potential deployments is unknown in advance, making informed management decisions difficult.

Existing approaches for prediction of non-functional properties of component deployment focus on performance. Architectural component modeling platforms, such as the ones of Becker et al., 2009 and Huber et al., 2017 support the prediction of performance characteristics at design time and run-time. Some architectural models also feature a power prediction aspect. However, these power prediction methods are used in a static context based on pre-defined system power models. Such models typically provide low accuracy or they are highly dependent on specific workloads (Rivoire et al., 2008) and require modeling in advance.

In this chapter, we propose an end-to-end modeling approach for predicting the power consumption of component placements at run-time. This includes a software deployment model, capturing the deployment of components on physical hosts while explicitly supporting the modeling of component replications and different component implementations. The deployment model is designed to only require basic information about the system, specifically about the structure and deployment infrastructure, such that it can be easily assembled without much modeling effort. The power prediction model makes use of run-time monitoring data about the observed power consumption under varying load levels. It captures the individual components' power profiles and their interactions based on which it can predict the power consumption of a new component placement configuration. The model can also be used to predict the power consumption at load levels that were not yet observed on the running system. In addition to the general end-to-end model, we introduce a separate single server power model, which is based on our offline power prediction model of Chapter 8.2. This single server power model eliminates the need for dedicated power measurement devices at run-time that would otherwise be required to use our general end-to-end prediction model.

The goal of this chapter is to enable informed component placement decisions that minimize the system power consumption. Our modeling approach can be used to find the least power consuming deployment option for a target throughput level. It can also be used in conjunction with any existing software performance model to optimize the energy efficiency of distributed systems. Note that our modeling approach is agnostic of the type of distributed architecture, e.g., if it is a component or (micro-)service based architecture. All of these architectures allow for distribution and replication of their software entities and our model is designed be used in either of these contexts. However, we use the term *component* throughout this paper to refer to components or (micro-)services.

The major contributions of this chapter and its evaluation in Chapter 12 are as follows:

1. We propose an end-to-end modeling approach for run-time prediction of the power consumption of component deployments in heterogeneous environments.

2. We present a deployment model that supports the use of replicated components with alternate implementations with the goal of enabling power predictions.

3. We integrate a separate single server power model that eliminates the need for dedicated power measurement devices for run-time power prediction.

4. We demonstrate the extent to which varying loads and varying component deployments can be used to train power prediction models at run-time.

We evaluate our models in Chapter 15 using two different web applications deployed in a heterogeneous environment with servers of different CPU hardware architectures. We predict the power consumption of different component placements, including placements that make use of component replication and that vary the component implementations. We show that we can predict the power consumption of previously unobserved deployments with a mean absolute percentage error of 2.2%. In addition, we show that we can predict the power consumption of a system at a previously unobserved load level and component distribution with an error of 1.2%.

9.2 Power Prediction Model

We introduce two separate power models. The first model is an end-to-end model for predicting the power consumption of a workload based on its deployment on physical or virtual machines and based on the load intensity. The second one predicts the power consumption of a physical server based on performance counters and standard benchmark results. The two models can be combined, as the end-to-end model requires the sum total power consumption of the physical machines on which it runs. It uses this data for training. This required information can be obtained using run-time measurements, which requires power instrumentation. This instrumentation can be omitted when combining it with our server power prediction model.

9.2.1 Workload Deployment Power Prediction

The workload deployment power prediction model is designed to predict the power consumption of an end-to-end system for a specific component deployment at a specified load.

The model input, training data, and expected output are illustrated in Fig. 9.1. The power model requires the current throughput and the current deployment as input. It is trained using the power consumption of the entire system, which depends on the deployment and throughput. The model predicts the end-to-end system power consumption for a specified new deployment. The prediction method is not fixed and may be changed.

9.2.1.1 Prediction Method

Power prediction is a problem on a continuous scale and can thus be posed as a regression problem. Consequently, both of our prediction models are designed to pose and solve regression problem statements. These problem statements

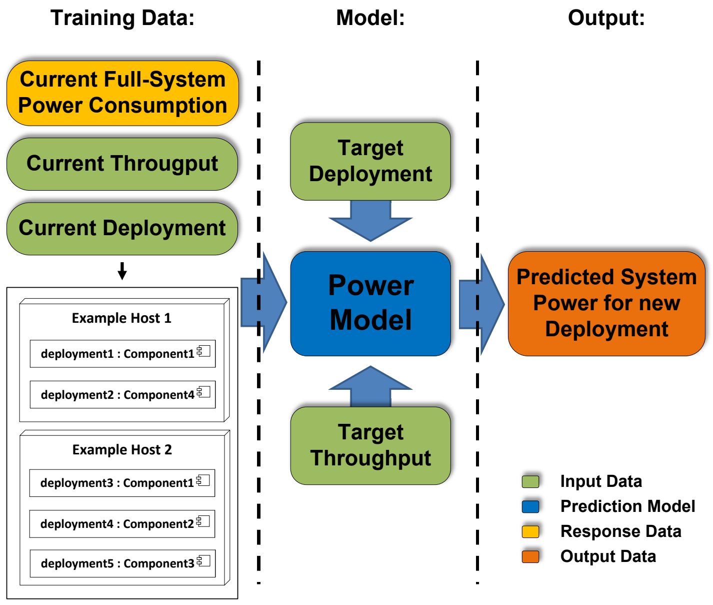


Figure 9.1: Workload deployment power prediction data flow with example deployment.

can be solved by various regression and/or classification algorithms. In general, regression problems have the following form (Eq. 9.1):

$$\boldsymbol{Y} \approx f(\boldsymbol{X}, \boldsymbol{\beta}) \tag{9.1}$$

where $\boldsymbol{Y}$ is the set of *dependent* variables (also called *response* variables), $\boldsymbol{X}$ the set of *independent* variables (also called *regressor* variables), and $\boldsymbol{\beta}$ the set of regression *parameters* to be trained. In this work, we use generic regressor variables and regression parameters, allowing any type of regression models to be used, as opposed to limiting ourselves to a single model, such as linear regression. The following regression models and algorithms are explicitly supported:

- Regression Trees, part of the Classification And Regression Tree (CART) class of algorithms, first introduced by Breiman et al., 1984,
- Random Forests, a classification algorithm using different uncorrelated parallel decision trees (Breiman, 2001),

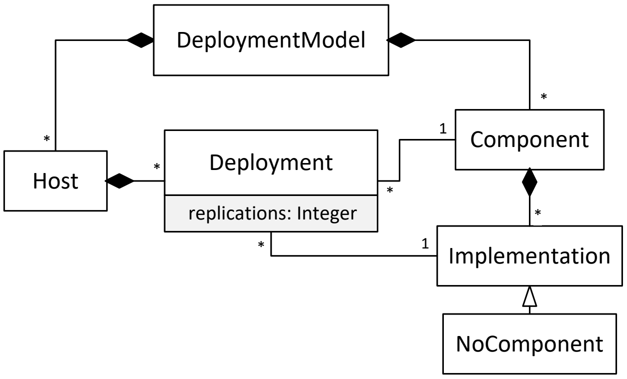


Figure 9.2: Deployment meta-model.

- Gradient Tree Boosting, a boosting method for decision trees, proposed by Friedman, 2001.

9.2.1.2 Throughput as Input Variable

We consider throughput on the level of the entire distributed application. We use throughput in favor of load intensity (request/job arrival rate) because of the scaling behavior of power consumption regarding these two properties. As long as the system is not saturated (remains under capacity), power consumption scales in a workload specific way as the load increases. At load levels below saturation, throughput equals the load intensity. However, once the load intensity exceeds system capacity (the maximum throughput), both the throughput and the power consumption stop scaling and stabilize at a given level. This observation is already used for workloads in power rating and benchmarking and is applied in our rating methodology in Chapter 4. It justifies our use of throughput instead of load intensity.

When mapping to regression model regressor variables, throughput can be mapped to a single variable.

9.2.1.3 Deployment Model as Input Variable

The deployment model describes which component (or service) is deployed on which host. It supports component replication (multiple instances of the same component) and multiple competing component implementations. Fig. 9.2 shows the deployment model's meta-model.

The deployment model maps the *components* to the physical *hosts* on which they are deployed. We refer to each mapping of a component to a host as a *deployment*. Each component may feature multiple deployments, as it may be replicated across multiple hosts. For each host, the components within its deployments must all be unique, however, if a component is deployed multiple times on the same host, the replication parameter of the deployment may be used to indicate this. Each component may have multiple component *implementations*. Each implementation has exactly one (possibly replicated) component that it implements. When deploying a component, the concrete implementation used in the deployment must be specified. We provide a *no component* implementation, which is used to create valid models for regression algorithms that require the specification of a mapping of each component to each host, even though the component may not be deployed on that host. Note that on a semantic level *no implementation* equals a replication factor of *0*. However, some power prediction methods may gain accuracy from modeling both the zero replication factor as well as the *no implementation* mapping. Also note that *hosts* may be either physical or virtual hosts, even if multiple virtual hosts are co-located on the same physical machine. However, our model assumes that virtual hosts are fixed to physical hosts to avoid having to explicitly model mappings between the two. Consequently, a VM migration would be modeled as a component changing its deployment to a new virtual host.

When mapping a deployment model to regressor variables $\boldsymbol{X}$, every component must feature a deployment for each host. It should use a *no component* implementation in case it is not actually deployed on the specific host. We specify regressor variables as shown in Algorithm 9.1. Firstly, we create a regressor for the total number of hosts. Then, for each host and each deployment, we create a regressor with the replication count of that deployment. We assign an integer value to each component implementation in ascending order, starting from 0. This implementation tag is used for the final regressor for the specific deployment. This final step may be omitted if only a single known implementation exists for the component in question.

The deployment model and the power prediction model as a whole are load distribution agnostic. This means that no explicit component assembly specification or seperate specification of distribution strategies is required. The effect of load distribution is implicitly learned when learning the relationship between deployment, throughput, and power consumption.

Algorithm 9.1: Mapping of the deployment model to regressor variables.

```
Data: regressors: ordered list of regressors $X$,
deploymentModel: instance of $DeploymentModel$
Function createRegressors()
    regressors.append(deploymentModel.hosts)
    for host ← deploymentModel.hosts do
        for deployment ← host.deployments do
            regressors.append(deployment.replications)
            if |deployment.component.implementations| > 1 then
                regressors.append(
                mapImplementationToInteger(
                deployment.implementation))
```

9.2.1.4 System Power Response Variable for Training

System power consumption in watts must be collected during run-time. We collect the aggregate power consumption of all physical hosts in the system. Power can be collected using one of two ways:

1. using one or more external power measurement devices which instrument the hosts under observation,
2. using the single server power prediction model, which is introduced in Section 9.2.2.

9.2.1.5 System Power as Expected Output

The model predicts system power for a given system throughput and deployment model instance. The predicted power in watts is the sum total consumption of all physical hosts.

9.2.1.6 Applying the Model to a Running System

The power prediction model is designed for use in a run-time system where data on current deployments, throughput, and power consumption (via measurements or through the use of our single server power modeling approach) is continuously being gathered. The model may be retrained at each point in time with the most current data sets. We envision the following application scenario (an example scenario is illustrated in Fig. 9.3).

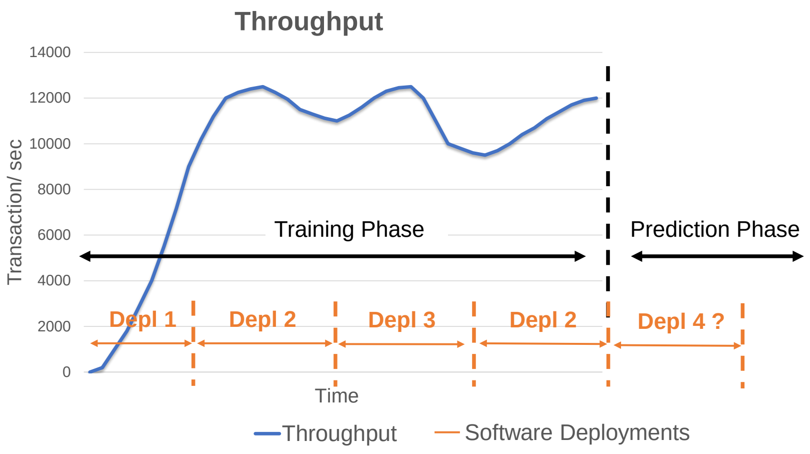


Figure 9.3: Example scenario for use of the workload deployment power prediction model.

The load intensity of production server systems typically varies over time. We train the model by observing the system under varying load intensity. We log the throughput of the system on a continuous basis (e.g., once per second). In addition, we log the power consumption and information on the current deployment. From these observations, we obtain a new training vector. The model may be retrained at any point in time and is expected to be retrained periodically. Once trained, it may be used to predict power consumption of a future configuration. This future configuration could be either a previously unobserved deployment, a previously unobserved load level, or both.

9.2.2 Single Server Power Prediction

The run-time power model described in the previous section requires power consumption data for training. However, we cannot assume that every physical server has a power analyzer attached to it. Consequently, in this section, we propose a server power prediction model that can be used to predict the power of a single server. This model is designed to be used as a substitute for separate power analyzers in case they are not available. It is inspired by the offline prediction model of Chapter 8.2, but adapted to use performance counters to enhance its accuracy in an online prediction context.

The single server power model requires offline training in order to fully characterize the power behavior of the servers in question. Again, we use the

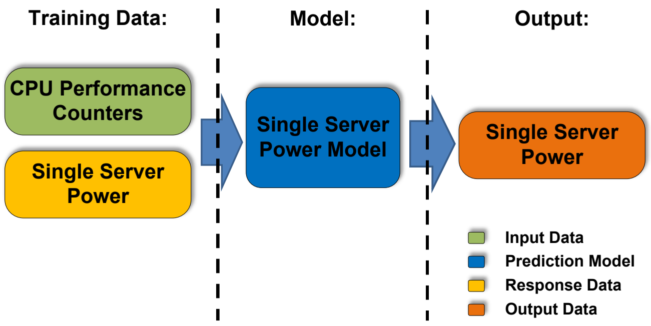


Figure 9.4: Single server power prediction model data flow.

SERT implementation of our rating methodology in Chapter 4 considering that it is run on all server models as part of the U.S. EPA Energy Star testing process. As a short recap: SERT uses a total of 13 different mini-workloads, called "worklets". Each of these worklets, except for the *idle* worklet, are executed at multiple load levels. For each of the load levels, SERT measures worklet throughput and power consumption on a per second basis. In contrast to the offline prediction method of Chapter 8.2, we configure our SERT runs, which gather the model's training data, to 25 load levels per worklet.

We also modify SERT to measure average CPU performance counters in addition to throughput and system power consumption for each load level. This is the same modification also used in Chapter 10. This allows us to train a power prediction model that predicts power consumption based on CPU performance counters (see Fig. 9.4). We train our model using those worklets for which power consumption scales with throughput and which are run at multiple throughput levels. Worklets fitting these criteria are the seven CPU worklets and the SSJ worklet (see Section 4.2.2.1).

Again, regression is used as prediction method. The CPU performance counters are mapped to regressor variables, the measured workload power is used as response variable.

We make use of the ability of modern CPUs to report their own power consumption. We employ this feature for power measurements, considering that CPU power consumption has been shown to correlate significantly with full-system power consumption (Rivoire et al., 2008). In addition, we consider workload memory characteristics. Unfortunately, the *memory power consumption* performance counter is only available on very few platforms, so we use *memory bytes written* counter instead. We expect it to have a greater correlation to the memory power consumption than the CPU counter for *memory bytes read*, as the latter includes cache reads (see our performance counter description and analysis in Chapter 6.1), which do not cause any memory power draw. We explore the model's accuracy when using further CPU performance counters in Chapter 15.4.

9.2.3 Concluding Remarks

This chapter addressed **RQ B.3** ("How to use power measurements or pre-measured results at run-time to predict the power consumption of software component placements?") by presenting an online prediction method for the power consumption of distributed component deployments on servers. It consists of two parts: an online part, which predicts power consumption of systems depending on component deployments and load intensity and an offline part, which predicts the power consumption of the single servers within the overall system using performance counters. We evaluate both parts separately and when used together in Chapter 15. The models and prediction methods of this chapter have been published at the ICAC 2018 conference (Kistowski et al., 2018a).

Part IV

Validation and Conclusions

Evaluation Goals

The work presented in this thesis follows two main goals, as introduced in Section 1.4:

Goal A: Create a comprehensive power and energy efficiency measurement and rating methodology for servers.
The methodology should enable rating of a broad range of servers and should be applicable for users in different contexts, including regulators, system designers, and potential buyers.

Goal B: Provide methods for using the results of the measurement methodology for data center provisioners and/or operators.
Data center provisioners and/or operators should be given methods to use the data provided in the measurement results for better decision making by helping to predict the effects of their actions.

To achieve this goal, we introduced a measurement and rating methodology, workloads, and load models and distributions in Part II and power models and prediction methods in Part III.

The goal of the evaluation presented in this part is to demonstrate that our rating methodology and the additional contributions adding to it adhere to the following quality criteria, as presented in Section 2.4: *(1)* relevance, *(2)* reproducibility, and *(3)* fairness. Regarding the models and prediction methods, our main evaluation goal is to determine the *accuracy* at which these formalisms can describe power consumption.

In the following, these evaluation goals are broken down into several specific evaluation questions (**EQs**) to be answered. We pose evaluation questions for our rating methodology and its related contributions and describe evaluation aspects concerning the accuracy of our power models and prediction methods.

Quality of Power Rating Methodology

Our power measurement and rating methodology should ensure the primary quality criteria for benchmarks (see Section 2.4). We evaluate it regarding the criteria of *relevance*, *reproducibility* and *fairness*. The remaining two criteria, verifiability and usability, do not receive focus since they depend on the specific implementations and applications of the methodology. Regarding reproducibility, our evaluation must answer these questions:

- **EQ A.1**: Does the methodology produce consistent power and performance results when applying it repeatedly? (addresses *reproducibility*)
- **EQ A.2**: Does the metric correctly order systems by their energy efficiency? (addresses *fairness*)
- **EQ A.3**: Does the metric have a relationship with the energy efficiency of real-world workloads? (addresses *relevance*)

We pose evaluation questions for the additional contributions of Part II, which are evaluated in Chapters 11 and 12. Those contributions also address some of the above mentioned quality criteria. Each of the following questions targets the quality criterion, which the specific contribution addresses the most. This is mostly *relevance*, with one exception.

- **EQ A.4**: Can the modeling formalism and extractor accurately express the load variations of real-world load profiles? (primary quality criterion: *relevance*)
- **EQ A.5**: Can we evaluate the energy efficiency of load distribution policies? (primary quality criterion: *relevance*)
- **EQ A.6**: Can we accurately emulate the power profile of a more complex workloads using performance counters? (primary quality criterion: *reproducibility*)
- **EQ A.7**: Can our distributed reference application be used to evaluate the quality of micro-service placements regarding energy efficiency and power? (primary quality criterion: *relevance*)

Prediction Accuracy

Our power models and prediction methods should model and predict the power consumption of servers with good accuracy. Therefore, our evaluation questions are relatively straight forward:

- **EQ B.1**: Can interpolation be used to accurately model power consumption per load level on servers?
- **EQ B.2**: Can we accurately predict the power consumption of hypervisors and applications on unavailable servers, based on results provided by our rating methodology?
- **EQ B.3**: Can we accurately predict the power consumption of a potential software component placement based on run-time data?

Chapter 10

Quality of the Server Efficiency Rating Methodology

We evaluate our methodology with respect to the primary quality criteria for benchmarks (see Chapter 2). We address relevance, reproducibility and fairness. Verifiability and usability are not considered since they depend on the specific implementations and applications of the methodology. We evaluate reproducibility by analyzing the power and performance variations when applying the methodology repeatedly. We show the fairness of the methodology's metric by presenting a mathematical proof and showing correlations of metric results to energy efficiency properties of servers. Finally, we evaluate relevance by analyzing the metric score's relationship to real-world workload energy efficiency.

The evaluation of our methodology answers the evaluation questions **EQ A.1** through **EQ A.3**. To recap the questions, they are:

- **EQ A.1**: Does the methodology produce consistent power and performance results when applying it repeatedly? (addresses *reproducibility*)
- **EQ A.2**: Does the metric correctly order systems by their energy efficiency? (addresses *fairness*)
- **EQ A.3**: Does the metric have a relationship with the energy efficiency of real-world workloads? (addresses *relevance*)

10.1 Reproducibility

We investigate the reproducibility of our methodology, addressing **EQ A.1** ("Does the methodology produce consistent power and performance results when applying it repeatedly?") by analyzing the variations in energy efficiency and power consumption of selected worklets. We analyze these variations between separate (re-)runs of a test. The goal of this investigation is to show that these variations are sufficiently small to be able to trust a result obtained using our methodology. We focus on CPU-worklets with high power consumption, as the CPU is the hardware part with the greatest power variation in commodity servers (Barroso and Holzle, 2007). Consequently, we focus on the *LU*, *Compress*, *SOR*, and *SHA256* worklets. As a reference point, we characterize and measure the power consumption of the *Idle* worklet. We investigate how variations in efficiency and power consumption are affected by a worklet's load level and by BIOS settings. Specifically, we investigate the effect that the CPU Turbo has on power variations. As a reference workload, we use LINPACK. It is a high power-consuming workload that does not employ our methodology, it only runs at a single (maximum) load level. However, it still enables us to compare the run-to-run and intra-run variation of workloads within our methodology to a workload that does not employ it. We use LINPACK to compare power consumption variations only (not efficiency), as it does not measure throughput. Similarly, Idle is only used when comparing power consumption.

We use two different systems for these experiments:

- **Sun Server X3-2** system with 4 x 4 GB RAM. This system has an Intel Xeon E5-2609 processor (80 W Thermal Design Power (TDP)) with four cores running at a frequency of 2.4 GHz. This system does not feature a turbo mode. Instead, we perform a series of tests with all BIOS power management disabled.

- **Fujitsu RX2540 M1** system with 4 x 16 GB RAM. We use an Intel's Xeon E5-2680 v3 processor on this system. It features 12 cores and a base frequency of 2.5 GHz (up to 3.3 GHz with turbo, 120 W TDP).

We employ the Intel Performance Counter Monitor (Intel PCM, see Willhalm et al., 2012) to collect additional performance data in addition to full-system power consumption and worklet throughput. Intel Performance Counter Monitor (Intel PCM) collects performance counters from the CPU itself. We use it to collect information on CPU power consumption, CPU temperature, and clock frequency.

10.1.1 Run-to-Run Efficiency and Power Variations

We investigate the run-to-run variations of our worklets by repeating our entire measurement suite 50 times on a Sun Server X3-2 system using a single E5-2609 processor. We run the entire experiment once with BIOS power management enabled and once with BIOS power management disabled. Firstly, we investigate the coefficient of variation (CV) for the energy efficiency score of each

Table 10.1: CVs in % for mean energy efficiency for measurement repeats on Sun Server X3-2.

	Load	Com.	LU	SHA	SOR
No Mgmt.	25%	0.92	1.05	0.88	0.85
	50%	1.02	0.95	0.72	0.69
	75%	0.96	0.83	0.91	0.71
	100%	0.91	1.04	0.79	0.72
Mgmt.	25%	0.95	1.01	1.04	0.83
	50%	0.87	0.87	0.92	0.75
	75%	0.78	0.87	0.77	0.68
	100%	0.77	0.84	0.67	0.68

interval. The CV is a normalized value defined as the ratio of the standard deviation divided by the sample's mean. In contrast to a comparison of the raw standard deviation, the CV allows comparing workloads and load levels with different mean power consumption or efficiency.

The CVs for the interval efficiencies in Table 10.1 are very small, regardless of BIOS setting. The greatest CV is the CV of 1.04% for LU at 100% load (and SHA at 25% load). Regarding absolute power consumption, Idle min/max power consumption difference is 0.1 W and the greatest absolute difference is LU at 75% load with 0.8 W. Notably, the repeated measurements exhibit a correlation between power consumption and CPU temperature. The corresponding correlation coefficient is greater than 84% for all interval measurements with the exception of SHA256 at 55% correlation. However, there seems to be no significant correlation between power and performance variations, with a mean correlation coefficient of 21.9%.

To investigate further, we compare mean power consumption measured in the different runs. This comparison includes LINPACK and Idle in addition to the SERT worklets.

The power means of the separate measurement runs only differ slightly. With balanced power management, Idle CPU power shows the smallest mean difference of 0.1 W (0.67%), ranging between 14.83 W and 14.93 W. The largest mean difference for the worklets is found for LU at 50% load. However, it is only a total difference of 1.32 W (4.02%), with the minimum being 32.82 W and maximum CPU power of 34.14 W. LINPACK features the overall greatest total difference of 3.50 W (6.80%).

Subsequently, the coefficients of variation for the measurement means in Table 10.2 are also very small. The greatest CV is the CV for LINPACK (which

Table 10.2: CVs in % for mean CPU power for measurement repeats on Xeon E5-2609.

	Load	Idle	Com.	LU	SHA	SOR	LIN.
No Mgmt.	0%	0.17					
	25%		0.06	0.31	0.23	0.25	
	50%		0.09	0.35	0.36	0.20	
	75%		0.11	0.59	0.41	0.15	
	100%		0.12	0.10	0.32	0.07	0.12
With Mgmt.	0%	0.13					
	25%		0.15	0.26	0.24	0.13	
	50%		0.13	0.81	0.31	0.28	
	75%		0.11	0.60	0.31	0.21	
	100%		0.15	0.15	0.21	0.18	1.19

also features the greatest min/max difference) with 1.19%, the next greatest being 0.81% for LU at 50% load. These numbers do not change significantly with BIOS power management disabled. Idle min/max difference is still 0.1 W and the greatest min/max difference is LU at 75% load with an absolute difference of 0.8 W and a CV of 0.6%.

Again, all workload/load level combinations show a positive correlation coefficient regarding CPU temperature. This coefficient is the largest at full load, with all coefficients greater than 84% with the exception of SHA256 at 55% correlation.

Generally, we see that efficiency variations are very small, both for BIOS power management enabled and disabled. Even these small variations can be correlated temperature, which is at least partially an external effect under control of the tester.

10.1.2 Intra-Run Power Variations

In addition to our run-to-run efficiency and power comparisons of the previous section, we analyze power variations within a single measurement run using our methodology executed on the Fujitsu system with its Xeon E5-2680 v3 processor. Regarding performance variations, note that the methodology specifies a maximum CV of 5% within a run. Results with a higher performance variation are considered invalid and must be re-run. As a result, we focus on the variations in power measurements.

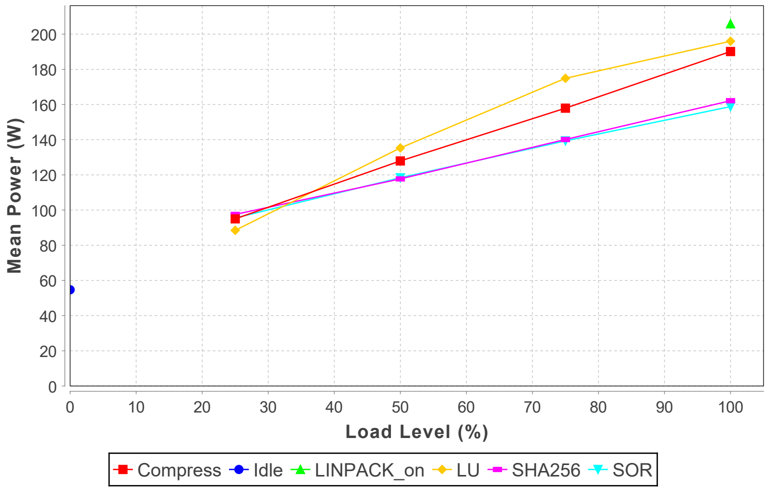


Figure 10.1: System power consumption of the Fujitsu server with median Xeon E5-2680 v3 processor.

System power consumption for all workloads on the Fujitsu system is shown in Figure 10.1. The figure displays the power consumption (in Watt) for each of the workloads over the full range of load levels. In contrast to the other workloads, Idle and the reference workload LINPACK feature only one load level each. Idle power is the smallest consumer, whereas LINPACK is the largest consumer, closely followed by LU at full load. The workloads scale almost linearly over load levels, increasing in power consumption with each additional level. In this section, we focus on the run-to-run and intra-run variances of the power measurement for each separate load level/workload combination. An analysis of the power scaling behavior of the workloads can be found in Section 10.3.

Again, we analyze the coefficient of variation (CV) for the power measurements. Table 10.3 displays the CV for CPU power consumption as measured by the Intel RAPL counter. It shows that variation during a 120 second measurement interval is relatively low and stable at a CV between 0.3% and 1.44%. This variation is independent of the processor's current load level and turbo setting. It is also similar for many workloads. The compression workload varies more in its power consumption. We attribute this behavior to the greater intensity of its memory access which introduces more seemingly random behavior in performance and power consumption.

The LINPACK workload behaves slightly different to the worklets running within our methodology as it varies even less in its power consumption. This is not purely attributable to the fact that LINPACK does not run within our methodology, though. The LU worklet, which runs within our methodology, features even smaller variations at maximum load. This indicates that LINPACK's low variation is more a result of workload design, rather than the underlying measurement methodology.

The major outlier in Table 10.3 is the Idle workload. On a single idle processor sample, power consumption can vary for 35.17% with turbo on, but only 0.84% without turbo. This observation underlines that operating system and CPU power management may lead to variations in power consumption, as idle power minimization is a major goal of those power saving mechanisms. This variation can already be observed during the time frame of a single measurement run. We back this assertion with the correlation of CPU power consumption and frequency. For most workloads, CPU power consumption and frequency correlate little. To illustrate, LU's CPU power/frequency correlation coefficient ranges between 8.6% (25% load) and 33.3% (100% load). Other workloads are similar, as frequency remains relatively stable during the workloads' execution. The Idle workload, however, shows significant correlation. The Idle interval with the turbo setting on, in particular, exhibits a CPU power/frequency correlation coefficient of 94.2%. This observation backs two design decisions within our methodology: It shows that our workload scheduling mechanism successfully manages to achieve a stable load on the CPU, preventing sudden frequency changes that would affect power measurements during the run. It also backs the decision to not include the Idle power consumption in the final metric score. Idle power is an important characteristic of computing systems and should thus be reported, but these results show that measuring it can be somewhat error prone, which might negatively affect the stability of any final metric which includes it.

Table 10.4 shows the variations in full system power for each measurement interval. Most CVs are smaller than their CPU power counterparts in Table 10.3. This indicates that a majority of the variation is caused by CPU power variations, rather than power variations due to other system components. This is especially true for the variations during system idle time. Specifically, the Idle worklet with turbo enabled features a CPU power standard deviation of 4.02 W at a mean CPU power consumption of 12.11 W, whereas the entire system features a similar standard deviation of 5.73 W at a mean power consumption of 54.07 W. The correlation coefficient of 99.93% between CPU and system power for the Idle measurement underlines this observation that CPU power directly influences system power.

LINPACK behaves differently, however. Its system power CV is significantly

Table 10.3: Coefficients of variaton (CV) in % for CPU power consumption on system, running a Xeon E5-2680 v3 processor.

	Load	Idle	Com.	LU	SHA	SOR	LIN.
Turbo on	0%	33.17					
	25%		5.63	0.53	1.20	3.63	
	50%		0.65	0.66	1.44	0.55	
	75%		0.61	0.66	1.23	0.42	
	100%		0.30	0.03	0.63	0.52	0.04
Turbo off	0%	0.84					
	25%		2.51	0.56	1.11	2.46	
	50%		0.57	0.60	1.23	0.39	
	75%		0.62	0.41	1.27	0.38	
	100%		0.38	0.40	0.61	0.05	0.04

Table 10.4: CV in % for system power consumption on Xeon E5-2680 v3 system.

	Load	Idle	Com.	LU	SHA	SOR	LIN.
Turbo on	0%	10.47					
	25%		2.59	0.30	0.71	1.73	
	50%		0.36	0.46	0.90	0.30	
	75%		0.50	0.45	0.89	0.31	
	100%		0.36	0.19	0.29	0.33	4.14
Turbo off	0%	0.34					
	25%		1.00	0.31	0.63	1.25	
	50%		0.34	0.35	0.70	0.21	
	75%		0.39	0.30	0.82	0.25	
	100%		0.28	0.30	0.17	0.15	3.69

larger than its CPU power CV. The same is true for the standard deviation of its power consumption. We attribute this effect to LINPACK using additional system resources, especially cooling, that are not visible in the CPU power consumption.

Concluding, we learn that power measurements are very stable, both within one measurement run and in their means for multiple runs, addressing **EQ A.1** ("Does the methodology produce consistent power and performance results when applying it repeatedly?"). The mean power consumption remains very similar. The Idle workload is the major outlier. It can pose challenges as unforeseen power management may cause significant variation during single measurement intervals. However, mean idle power remains consistent over multiple measurements, which indicates that our default measurement duration is reasonable.

10.2 Fairness

We consider the fairness of our methodology to be primarily influenced by its metric, as the metric produces the results used for any comparison for which fairness should be ensured. In addition to the metric, testing constraints as defined by the regulators using the methodology also influence fairness, but are outside of the scope of the methodology itself.

We investigate the applicability of the metric in two steps, addressing **EQ A.2** ("Does the metric correctly order systems by their energy efficiency?"): Firstly, we show that the metric correctly accounts for changes in basic energy efficiency related properties of the tested system by presenting a mathematical proof. Secondly, we use measurements from 385 different servers to show that the metric results correlate to many intuitions regarding energy efficiency. Note that we evaluate the relationship of the partial metrics within our metric in Section 10.3.

10.2.1 Metric Changes for Basic Energy-Efficiency Properties

Firstly, we show that our metric scales as expected with changes in system properties that should always result in an improvement of the energy efficiency score (or inverse). This helps to validate the fundamental properties of our metric. For this validation, we consider two systems A and B, where system A is identical to system B in all but one characteristic (e.g., power consumption). This characteristic differs for at least one point of measurement (measurement result of one interval for a single worklet). Basically, we show that a simple improvement in a single characteristic of a server would lead to the expected result in the power methodology's metric.

The weighted geometric mean (GM) over a list $X = (x_1, \ldots, x_n)$ of size n is smaller than the weighted geometric mean over a list $Y = (y_1, \ldots, y_n)$ of size n if Eq. 10.1 holds true for all elements within X and Y:

$$\forall i \in [1, n] : x_i \leq y_i \text{ and } \exists j \in [1, n] : x_j < y_j. \tag{10.1}$$

Lemma 1 can also be applied to unweighted geometric means, as these are weighted geometric means where each weight $w_i = \frac{1}{n}$.

Proof of Lemma 1.

$$\begin{aligned}
x_j < y_j &\implies ln(x_j) < ln(y_j) \\
&\implies w_j * ln(x_j) < w_j * ln(y_j) \\
&\xRightarrow{Eq.10.1} \sum_{i=0}^{n} (w_j * \ln(x_i)) < \sum_{i=0}^{n} (w_j * \ln(y_i)) \\
&\implies \exp\left(\sum_{x_j \in X} w_j * \ln(x_j)\right) < \exp\left(\sum_{y_j \in Y} w_j * \ln(y_j)\right) \\
&= \quad GM(X) < GM(Y)
\end{aligned}$$

□

To recap: The final metric score is the weighted geometric mean of the workload scores (O), which, in turn, are the geometric means of the worklet scores (E). The worklet scores are the geometric means of the load level scores (L, with load levels $l_i \in L$), and each load level score is the ratio of normalized throughput over power consumption ($\frac{t}{p}$).

Using Lemma 1, we show the following three properties of the metric:

1. If server A has a lower power consumption than server B for at least one measurement interval (with otherwise identical power consumption and performance), the final metric score improves.

2. If server A has a higher performance than server B for at least one measurement interval (with otherwise identical power consumption and performance), the final metric score improves.

3. If server A has a higher overall efficiency than server B for at least one measurement interval (with otherwise identical power consumption and performance), the final metric score improves (generalization of 1. and 2.).

Proof of Metric Properties.

1. $\exists i : l_i(A) \in L(A), l_i(A) = \frac{t_i(A)}{p_i(A)}$ and $l_i(B) \in L(B), l_i(B) = \frac{t_i(B)}{p_i(B)}$ where $t_i(A) = t_i(B)$ and $p_i(A) < p_i(B)$,
   but $\forall l_j(A)$ and $\forall l_j(B) \in L(B)$ where $j \neq i : l_j(A) = l_j(B)$
   $p_i(A) < p_i(B) \implies \frac{t_i(A)}{p_i(A)} > \frac{t_i(B)}{p_i(B)} \implies l_i(A) > l_i(B)$

2. $\exists i : l_i(A) \in L(A), l_i(A) = \frac{t_i(A)}{p_i(A)}$ and $l_i(B) \in L(B), l_i(B) = \frac{t_i(B)}{p_i(B)}$ where $t_i(A) > t_i(B)$ and $p_i(A) = p_i(B)$,
   but $\forall l_j(A)$ and $\forall l_j(B) \in L(B)$ where $j \neq i : l_j(A) = l_j(B)$
   $t_i(A) > t_i(B) \implies \frac{t_i(A)}{p_i(A)} > \frac{t_i(B)}{p_i(B)} \implies l_i(A) > l_i(B)$
3. $\exists i : l_i(A) \in L(A)$ and $l_i(B) \in L(B)$ where $l_i(A) > l_i(B)$,
   but $\forall l_j(A) \in L(A)$ and $\forall l_j(B) \in L(B)$ where $j \neq i : l_j(A) = l_j(B)$

$$
\begin{aligned}
l_i(A) > l_i(B) &\xRightarrow{\textit{Lemma 1}} GM(L(A)) > GM(L(B)) \implies E(A) > E(B) \\
&\implies GM(E(A)) > GM(E(B)) \implies O(A) > O(B) \\
&\implies GM(O(A)) > GM(O(B))
\end{aligned}
$$

□

10.2.2 Metric Score Correlations

We investigate the applicability of the metric by analyzing the correlation coefficient of metric scores with changes in basic server properties. For this analysis, we collect SERT results from a total of 385 different physical servers of different generations and architectures[1]. Analogously to the mathematical analysis in Section 10.2.1, we analyze how changes in basic energy and power-related server properties correlate with the metric's score.

Similar systems (systems with similar or the same hardware) should feature some correlation between these basic properties and the metric score. For example, due to their similarity, system configurations based on the same processor and overall system manufacturer are likely to be more energy-efficient if they feature a lower idle power consumption. We expect such correlations to exist for similar systems. However, we also expect this correlation to be lower for systems from different generations or architectures, as these become more diverse and feature more unknown factors that impact energy efficiency. Similarly, we expect large systems (systems with many processor sockets, multi-server blade systems, etc.) to have too many variation points regarding energy efficiency, making it difficult to correlate their efficiency to a single property. Even for similar systems, we never expect near-perfect correlation as that would mean that energy efficiency can be reduced to the simple properties under analysis.

[1]Note: Some results are non-compliant and have been obatined according to the SPEC fair-use policy for academic use.

Table 10.5: Correlation coefficients of energy efficiency score and power characteristics for servers, grouped by number of sockets.

Servers		Correlation Coefficients		
#Servers	**#Sockets**	**Idle Power**	**Min Power**	**Max Power**
82	1	-0.68	-0.52	-0.48
326	2	-0.19	-0.17	0.06
67	4	-0.45	-0.41	-0.29

We consider the following properties:

- **Idle power consumption**: Systems that scale similarly over load levels regarding their energy efficiency but differ in base power consumption (idle power) should feature a negative correlation with idle power consumption. Systems with great variations in their efficiency scaling and/or performance might not feature a significant correlation with idle power, though.
- **Minimum power consumption under load**: Analogously to idle power, we expect similar system's energy efficiency to correlate with the minimum power consumption measured using our methodology while the system was under load.
- **Maximum power consumption**: We also expect similar system's energy efficiency to correlate with maximum power consumption.

We calculate the Pearson correlation coefficient for these properties and the metric's score on the servers in question. We group the servers into categories based on name-plate similarity, meaning similarity regarding their nominal hardware configuration. Firstly, we investigate server energy efficiency score correlations based on processor socket count.

Table 10.5 shows the correlation coefficients for the servers' energy efficiency with idle, minimum, and maximum power, grouped by the number of CPU sockets. It shows an expected negative correlation between energy efficiency and idle power for the single-socket system. Similarly, minimum and maximum power feature a small, slightly less pronounced negative correlation. These correlations show that many systems with a lower idle, minimum, or maximum power consumption manage to get a better energy efficiency score. Yet, it also shows that servers are diverse enough that energy efficiency is not always better for systems with lower idle, minimum, or maximum power. These correlations indicate that our efficiency metric behaves as expected. Interestingly, the

Table 10.6: Correlation coefficients of energy efficiency score and power characteristics for servers, grouped by processor models.

Servers		Correlation Coefficients		
#Servers	**CPU**	**Idle Power**	**Min Power**	**Max Power**
12	E3-1230 v3	-0.97	-0.94	-0.95
19	E5-2603 v3	0.05	-0.20	0.05
16	E5-2620 v3	-0.73	-0.65	-0.69
16	E5-2630 v3	-0.94	-0.92	-0.98
20	E5-2660 v3	-0.51	-0.44	-0.47
17	E5-2690	-0.36	-0.83	-0.32
24	E5-2690 v2	-0.36	-0.40	-0.41
29	E5-2699 v3	-0.16	-0.11	-0.34

four-socket systems also feature some correlation for their power consumption metrics and energy efficiency. Idle and min power have only slightly lower correlations to energy efficiency when compared to the single-socket systems. Maximum power does not follow this trend, though, as four-socket systems feature significant variations and volatility regarding power consumption under high load. The dual-socket systems are the major outliers, their great number and variations between systems cause no significant correlation between efficiency and the simple power properties.

To stress that the correlations exist for similar classes of servers, we group servers by CPU model for all models with at least 12 different measured configurations. Table 10.6 shows the correlation coefficients for idle, min, and max power with energy efficiency for these servers. These groups show significant correlations between the power metrics and energy efficiency, as derived by our methodology's metric. Correlations are greater for "smaller" processors (i.e., processors with less power and performance volatility). The only exception is the group of servers using the E5-2620 v3 processor. The measurements for these processors have been taken on almost-identical server configurations, leaving too much space for the random measurement noise to influence the correlation coefficient.

In general, the correlations show that our energy efficiency metric shows an increased efficiency for systems where such an increase would be expected. However, they also show the need for a complex energy efficiency metric, as the characteristic of energy efficiency does not always boil down to simple power properties, such as a reduced idle consumption.

Answering **EQ A.2** ("Does the metric correctly order systems by their energy efficiency?"), our mathematical proof and the correlations show that the metric does order our servers under consideration correctly by matching the expected order and properties of energy efficiency in those cases where they are already known.

10.3 Relevance

Finally, we address the aspect of relevance and **EQ A.3** ("Does the metric score have a relationship with the energy efficiency of real-world workloads?"). To this end, we show that our metric produces relevant results by defining a relationship between those results and the efficiency of a real-world workload. The goal of showing such a relationship is demonstrating that it is possible to define a semantic relationship between the workload scores within the metric and the energy efficiency of a third-party software. This shows that a person running tests using our methodology could use a result based on our metric to decide on the efficiency of a system regarding their target software applications.

We measure the energy efficiency of SERT Suite and the Dell DVDStore (Dell, Inc., 2011), as our third-party application, on six servers. Next, we weight our SERT workload scores with weights that correspond to the CPU, memory, and storage behavior of the DVDStore workloads. We then use these weights to calculate our final server efficiency score (see Eq. 4.6), instead of the standard U.S. EPA weights. We use this alternate server score to rank the systems under test by their expected DVDStore energy-efficiency ranking. Comparing this ranking to the actual DVDStore results and their respective ranking, we gain insights into the applicability of our metric for estimating a server's efficiency in production use. In addition, by modeling real-world applications using our workload scores this analysis demonstrates that relevant worklets and workloads were selected for the methodology implementation.

10.3.1 Deriving Weights

We weigh the SERT workload scores for their similarity with the DVDStore regarding energy efficiency. We do not derive weights using regression or a similar mechanism. Regression would optimize the weights to achieve final weighted scores that mimic the DVDStore scores with the smallest error, which defeats the purpose of our test. Instead, we try to derive weights by investigating how the DVDStore utilizes a system using power measurements and power/performance counters. Using this approach, we get a more "semantic" similarity that focuses on workload behavior similarities.

We collect the power counters p on a per-second basis during each worklet's measurement phase and compute the average counter value for the interval $p(int)$ at the end of the measurement phase. For each worklet and interval, we obtain a vector $\vec{p}(int)$ of all performance counters. Finally, we construct a vector for each workload by averaging the performance counter values measured for all the separate worklets within the workload. We choose the average, as the power and performance counters are not ratios (in contrast to energy efficiency). With three total workloads, we arrive at three vectors for each interval: $\vec{c}(int)$ (CPU), $\vec{m}(int)$ (Memory), and $\vec{s}(int)$ (Storage). Finally, we measure the performance counter vector $\vec{t}(int)$ for each interval of the target workload (DVDStore).

We define workload similarity weights using barycentric coordinates Floater, 2003 that express the normalized distance between our target workload and the SERT workloads. Using the three workload vectors $\vec{c}(int)$, $\vec{m}(int)$, and $\vec{s}(int)$, we are able to span a plane within the n-dimensional space of possible performance counter values (n being the number of power/performance counters and thus the dimension of the vectors). We project the target workload vector $\vec{t}(int)$ onto this plane, arriving at two-dimensional vectors $\vec{c}_{2d}(int)$, $\vec{m}_{2d}(int)$, $\vec{s}_{2d}(int)$ for the SERT workloads and $\vec{t}_{2d}(int)$ for the target workload. $\vec{c}_{2d}(int)$, $\vec{m}_{2d}(int)$, $\vec{s}_{2d}(int)$ can be used to define a triangle. We derive a barycentric coordinate $\vec{b}(int)$ for $\vec{t}_{2d}(int)$ within this triangle. This coordinate shows the normalized relative position of $\vec{t}_{2d}(int)$ in relation to $\vec{c}_{2d}(int)$, $\vec{m}_{2d}(int)$, $\vec{s}_{2d}(int)$.

We choose the following concrete power counters: wall power, measured using the external power analyzer, CPU power, and memory power, measured using Intel's Intel PCM (Willhalm et al., 2012). We measure the SERT and the DVDStore on a reference machine with an eight-core Intel Xeon E5-2640 v3 and 32 GB of RAM. The DVDStore is deployed and driven using the SPEC ChauffeurWDK framework (Arnold, 2013). We arrive at a CPU weight of 0.55, memory weight of 0.26, and storage weight of 0.19.

10.3.2 Ranking Servers for the DVDStore

Table 10.7 shows the emulated and measured DVDStore scores on all tested systems. Note that the SERT results with weights are based on normalized performance results, whereas the actual DVDStore results are not normalized, resulting in the different order of magnitude of the scores.

The servers are ranked in the same order based on the DVDStore scores derived using weighted SERT scores, and based on the actually measured DVDStore scores. This indicates that a ranking based on separate workload scores within the overall metric is helpful for decision makers regarding production deployments and indicates that the worklets within the SERT implementation

Table 10.7: Ranking of measured and weight-simulated DVDStore efficiency scores.

Cores	Idle Power	Max Power	SERT	DVDStore (weights)	DVDStore (measured)	Rank (weights)	Rank (meas.)
4	29.4 W	99.4 W	12.19	13.41	48.23	5	5
4	28.3 W	106.1 W	14.79	16.25	65.12	3	3
8	38.1 W	140.4 W	15.99	16.73	68.92	2	2
10	42.6 W	151.8 W	15.40	15.71	58.15	4	4
2 x 8	98.4 W	423.9 W	9.99	9.73	39.1	6	6
32	53.3 W	238.0 W	39.26	44.41	93.99	1	1

are representative regarding the amount of information that they can provide. Note that the 10-core server in Table 10.7 features an anomaly when comparing the regular SERT score and the DVDStore scores. The server is more efficient than the second quad-core server based on the regular SERT score. However, it is less efficient for both the emulated and measured DVDStore results. This, again, stresses the need for a broader selection of workloads within a benchmarking suite, as opposed to a single application benchmark. It also answers **EQ A.3** ("Does the metric have a relationship with the energy efficiency of real-world workloads?") regarding our partial metrics, showing that they can indeed be used to define relationships to real-world workloads, enhancing relevance.

10.4 Summary

We showed that our methodology exhibits the three primary properties of benchmark quality. We evaluated its reproducibility (answering **EQ A.1**: "Does the methodology produce consistent power and performance results when applying it repeatedly?") by analyzing the coefficients of variation in and between runs of our suite. In our tests, the coefficient of variation for worklet interval efficiency stayed below 1.1%. Next, we analyzed fairness of our metric (answering **EQ A.2**: "Does the metric correctly order systems by their energy efficiency?"). We showed that several improvements in system efficiency lead to an increase in metric score. We also showed that some correlations exist in metric score and system properties that would intuitively be considered more efficient. Finally, we evaluated the relevance of our methodology and metric (answering **EQ A.3**: "Does the metric have a relationship with the energy efficiency of real-world workloads?") by demonstrating that our metric can be used to derive relationships to third-party workloads, enabling decision makers to select energy-efficient systems for their specific domain.

Chapter 11

Evaluation of Load Profiles and Placements

We introduce two methods for extending our energy efficiency rating methodology in Chapter 5. The first is a modeling method for load intensity profiles, which allows specification and extraction of load traces that vary with time. Among other things, this model can be used for testing of power management and optimization mechanisms at run-time. The latter method is an adaptation of our methodology for hierarchical load distribution. This method enables testing of uneven load distribution and heterogeneous workloads.

In this chapter, we evaluate both methods. Addressing **EQ A.4** ("Can the modeling formalism and extractor accurately express the load variations of real-world load profiles?"), the DLIM load profile model's expressiveness is evaluated using its extraction mechanism. We extract model instances from real-world load traces and analyze the extraction's accuracy. Addressing **EQ A.5** ("Can we evaluate the energy efficiency of load distribution policies?"), the workload distribution mechanism, on the other hand, is used for a thorough analysis of system efficiency, in which we analyze how systems behave for different load distribution policies. We also evaluate our own distribution policy, described in Section 5.2.1, as part of this analysis.

11.1 Evaluation of DLIM Load Profile Extraction

We evaluate the Simple DLIM Extraction Method (s-DLIM) extraction method for our DLIM load profile model from Chapter 5, Section 5.1 and address **EQ A.4** ("Can the modeling formalism and extractor accurately express the load variations of real-world load profiles?") using nine different real-world Web server traces covering between two weeks and seven months. The traces all are strongly influenced by human usage patterns.

s-DLIM is applied to extract model instances for all traces. We also separately evaluate the effect of noise extraction including noise reduction. The shape of the interpolating functions is always selected as the DLIM *SinTrend*, meaning that sin-flanks are always used for the interpolation between arrival rate peaks and lows. We choose *SinTrend* because it fits closest to the original trace in the majority of cases. For the same reasons, *ExponentialIncreaseAndDecline* is always selected for *Burst* modeling (it is a child of *Burst* in the DLIM meta-model). *Trends* are selected to be multiplicative since this way they have a lower impact on arrival rate lows and a relatively high impact on arrival rate peaks (contrary to additive *Trends*, which have a constant impact on both). We do this, since arrival rate lows vary less than arrival rate peaks according to our observations.

Extraction is also configured with varying *Trend* lengths. Best results are expected at *Trend* length of one *Seasonal* period, whereas lower accuracy is expected at the longest evaluated *Trend* length of three *Seasonal* periods.

We evaluate the model extraction accuracy by computing the relative errors for each pair of corresponding entries in the extracted model instance and trace. The median of the absolute percentage error values (MdAPE) is presented in this work. In this context, the mean absolute percentage error (MAPE) is prone to deflection by positive outliers. Moreover, we compare the extraction error and run-time on commodity hardware (Core i7 4770, 16 GB RAM) against the BFAST time-series decomposition of (Verbesselt et al., 2010) (which returns split data as opposed to a descriptive model). To enable a fair comparison, we configured BFAST to extract one seasonal pattern and not more than one trend per day. In contrast to DLIM, where seasonal patterns are represented by piece-wise interpolating functions, in BFAST's output, the seasonal pattern is represented as a less compact discrete function.

Table 11.1: Model extraction errors for the Internet Traffic Archive and BibSonomy traces.

Trace	1. **ClarkNet**	2. **NASA**	3. **Saskat.**	4. **WC98**	5. **BibSonomy**
Parameters (Trend, Noise)	**relative median (%)**	**relative median (%)**	**relative median (%)**	**relative median (%)**	**relative median (%)**
length 1, extracted	21.195	26.446	35.551	19.735	26.988
length 1, reduced	17.509	23.56	**26.492**	16.882	**21.479**
length 1, ignored	**12.409**	**18.812**	29.171	**12.979**	23.831
length 2, ignored	14.734	20.8	30.273	15.691	26.786
length 3, ignored	14.919	27.577	32.085	19.161	28.218
BFAST	12.243	no result	no result	no result	no result
s-DLIM (avg. ms)	4.2	25.2	118.8	11.8	125
BFAST (avg. ms)	76276	no result	no result	no result	no result

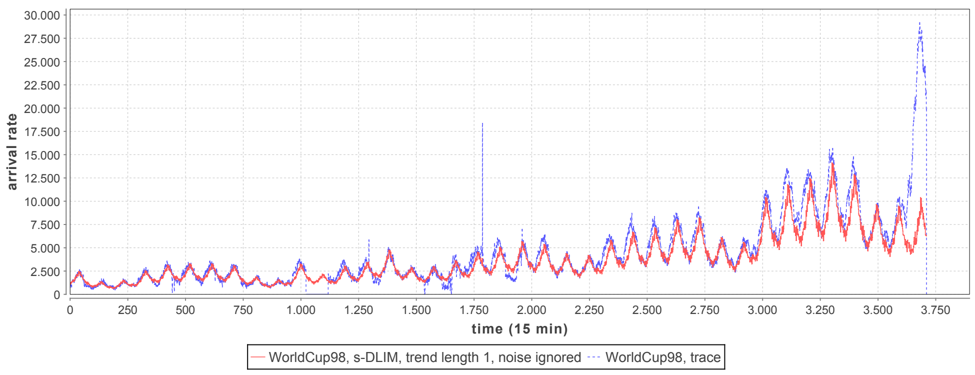


Figure 11.1: Arrival rates of the original WorldCup98 trace (blue) and the extracted DLIM instance (red) using s-DLIM with a *Trend* length of 1 and ignoring noise.

11.1.1 Internet Traffic Archive and BibSonomy Traces

The first batch of traces was retrieved from The Internet Traffic Archive[1]. The Internet Traffic Archive includes the following traces: *ClarkNet-HTTP* (Internet provider WWW server), *NASA-HTTP* (Kennedy Space Center WWW server), *Saskatchewan-HTTP* (University WWW server), and *WorldCup98* (official World Cup 98 WWW servers). Additionally, we used a six week long trace of access times to the social bookmarking system *BibSonomy* (Benz et al., 2010), beginning on May 1st 2011[2]. All traces were parsed to arrival rate traces with a quarter-hourly resolution (96 arrival rate samples per day).

Table 11.1 shows the relative median errors for our extraction using different configurations. It also displays run-time of the overall most accurate extraction configuration (ignoring noise, trend length 1) as an average value over ten runs. In cases in which BFAST decomposition terminated, errors and run-times are also displayed for BFAST. The ClarkNet and NASA extraction results show that a *Trend* length of 1 provides the best accuracy. Noise reduction does not seem to help for this particular trace during the DLIM extraction. The result does not improve when extracting a noise distribution model, as noise generated by a random variable does not reproduce the exact measured results and increases the absolute arrival rate difference between trace and model. We trace the

[1]Internet Traffic Archive: `http://ita.ee.lbl.gov/`

[2]The request log dataset is obtainable on request for research purposes: `http://www.kde.cs.uni-kassel.de/bibsonomy/dumps/`

discrepancies between the extracted model instance and the original trace to three major causes:

- In some cases, bursts are not detected with full accuracy.
- The NASA server was shut down for maintenance between time-stamps 2700 and 2900. The extraction method does not have contingencies for this case, as we only extract positive bursts.
- Deviating Seasonal patterns are a major cause of inaccuracy in the extracted models. The extraction method assumes a single, repeating *Seasonal Part*. Depending on the trace, this assumption may be valid to a different extent. In this case, the extracted Seasonal pattern is able to approximate most days in the trace, but a number of significant deviations occur. Manual modeling in the DLIM editor can circumvent this problem, as DLIM itself supports mixes of multiple seasonal patterns. We are currently working on extending the automated extractors to make use of this feature. Ideas range from the inclusion of additional meta-knowledge, such as calendar information, to the implementation of seasonal break detection, as introduced in BFAST (see Verbesselt et al., 2010).

In the case of the Saskatchewan-HTTP extraction, noise reduction improves the s-DLIM results. However, overall the results are not as good as they are for the other traces. The major explanation for the relatively poor results is once more the *Seasonal* pattern deviation. Since the Saskatchewan-HTTP trace extends over 7 months, the *Seasonal* patterns have a lot of room for deviation. The model extractor fails to capture this. This leads to an additional error in the *Trend* calibration, as trends are supposed to be calibrated, so that the greatest *Seasonal* peak in every *Seasonal* iteration matches the trace's nearest local arrival rate maximum. Since the *Seasonal* pattern deviation causes the extracted *Seasonal* peak's time of day to not match the trace's *Seasonal* peak's time of day, the calibration takes place at the wrong point of time. This also explains why a majority of extracted days have a lower peak then their counterparts in the original trace.

The major deviation from the trace's *Seasonal* patterns also explains why s-DLIM performs better using noise reduction for the Saskatchewan-HTTP extraction. Noise reduction helps to mitigate the effect of seasonal pattern changes over time, thus reducing the effect of the *Seasonal* pattern deviation.

Similarly to the Saskatchewan trace, s-DLIM extraction of the BibSonomy trace also improves with noise filtering. We explain this through the observation that the BibSonomy trace features a significant number of bursts, occurring at

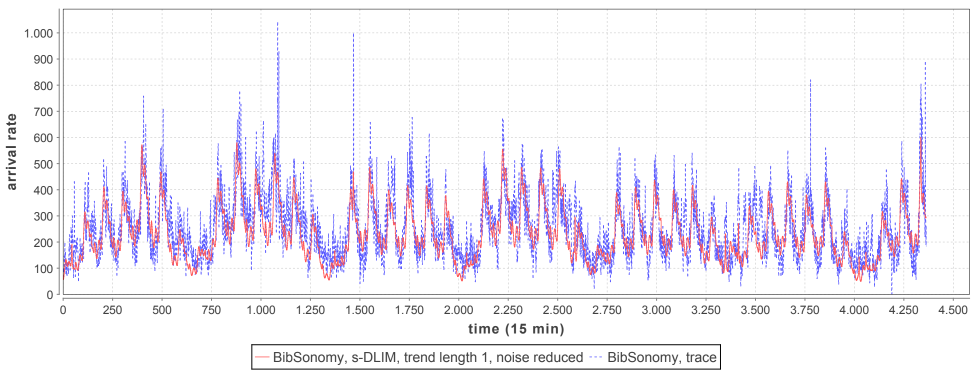


Figure 11.2: Arrival rates of the original BibSonomy trace (blue) and the extracted DLIM instance (red) using s-DLIM with *Trend* length 1 and noise reduction.

a relatively high frequency, as well as significant noise (as seen in Fig. 11.2). Without filtering, some of these bursts are included in the seasonal pattern by the s-DLIM extractor, distorting the extracted seasonal pattern. When applying noise reduction, the influence of these bursts is diminished. Therefore, the extracted seasonal pattern is more stable, leading to increased accuracy as major bursts are still extracted during s-DLIMs burst extraction. The BibSonomy trace demonstrates that s-DLIM is capable of dealing with traces featuring a significant amount of noise.

Comparing the accuracy of our extraction methods with that of BFAST proves difficult, given that, for four of the five considered traces, BFAST did not terminate within 1.5 hours, which would make BFAST execution at least 45000 times slower than s-DLIM in these cases. However, the ClarkNet trace extraction shows that our extraction methods exhibit accuracy comparable to the accuracy of BFAST in cases where BFAST terminates in a reasonable amount of time. To eliminate the possibility of BFAST not terminating due to configuration errors on our side, we ran the same configuration on shortened versions of the respective traces. BFAST's time-series analysis of these shortened traces terminated, however, the latter are too short to meet our criteria of sufficiently long traces with recurring seasonal patterns and trends.

Table 11.2: wikipedia.org model extraction errors.

Trace	1. **German Wikipedia**	2. **French Wikipedia**	3. **Russian Wikipedia**	4. **English Wikipedia**
Parameters (Trend, Noise)	**relative median (%)**	**relative median (%)**	**relative median (%)**	**relative median (%)**
length 1, extracted	11.215	10.472	9.964	7.764
length 1, eliminated	10.511	8.566	**9.912**	7.838
length 1, ignored	**8.538**	**7.6**	11.251	**4.855**
length 2, ignored	9.956	8.973	11.683	5.27
length 3, ignored	11.771	9.813	11.42	7.23
BFAST	11.223	8.511	5.809	2.302
s-DLIM (avg. ms)	3.9	3.5	5.8	3.2
BFAST (avg. ms)	23518	23630	23803	21517

11.1.2 Wikipedia Traces

The second batch of traces was retrieved from the Wikipedia page view statistics[3]. They were parsed from the *projectcount* dumps, which already feature arrival rates with an hourly resolution. We restrict our analysis to the English, French, German and Russian Wikipedia projects, covering four of the six most requested Wikipedia projects and being distributed over different time-zones. All traces are from December 2013, with the exception of the English Wikipedia trace, which is from November 2013. The English December 2013 trace exhibits a major irregularity during the 4th day, which we attribute to a measurement or parsing error. While the French, German, and Russian Wikipedia projects are mostly accessed from a single time zone, the English Wikipedia is retrieved from all over the world. Thus, evaluating the impact of access behavior over different time zones and helping to assess how well the DLIM extraction methods deal with local vs. global access patterns.

The Wikipedia extraction results in Table 11.2 confirm many of the observations made with the Internet Traffic Archive traces. E.g., *Trend* length of 1 performs best. The overall accuracy, however, is significantly better than for the Internet Traffic Archive traces since the *Seasonal* pattern deviation, while still relevant, exhibits less impact than before.

The Russian Wikipedia trace differs from the other Wikipedia traces. Noise reduction improves s-DLIM. The overall accuracy is similar to the other Wikipedia trace extractions. For this single trace, however, the *Seasonal* patterns are shaped in such a way that the noise reduction lessens the impact of the *Seasonal* pattern deviation.

[3]Wikipedia traces: `http://dumps.wikimedia.org/other/ pagecounts-raw/2013/`

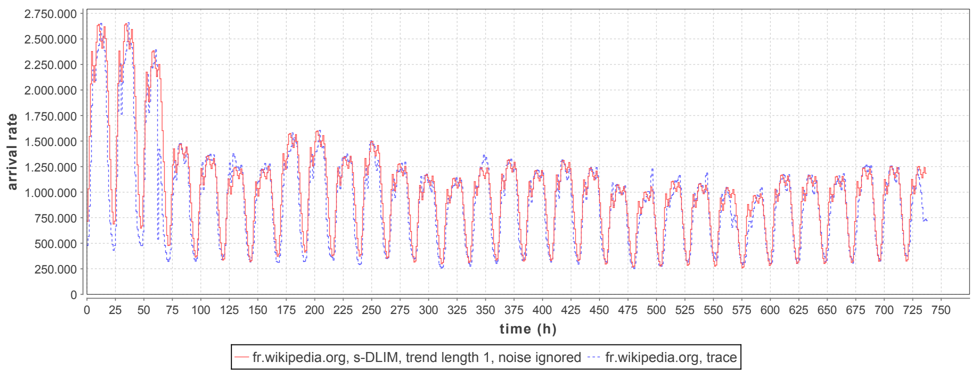


Figure 11.3: Arrival rates of the original French Wikipedia trace (blue) and the extracted DLIM instance (red) using s-DLIM with *Trend* length 1 and ignoring noise.

The extraction results for the English Wikipedia trace exhibit by far the best overall accuracy across all examined traces. The reason for this is the unusually high arrival rate base level. Since wikipedia.org is accessed globally at all times, the load intensity variations on top of the base level have little impact on the relative load variations in general. As a result, all modeling errors are also relatively small.

In terms of accuracy, our extraction process performs as well as the BFAST decompositions. s-DLIM performs better than BFAST for both the German and French Wikipedia traces (e.g., see Fig. 11.3). Here, s-DLIM's accuracy profits from its support of multiplicative trends. BFAST does, however, provide better accuracy for the English and Russian traces. In these cases BFAST's sophisticated trend calibration mechanisms outperform s-DLIM. s-DLIM is, however, significantly faster than BFAST. Running on the same machine, LIMBO's s-DLIM implementation performed on average 8354 times faster than BFAST's R implementation and returned results in all cases in less than 0.2 seconds.

Summarizing, our evaluation of the DLIM model extractor's accuracy answers **EQ A.4** ("Can the modeling formalism and extractor accurately express the load variations of real-world load profiles?"). It shows that DLIM is an expressive modeling mechanism that is able to capture load profiles with great accuracy. The extractor is able to extract load intensity models with a median modeling error of 12.7% from a representative set of nine different real-world traces. It is also able to perform these extractions quickly, as each extraction completes in less than 0.2 seconds. The major source of errors in models is seasonal drift in original traces. The “s-DLIM” model extractor, used in this work, does not yet have a method to account for this drift. In general, the results demonstrate and validate the capability of DLIM to capture realistic load intensity profiles.

11.2 Energy Efficiency of Hierarchical Load Distribution

We address **EQ A.5** ("Can we evaluate the energy efficiency of load distribution policies?") by evaluating the three load distribution strategies using our adapted methodology on a range of systems with increasing complexity: a one-socket system, two dual-socket systems of different architectures and two dual-socket systems running the workload in a multi-node configuration. For each of these systems, we test a variety of combinations applying different load distribution strategies on the different levels of the execution hierarchy (nodes, sockets, physical CPU cores, logical CPU cores).

As this part of our work focuses on distribution strategies for server loads, we employ those worklets within our methodology's SERT implementation that offer opportunities to be distributed in multiple ways. As a result, we use worklets with at least some CPU-bound work, as this enables core-wise placement of the worklet's load. For a more detailed analysis, we use 10% increments in load levels. The following worklets are the core worklets employed in our measurements: *CryptoAES*, *LU*, *XMLValidate* (a variant of the *Capacity* worklet that scales with transaction rates instead of memory size), and *SSJ*.

We evaluate the different load distribution strategies on four systems: A Fujitsu PRIMERGY TX1310 M1 machine is used as our SUT for the evaluation of physical core and SMT unit distribution strategies on a one socket system. The machine is equipped with an Intel Xeon E3-1230 v3 CPU (Haswell) featuring a base frequency of 3.3 GHz (up to 3.9 GHz with Turbo) and four DIMMs of DDR3 RAM. Power consumption of this machine is measured using a ZES Zimmer LMG95 power analyzer. Multi-socket load distribution strategies are tested on a Dell PowerEdge R720 and Dell PowerEdge R730 system. Both systems are equipped with 2 CPU sockets and eight DIMMs of RAM each. The Ivy Bridge based PowerEdge R720 system features two Intel Xeon E5-2667 v2 processors with a base frequency of 3.3 GHz (up to 4.0 GHz with Turbo), whereas the Haswell based PowerEdge R730 system is equipped with two Intel Xeon E4-2667 v3 CPUs with a base frequency of 3.2 GHz (up to 3.6 GHz with Turbo). The AC wall power for both of the dual-socket systems is measured using a Yokogawa WT210 power analyzer. Finally, we evaluate multi-node load distribution strategies by simulating multi-node results using the Dell systems and through separate measurements on a cluster featuring two HP ProLiant BL260a Gen9 Blades with two 18-core Intel Xeon E5-2699 v3 CPUs each. The blades differ from the other systems as they make use of shared PSUs, provisioned to support up to 16 servers. CPU frequency scaling (including Turbo) and all other BIOS power saving mechanisms have been turned on. For these experiments, we use Windows Server 2012 R2 as the operating system.

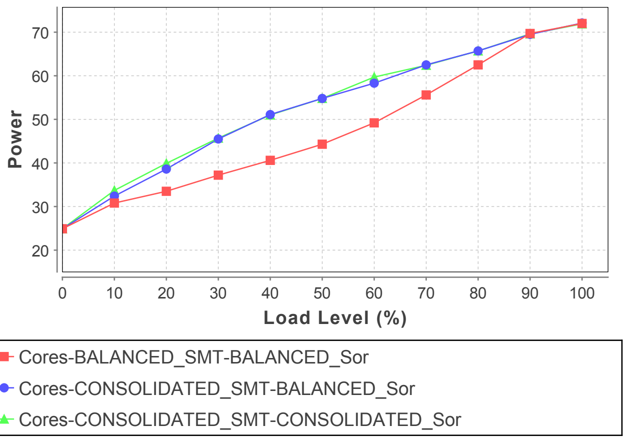


Figure 11.4: Power consumption of distributions for SOR on single-socket system.

11.2.1 Energy Efficiency for Homogeneous Workloads

We evaluate the power consumption and energy efficiency of hierarchical load distribution for homogeneous workloads on multiple systems. Firstly, we evaluate different combinations of load distribution on the core and logical processor level using a single-socket system. The balancing and consolidation strategies are evaluated for all worklets, and the efficiency strategy is applied for the LU worklet. We then test the impact of socket-level load distribution on Ivy Bridge and Haswell systems, introducing additional load distribution combinations with the added level in the distribution hierarchy. We also evaluate load distribution with operating system level load migration, before finally investigating the effects of node level load management.

11.2.1.1 Load Distribution for all Worklets on Single-Socket System

Load distribution is achieved by varying the target throughput per logical processor with the goal of achieving the pre-specified global load level. This target load level is achieved with consistency for all distribution strategies. The coefficient of variation (CV) for any given worklet throughput at a target load level never exceeds 3.3%, with an average CV of 0.7%. As a result, energy efficiency of the distributions is primarily influenced by power consumption.

The load distribution strategies' power consumption is shown in Figure 11.4. Simultaneous Multi-Threading (SMT) is turned on for these measurements.

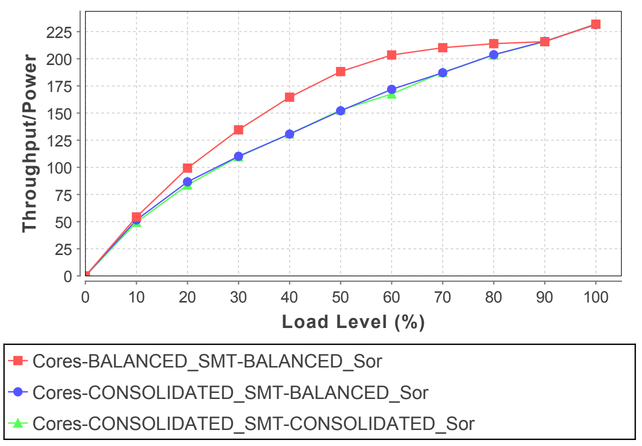


Figure 11.5: Energy Efficiency of distributions for SOR on single-socket system.

The power consumption can differ significantly depending on load level. Specifically, the balanced distribution outperforms the consolidation strategies between 30% and 70% load. This observation can be explained by the CPUs frequency scaling. State of the art processors feature dynamic frequency scaling mechanisms designed to increase performance on a subset of available cores in case of uneven load distribution. This feature is usually referred to as "Turbo" and causes single CPU cores to dynamically overclock as long as a number of thermal and power constraints are met. Core-wise load consolidation causes single cores to quickly reach a load level that triggers the core's turbo. In the case of our quad-core system, a global load level of 20% already causes a local load level of 80% on core 0. At this point, power consumption diverges and energy efficiency drops in comparison to a balanced load distribution (as can be seen in Figure 11.5).

However, power consumption and energy efficiency are not affected significantly by load consolidation on the level of SMT units. While SMT threads are provided and executed in hardware, they are not pinned to an exclusive set of execution units in the same sense as a physical CPU core. Instead, logical processors on the same core share their execution units. Subsequently, redistributing load between those logical processors on the same core has little impact for homogeneous loads, as the same execution units remain in use.

We were able to repeat the observations made for the SOR worklet for all other worklets. All worklets exhibited maximum power consumption when using consolidation strategies and maximum energy efficiency when using a balanc-

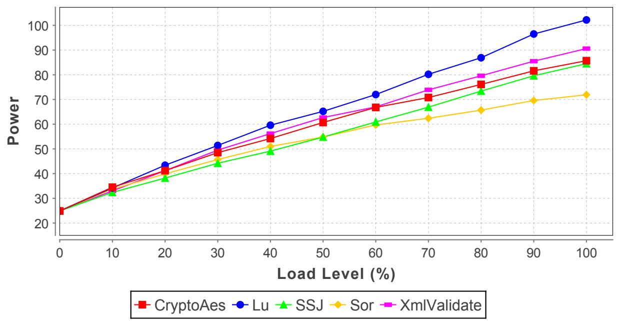


Figure 11.6: Power consumption of fully consolidated load on single-socket system.

ing strategy. Power consumption for all worklets using the fully consolidated strategy is displayed in Figure 11.6. The worklet's order in power consumption remains identical over utilization levels and distribution strategies, with one exception. As the SSJ worklet is not a purely CPU bound workload, but a hybrid workload emulating typical business transaction software, it contains a number of non-CPU bottlenecks. As a result, it scales differently with increasing load compared to the CPU bound worklets. It is also affected differently by the selected load distribution strategy. Specifically, it always consumes more power than SOR using the fully balanced strategy, yet consumes less power between the 10% and 50% load levels using the fully consolidated strategy.

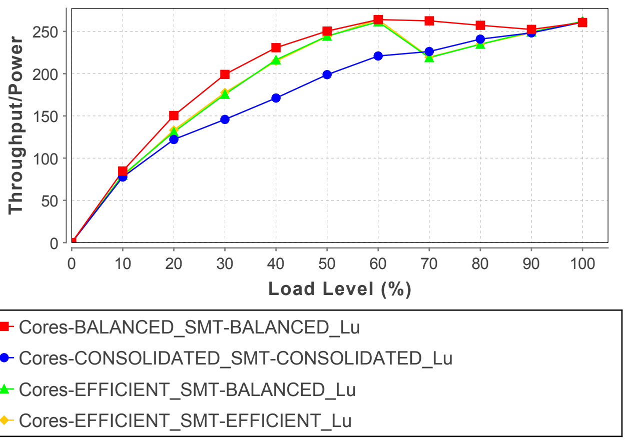


Figure 11.7: Power consumption of efficiency strategies on single socket system.

11.2.1.2 Efficiency Strategy on Single-Socket System

We evaluate the efficiency strategy using the LU worklet, as LU reaches the point of optimal energy consumption on a load level less than 100% on the majority of systems. On the single-socket system the load level for maximum energy efficiency with LU is 60% with 264.0 transactions/watt. However, energy efficiency at full load is only slightly worse at 260.5 transactions/Watt. The new strategy is not as efficient as the balanced strategy on the single-socket system (note: this is in contrast to the multi-socket system in Section 11.2.1.4). It is, however, more efficient than full load consolidation, confirming the energy inefficiency of forced turbo overclocking through load consolidation. Once the point of maximum energy efficiency is passed, the strategy allows consolidation of load up to 100%, thus displaying the power consumption of the consolidation strategies.

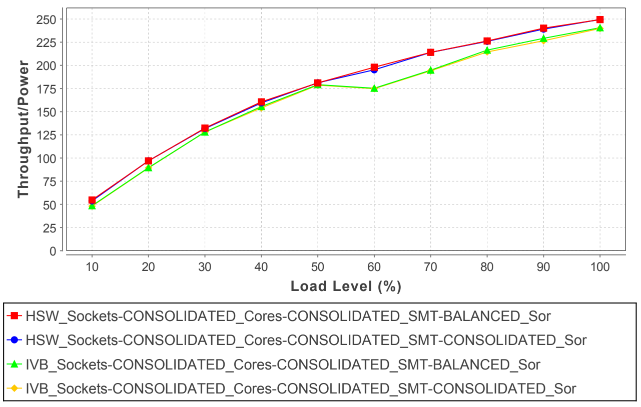


Figure 11.8: Energy efficiency of consolidated strategy with and without SMT unit consolidation on both dual-socket systems for SOR.

11.2.1.3 Load Distribution for all Worklets on Dual-Socket System

The dual-socket systems offer an additional layer on the load distribution hierarchy, as load can be distributed on a socket-by-socket basis. Performance variation depending on the selected strategy remains minimal with an average CV of 0.4% on the Haswell and 0.5% on the Ivy Bridge System.

Measurements on the dual-socket systems also confirm that load distribution on SMT units within the same core has no effect on power consumption and energy efficiency. This observation remains true for both Haswell and Ivy

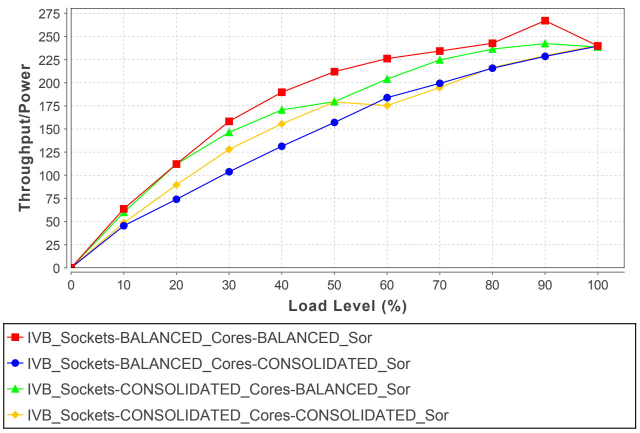


Figure 11.9: Energy efficiency of load distribution strategies on Ivy Bridge dual-socket systems for SOR.

Bridge architectures across all worklets and strategies. A selected comparison is shown in Figure 11.8. As a result, we will only display the results for strategy compositions using a balancing strategy on the SMT level for all following measurements.

The dual-socket Ivy Bridge measurement results in Figure 11.9 are representative for the results with all workloads. Keeping the workloads fully balanced on all levels is still the most efficient strategy, as it was for the single-socket system. However, balancing is not always the most efficient sub-strategy in all cases. When core-level load consolidation is employed, load consolidation on a socket level improves energy efficiency at lower load levels, meaning that full consolidation on both the socket and CPU levels beats a CPU level consolidation on balanced sockets. This can be explained by the observation that the amount of cores onto which work is consolidated for both of these strategies remains the same. The difference is only in the location of the given cores. When consolidating sockets, only cores on one socket are activated, allowing cores on the other socket to remain in more energy efficient modes and, more importantly, keeping them from going into turbo. Balancing cores on consolidated sockets improves energy efficiency further, as cores are kept in their most energy efficient load range.

The dual-socket Haswell measurement results in Figure 11.10 are the first results to significantly deviate from one of our principal previous observations, as the balanced strategy is not the reliably best strategy at all load levels. Socket level load balancing improves energy efficiency for all workloads making it

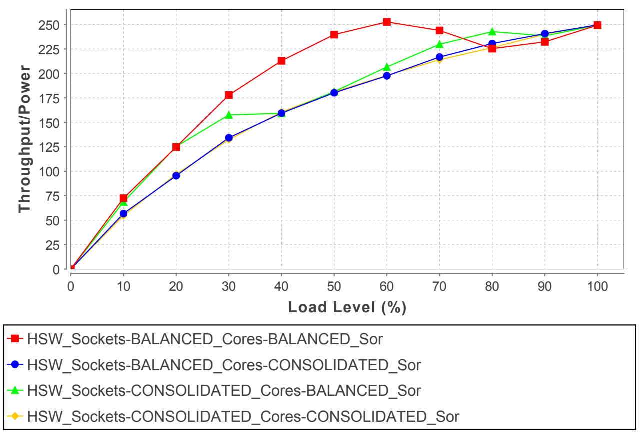


Figure 11.10: Energy efficiency of load distribution strategies on Haswell dual-socket systems for SOR.

more efficient than pure load balancing at high load. This effect is visible to a different extent depending on the specific workload. It can differ both in the range of load levels for which consolidation is superior to balancing, as well in the amount of energy saved. A comparison of these factors is shown in Table 11.3.

Worklet	**Load level range**	**Avg. power difference**	**Max. power difference**
CryptoAES	80% – 90%	12.3 W	16.3 W
LU	80% – 90%	30.0 W	34.4 W
SOR	80% – 90%	11.5 W	16.6 W
SSJ	90%	1.6 W	1.6 W
XMLValidate	90%	10.3 W	10.3 W

Table 11.3: Load level range where socket consolidation has a lower power consumption than fully balanced load (inclusive). The shown differences are power consumption differences within this range.

Core-level load consolidation is always less efficient than the same strategy composition without core level consolidation. However, some workloads (SOR, LU, and CryptoAES) gain such a significant increase in energy efficiency from using socket level consolidation that consolidation on all levels still outperforms

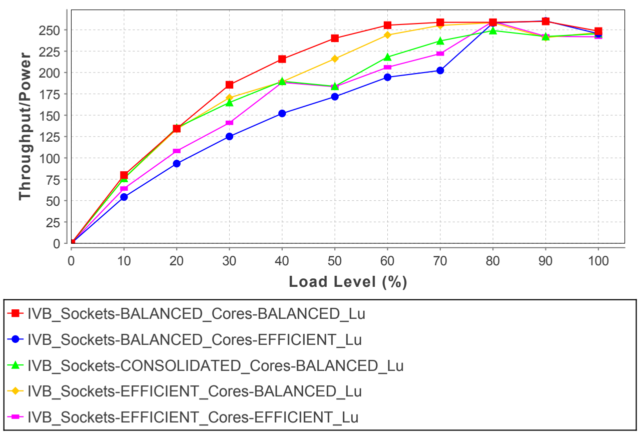


Figure 11.11: Energy efficiency of load distribution strategies on Ivy Bridge dual-socket system for LU.

the fully balanced strategy at high load. Similarly, these worklets also profit from core-wise load balancing at high load: It is, however, not as efficient as socket level consolidation only.

11.2.1.4 Efficiency Strategy on Dual-Socket System

Effectiveness of the efficiency strategies varies depending on processor architecture. On Ivy Bridge, maximum energy efficiency for LU is reached at the 80% load level (see Figure 11.11). Socket-level load distribution using the efficiency strategy features better energy efficiency than full socket-level consolidation, yet still doesn't reach the energy efficiency of a fully balanced system.

The Haswell dual-socket system reacts in a completely different way to the efficiency strategies, as shown in Figure 11.12. Particularly, the socket level efficiency strategy features better energy efficiency than the fully balanced distribution and the socket level consolidation strategy at a number of load levels. LU's load level of maximum energy efficiency is 50%. Per definition, the efficiency strategy and the fully balanced strategy result in an identical load distribution at this load level. At lower load than 50% the fully balanced approach still remains the most energy efficient. At those load levels, the efficiency strategy attempts to keep as many sockets as possible at 50% load, which leads to partial load consolidation and consumes more power than a balanced load at less than 50%. Beyond 50% load, the efficiency strategies attempt to consolidate work on as few units as possible, while keeping the

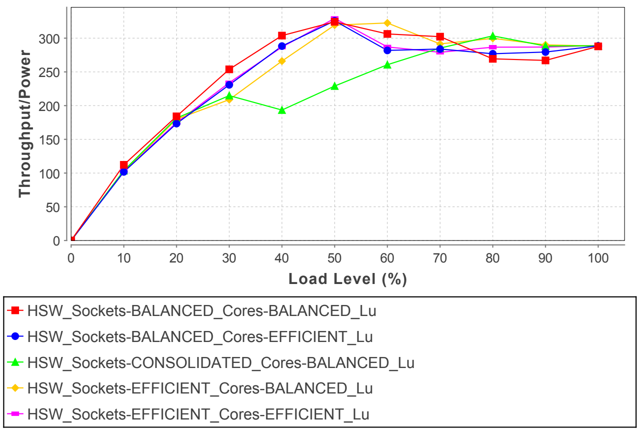


Figure 11.12: Energy efficiency of load distribution strategies on Haswell dual-socket system for LU.

rest at 50%. On the socket level this means that one socket is kept at 50% load, while the other increases in load. Beyond the 75% load level, the first socket is fully utilized and the efficiency strategy behaves identical to the consolidation strategy, as it can now only increase load on the remaining socket. At 60% load the strategy of keeping a socket at maximum energy efficiency pays off, resulting in greater efficiency than using pure balancing. Beginning at the 70% global load level the socket level efficiency strategy starts distributing load similarly (and then identically) to the socket level consolidation strategy. Consequently, the energy efficiency of these two strategies behaves the same.

On our dual-socket Haswell system, the efficiency load distribution strategy saves up to 33 W (10.7% relative savings) in comparison to the fully balanced strategy. Within the interval between maximum energy efficiency (50%) and full load (both exclusive) it allows for an average energy saving of 15.4 W (5.8% relative savings). In conclusion, we recommend a mixed load balancing strategy for a multi-socket Haswell system. We recommend a fully balanced strategy up to the load level of maximum energy efficiency. At greater loads our efficiency strategy performs better and should be used instead.

11.2.1.5 No CPU Pinning

The OS features additional mechanisms, which can be used once CPU pinning is disabled. Specifically, it is able to migrate running threads from one core to another. This way the local CPU utilization can be changed to a level, which is

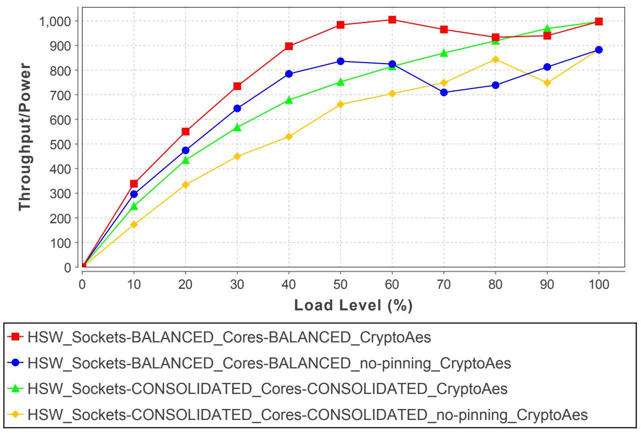


Figure 11.13: Energy efficiency of load distribution strategies, including non-pinning strategies, on Haswell dual-socket system for CryptoAES.

seemingly independent of the actual thread loads, as threads can be continuously relocated at run-time. We disable CPU pinning to analyze the effect of this behavior on power consumption.

For the previous measurements using CPU affinity, most observations were observable over all worklets and only varied in their respective impact. Non-pinned distribution shows far greater variability depending on the workload. Figure 11.13 shows the energy efficiency of a few selected strategies using the CryptoAES worklet. For this specific worklet, non-pinned strategies cannot match the energy efficiency of the fully balanced strategy using CPU pinning. There are still significant differences in the energy efficiency of the different non-pinning strategies, as thread migration does not cancel unevenly distributed load.

LU (see Figure 11.14) behaves completely different, as the non-pinned balanced strategy performs with greater energy efficiency than the pinned strategy at high loads. It even outperforms the efficiency strategy at 60% load. It does, however, not perform as efficiently as the efficiency strategy at loads of 80% and greater. For the other worklets, which were not evaluated for the efficiency strategy, the non-pinned balanced strategy performs equal to (SOR) or better (SSJ, XMLValidate) than its pinned counterpart. The energy efficiency of pinned socket-level consolidation at 90% load is always greatest, though.

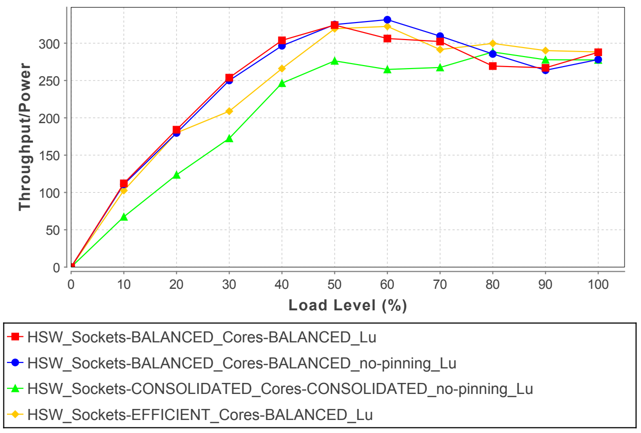


Figure 11.14: Energy efficiency of load distribution strategies, including non-pinning strategies, on Haswell dual-socket system for LU.

11.2.1.6 Multi-Node Load Distribution

In contrast to hardware components within a system, separate server nodes executing independent units of work without any common external resources do not feature any contention. As a result, we can evaluate the energy efficiency of multi-node distribution strategies by simulating multi-node performance and power from single system measurements. For each global load level, we calculate the load that must be reached by any single node within the cluster. We then read the measured power and performance data for the given local load level and compute cluster wide power and performance. Performance variation is again negligible with an average CV of 0.2% over all strategies.

Figure 11.15 shows the energy efficiency of multi-node distribution strategies when unused systems are turned off. Figure 11.16, on the other hand, assumes that unused nodes remain in an idle state. Most notably, node level load consolidation only leads to maximum energy efficiency when unused nodes are switched off. Even then it is only the best strategy at low load up to 30 – 40%. In the 40% to 70% load range it is less efficient than a number of strategies, including balancing and efficiency strategies. At the highest load levels it shares maximum efficiency with node level balancing in combination with the socket level efficiency distribution. Load consolidation loses its advantage at low loads once unused systems are left idling. In that case it is only slightly more efficient than load balancing and equally as efficient as the node-level efficiency strategy at 20% load.

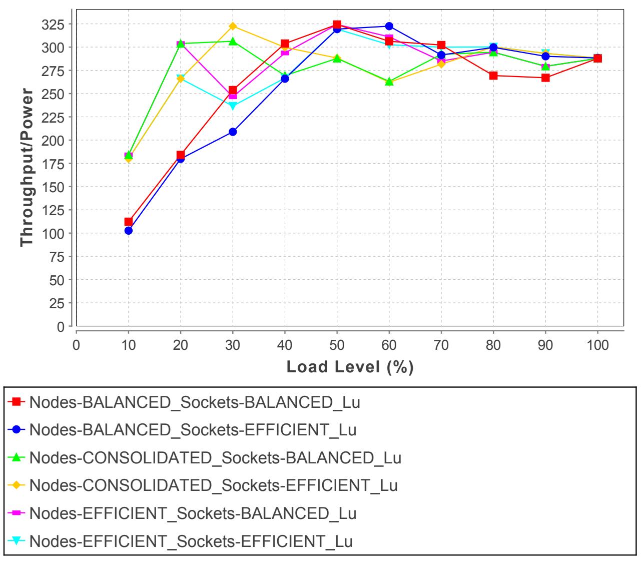


Figure 11.15: Energy efficiency of load distribution strategies on two nodes using the Haswell dual-socket system for LU. Unused nodes are turned off.

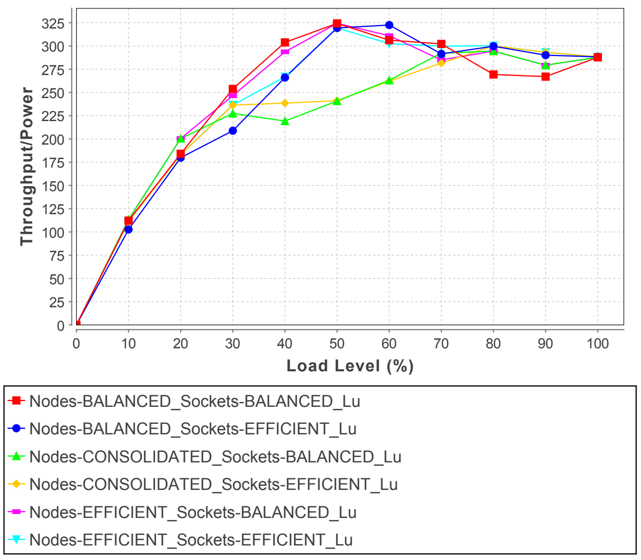


Figure 11.16: Energy efficiency of load distribution strategies on two nodes using the Haswell dual-socket system for LU. Unused nodes remain idle.

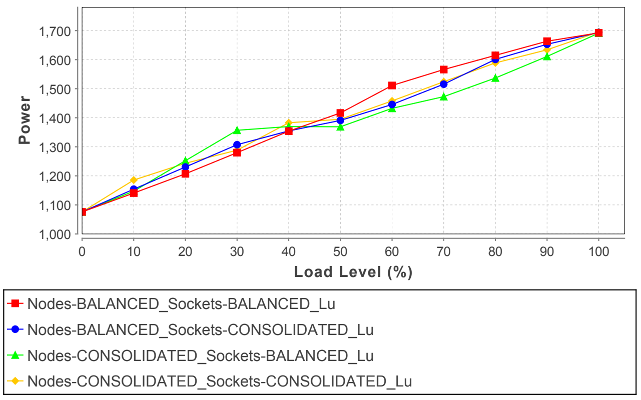


Figure 11.17: Power consumption of load distribution strategies on two HP dual-socket nodes for LU. Unused nodes remain idle.

Figure 11.17 shows the power consumption of the multi-node results, as measured on our two HP servers. These results differ significantly from the previously simulated results. On these systems load consolidation consumes more power than balanced load at low load levels, whereas it saves power at higher loads. We attribute most of this difference to the Xeon E5-2699 v3 CPUs, which operate within the HP servers. These CPUs have a lower frequency (2.3 GHz) in comparison to the Xeon E5-2667 v3 CPU (3.2 GHz) yet feature 18 physical cores per socket in comparison to the 8 cores of the Xeon E5-2667 v3. Another influencing factor for the blade's power consumption is the shared power infrastructure, including shared PSUs.

11.2.1.7 Homogeneous Workloads - Conclusions

The measurement results using homogeneous workloads show that our new efficiency load distribution strategy, applied on the socket level, can reduce power consumption of servers at high utilization. To a lesser degree, load consolidation on the socket level can also improve energy efficiency at high utilization. In contrast, a balanced load is the best choice for low load levels and the CPU core level. Multi-node systems benefit from load consolidation at low loads as unused systems can enter power saving states or shut down. We also demonstrate that CPU-pinned strategies perform equally or better than their non-pinned counterparts.

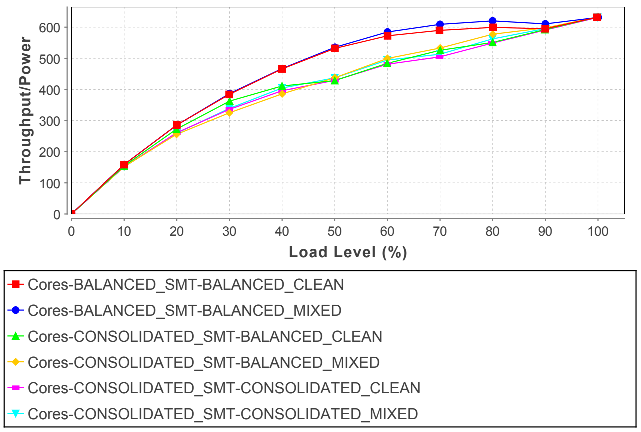


Figure 11.18: Energy efficiency of heterogeneous load distribution strategies on Haswell single-socket system.

11.2.2 Energy Efficiency for Heterogeneous Workloads

We evaluate the energy efficiency of concurrently executing independent workloads using the previously introduced distribution strategies. In addition, we allow the specific worklets to be pinned to logical processors on given cores. As each core on our SUT's features two logical processors, worklets can be either deployed on a core with a worklet of a different type already running on it (**mixed**) or it can be deployed on a core with another instance of the same worklet running concurrently (**clean**). For our experiments, we use the SOR and CryptoAES worklets, as they are sufficiently heterogeneous featuring different performance and energy bottlenecks.

Figure 11.18 shows the energy efficiency for the different distribution strategies on the single-socket Haswell based system. Performance impact of the distribution strategies is very small again, with an average CV of 0.1% for each load level over the different strategies, including mixed and clean strategies. Power consumption is influenced, however, resulting in different efficiency for the strategies. For the balanced strategy, mixing worklets onto the same core has a positive effect on efficiency, as hyperthreading distributes used and unused execution units in a more efficient manner. Heterogeneous workloads offer more options for efficient hyperthreading as heterogeneous workloads tend to use less of the same execution units, enabling concurrent hardware multi-threading on the same core. Heterogeneous load distribution on the dual-socket system (see Figure 11.19) confirms observations from the single-

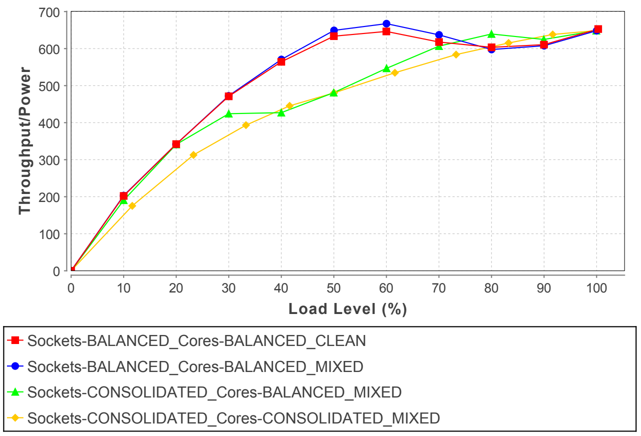


Figure 11.19: Energy efficiency of heterogeneous load distribution strategies on Haswell dual-socket system.

socket result. Mixed execution of worklets remains more energy efficient for the balanced strategy. Additionally, balanced load distribution is still less energy efficient than consolidated loads at high utilization, as it was for homogeneous workloads.

Concluding, the observations made for homogeneous workloads remain valid for heterogeneous workloads. In addition, energy efficiency can be improved by deploying different workload types onto the same core.

Summarizing, we answer **EQ A.5** ("Can we evaluate the energy efficiency of load distribution policies?") by demonstrating the impact of hierarchical load distribution on the energy efficiency of both homogeneous and heterogeneous workloads. We evaluate load distribution and load distribution policies on four levels of the computational hierarchy (servers, sockets, CPU cores, and SMT units). We show that the selection of a single most energy-efficient strategy is only possible on smaller or older systems. For other systems the most energy efficient load distribution strategy depends on workload type and load level. For some loads, the most efficient strategy is not always the commonly assumed one, for example, full load consolidation on a multi-node level can have a smaller energy efficiency than other load distributions. We also show that our new strategy can save up to 10.7% of power consumption on a single server node.

11.3 Summary

This chapter addresses **EQ A.4** ("Can the modeling formalism and extractor accurately express the load variations of real-world load profiles?") and **EQ A.5** ("Can we evaluate the energy efficiency of load distribution policies?") by evaluating our load profile modeling formalisms and extractors and load distribution policies.

Our evaluation of the DLIM model extractor's accuracy answers **EQ A.4** and shows that DLIM is an expressive modeling mechanism that is able to capture load profiles with great accuracy. The extractor is able to extract load intensity models with a median modeling error of 12.7% from a representative set of nine different real-world traces. It is also able to perform these extractions quickly, as each extraction completes in less than 0.2 seconds. The major source of errors in models is seasonal drift in original traces. The "s-DLIM" model extractor, used in this work, does not yet have a method to account for this drift. In general, the results demonstrate and validate the capability of DLIM to capture realistic load intensity profiles.

We answer **EQ A.5** by demonstrating the impact of hierarchical load distribution on the energy efficiency of both homogeneous and heterogeneous workloads. We evaluate load distribution and load distribution policies on four levels of the computational hierarchy (servers, sockets, CPU cores, and SMT units). We show that the selection of a single most energy-efficient strategy is only possible on smaller or older systems. For other systems the most energy efficient load distribution strategy depends on workload type and load level. For some loads, the most efficient strategy is not always the commonly assumed one, e.g., full load consolidation on a multi-node level can have a smaller energy efficiency than other load distributions. We also show that our new strategy can save up to 10.7% of power consumption on a single server node.

Chapter 12

Accuracy and Applicability of Workloads for Energy Efficiency Measurement

This chapter evaluates the advanced workloads introduced in Chapter 6. We evaluate the ability of performance event triggers of Section 6.1 to emulate an application's power profile using multiple reference workloads in Section 12.1. This evaluation addresses **EQ A.6** ("Can we accurately emulate the power profile of more complex workloads using performance counters?"). We show the use of the TeaStore reference and test application of Section 6.2 in three application scenarios in Section 12.2, with one of these scenarios addressing **EQ A.7** ("Can our distributed reference application be used to evaluate the quality of micro-service placements regarding energy efficiency and power?").

12.1 Power-Profile emulation using Performance Event Triggers

We address **EQ A.6** ("Can we accurately emulate the power profile of more complex workloads using performance counters?") and evaluate our approach for power emulation using performance event triggers introduced in Chapter 6.1 by measuring each trigger implementation by itself to determine its accuracy. Each trigger's most viable solution is incorporated into PET for use in the final workload compositions. This final framework is then evaluated for its ability to emulate the power consumption behavior of different server workloads with increasing complexity. For each of the workloads, measurements are taken for parallel and sequential versions with one, four and eight processes. As L2 cache misses are triggered as L3 hits, L2 misses are implicitly evaluated using L3 hits.

12.1.1 Accuracy of Performance Event Triggers

We evaluate the event trigger implementations based on their ability to reach pre-defined constant event counter numbers. Cache counters are evaluated

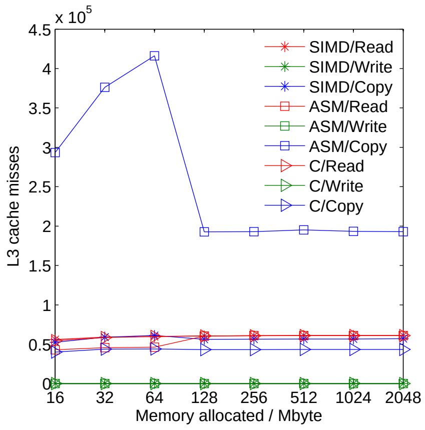


Figure 12.1: L3 cache misses for one process, step size 2 and target value 1×10^6.

based on their ability to accurately reach a target event count of 1×10^6. As memory access will always result in a complete cache line being read or written, memory read and write triggers must reach a total byte count of 64×10^6. For instructions retired, a value of 1×10^{10} is set as the target event count. Context switches and interrupts are set to a target value of 1×10^5. For multiple triggers of the same type running in parallel, target values must be reached for each parallel process. Performance counters are measured on an idle system for $120\,\mathrm{s}$ to establish the background noise of the system.

L3 Cache Misses The evaluation of non-parallelized L3 cache miss triggers with sequential memory access is shown in Figure 12.1 and 12.2. Except for the ASM read implementation, sequential access with varying step sizes performs poorly. Write functions in particular generate almost no cache misses. Only ASM read features a small deviation of less than -3.5% for memory sizes larger than $512\,\mathrm{MiB}$ with a step size of 6.

If a random factor is added to the step size, an overall improvement can be observed, as shown in Figure 12.3, especially for step sizes larger than 2. Increasing the step size to 4 and 6 improves the obtained results with -6.6% and -3.5% deviation for the ASM read trigger implementation and a memory size greater $512\,\mathrm{MiB}$. It can be seen that the SIMD implementations trigger caches but deviate further from the target event count than the C and ASM implementations. Adding a random factor does not have an effect on the accuracy of the write functions.

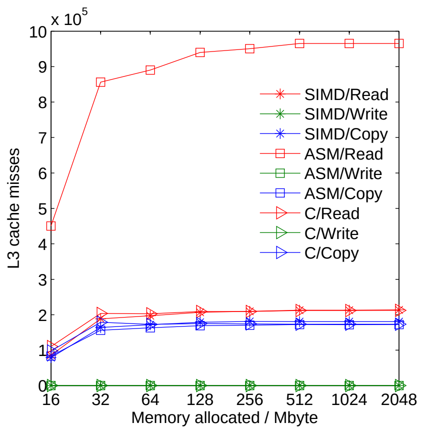


Figure 12.2: L3 cache misses for one process, step size 6 and target value 1×10^6.

Using shared memory instead of virtual process memory yields similar behavior for non-parallelized triggers. Setting parts of memory as uncachable results in a neglectable amount of L3 cache misses. We assume that accessing uncachable memory is not counted towards cache misses and therefore not further investigated for triggering cache misses or hits.

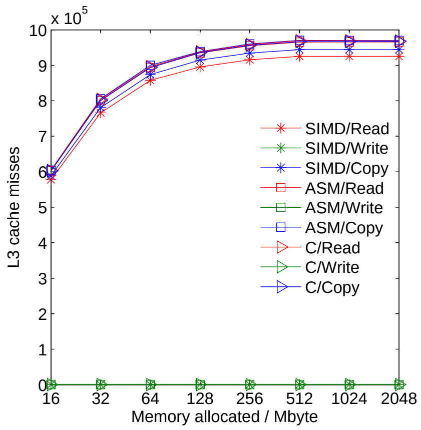


Figure 12.3: L3 cache misses for one process, step size 6, random step and target value 1×10^6.

The L3 cache miss trigger exhibits similar behavior when used in parallel with 4 and 8 processes. Using shared memory requires system calls but is otherwise comparable to using process owned memory and therefore not used

Table 12.1: L3 cache hits for process owned memory and target values 1×10^6, 4×10^6 and 8×10^6 bytes.

		1 Process		4 Processes		8 Processes	
		No rnd	Rnd	No rnd	Rnd	No rnd	Rnd
SIMD	Read	165 031	923 975	443 037	3 391 786	2 248 193	5 253 885
	Write	932	1420	6984	6088	22 623	31 034
	Copy	193 001	922 185	1 328 717	2 218 435	1 579 026	3 738 683
ASM	Read	204 282	966 181	1 225 584	3 408 932	1 747 484	4 047 238
	Write	914	1489	12 381	10 786	26 052	34 182
	Copy	222 718	979 248	1 459 860	3 155 795	2 478 220	4 343 272
C	Read	213 862	971 336	743 568	3 537 446	1 516 549	4 199 528
	Write	778	1421	8872	13 675	26 676	28 181
	Copy	259 416	968 232	1 358 422	2 701 179	1 556 921	4 320 763

in PET. We can conclude that a step size of 6 works best overall and that the C and ASM implementations exhibit a higher overall accuracy than the SIMD implementation, with ASM exhibiting the most promising characteristics. It is included in PET without randomness and using a step size of 6 together with a memory size of 512 MiB, as a compromise between memory footprint and accuracy. Larger step sizes were not evaluated as accurate results can be achieved with a step size of 6.

L3 Cache Hits and L2 Cache Misses Cache hits are measured similarly as L3 cache misses, except we only evaluate a step size of 6 due to its promising characteristics. The results are presented in Table 12.1. Not applying a random factor leads to fewer cache hits than targeted. Using shared memory results in similar behavior as observed with L3 cache misses. As with L3 cache misses, ASM and C implementations perform better than SIMD.

With a deviation of -11.5%, the C read implementation performs best when four processes are executed in parallel. The ASM and SIMD functions deviate further from the target value. Major discrepancies can be observed when using eight processes. Increasing the parallel process count from one to four processes scales reasonably well, yet increasing the process count above the number of physical cores does not. An exception to this observation is the SIMD read function. Its accuracy does not decrease as sharply with high parallelization. However, it does have a lower accuracy at lower process counts. The C read trigger shows the most deterministic behavior overall and scales

Table 12.2: Bytes read results with uncachable memory and target values 64×10^6, 256×10^6 and 512×10^6 bytes.

Processes	Implementation	Bytes read	Deviation
1	SIMD	64 002 560	0.004 %
	ASM	64 038 080	0.060 %
	C	64 025 024	0.039 %
4	SIMD	242.41×10^6	−5.31 %
	ASM	255.98×10^6	−0.01 %
	C	191.16×10^6	−25.33 %
8	SIMD	326.43×10^6	−36.24 %
	ASM	343.67×10^6	−32.88 %
	C	347.82×10^6	−32.07 %

with the process count up to the physical core count. We therefore integrate it into PET.

L2 Cache Hits The separate L2 cache hit trigger is not able of generating enough cache hits to reach the target value. Using more than one process does not result in an improvement. It is therefore not deemed sufficiently accurate to be included in PET.

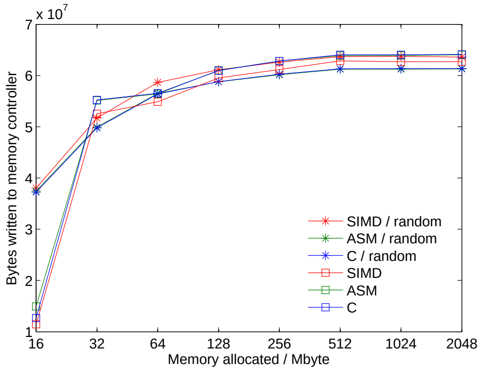


Figure 12.4: Bytes written results for one process with process owned memory and target value 64×10^6 bytes.

Bytes Read from Memory Controller As shown for L3 cache misses, a step size of 6 works best in triggering cache misses. Therefore, investigations for other step sizes are omitted. The results in Table 12.2 show a large overcount on

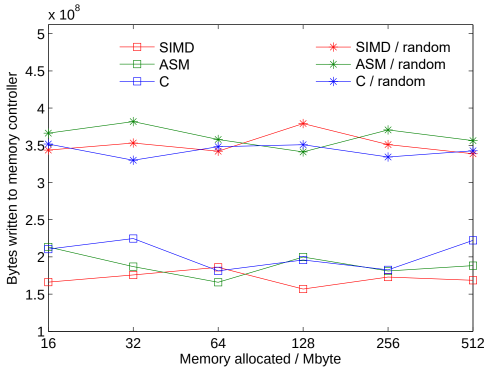


Figure 12.5: Bytes written results for eight processes with process owned memory and target value 512×10^6 bytes.

all implementations and memory sizes. When using uncachable memory, overcounting can be reduced to a minimum. Especially if parallelizing using 4 processes, the ASM read implementation comes close to the target value with only a small deviation. With 8 processes, deviations are larger with over -30% across all implementations. The ASM read function performs similarly as the C function and works well on lower process counts if combined with uncachable memory and therefore integrated into PET.

Bytes Written to Memory Controller   The results for a single process are presented in Figure 12.4 using process owned memory. The implementation works well for memory sizes of $512\,\mathrm{MiB}$ and larger. For parallelized triggers, as shown in Figure 12.5, the implementations are not as accurate. Functions including a random step size perform better. Using shared memory yields the same characteristics. The uncachable memory kernel module, as shown with the bytes read triggers, is expected to work well but instead less than half of the target bytes are written to memory. As the results show, the bytes written event trigger struggles at approximating the target value in multi-process environments.

Instructions Retired   Retiring instructions works with good accuracy and deviations below 1%, as Table 12.3 shows.

Context Switches   To test if our trigger implementations actually cause context switches in groups of two, we evaluate the implementation with different correction factors f by which the target event count is divided. The results in

Table 12.3: Retired instructions measurement results with target values of 1×10^{10}, 4×10^{10} and 8×10^{10}.

Processes	Result	Deviation
1	10.005×10^9	0.05 %
4	40.121×10^9	0.30 %
8	80.414×10^9	0.52 %

Table 12.4: Context switches measurement results and target values of 1×10^5, 4×10^5 and 8×10^5 switches.

	Factor f					
Proc.	2.00		1.25		1.00	
1	70 350	−29.7 %	100 588	0.6 %	120 641	20.6 %
4	271 683	−32.1 %	400 689	0.2 %	481 324	20.3 %
8	470 265	−41.2 %	757 056	−5.4 %	940 653	17.6 %

Table 12.4 show, using $f = 2.00$ causes a high deviation for one process and even higher for 4 and 8 processes. A deviation of around 20% can be measured if no correction is made. We also test if using a correction factor of $f = 1.25$ can correct for overcounting. This works well with one and four processes. With eight processes, a slightly higher deviation can be observed in comparison to fewer processes, but overall accuracy is still better. We therefore use $f = 1.25$ in PET.

Interrupts The results in Table 12.5 show that the interrupt implementation performs well with only minor deviations from the target value.

Table 12.5: Interrupt measurement results and target values of 1×10^5, 4×10^5 and 8×10^5 interrupts.

Processes	Result	Deviation
1	100 276.0	0.3 %
4	400 946.0	0.2 %
8	798 682.3	−0.2 %

Table 12.6: Side effects of performance event triggers per event generated for eight processes.

	Source event						
P. ctr.	L3MISS	L3HITS	READ	WRITE	INST	irq	ctxt
L3MISS		0.04	42.5×10^{-5}	8.1×10^{-4}	1.03×10^{-7}	0.3	10.4
L3HITS	0.02		36.1×10^{-4}	0.004	6.20×10^{-7}	4.7	73.7
READ	85.5	38.5		65.1	20.09×10^{-5}	265.4	7604.8
WRITE	21.2	10.4	13.6		55.71×10^{-5}	119.4	4732.3
INST	131.1	75.8	24.1	130.6		15 385.1	23 620.4
irq	0.000 03	0.0003	3.1×10^{-5}	1.9×10^{-5}	5.24×10^{-8}		0.001
ctxt	0.000 02	0.0007	8.5×10^{-4}	18.6×10^{-4}	11.63×10^{-8}	2.0	

12.1.2 Side Effects of Event Triggers

The results for performance counters not explicitly triggered are shown in Table 12.6. We present only the side effects for parallelized measurements with 8 processes, as the final evaluation of PET uses as many processes as there are logical cores on the SUT (as is default in our power methodology). All side effect measurements exceed background noise and are accounted for in PET, using one of the three composition methods described in Section 6.1.

Table 12.7: PI workload mean and maximum deviation and CV.

Measurement		Power Deviation Mean	Max.	CV
Full	Naive	2.95 %	14.96 %	6.94 %
	Accumulation	−1.93 %	−12.43 %	7.24 %
	Sim. An.	−42.05 %	−59.63 %	28.26 %
Pruned	Naive	−1.32 %	−4.39 %	1.79 %
	Accumulation	−0.62 %	−4.51 %	2.00 %
	Sim. An.	2.18 %	9.36 %	3.45 %

12.1.3 PET

PET is evaluated using four workloads with increasing complexity on the same set of selected performance counters. The first is the synthetic workload *PI*, designed to only stress the CPU by approximating π using the Gregory-Leibniz series $\pi \approx 4 \cdot \sum_{k=1}^{n} (-1)^{k+1} / 2k - 1$ with n randomly chosen between $[1000, 100000]$ for each transaction. As our second and third workloads in this evaulation we utilize *XMLValidate* and *SSJ*. The latter is also used in the SERT

Table 12.8: PI workload performance counter results.

Perf.		Background	Achieved / Target Value		
counter	Corr.	noise factor	Naive	Accu.	Sim.An.
L3MISS	0.91	0.031	454.520	404.483	518.529
L3HITS	0.91	0.003	621.546	532.234	673.416
READ	0.98	1.079	348.933	142.329	175.925
WRITE	0.99	2.244	250.555	166.988	159.703
INST	0.99	3.406	1.083	1.077	0.154
irq	0.22	2.319	0.685	0.596	6.302
ctxt	0.97	24.310	0.678	0.634	1.306

implementation of our measurement methodology from Chapter 4. Both XML-Validate and SSJ stress the CPU and memory. For each transaction, XMLValidate randomly moves comments inside an XML document which is then validated against an XML schema using the simple Application Programming Interface (API) for XML (SAX) and the document object model. SSJ resembles a real world application with six different transactions. It simulates an online transaction processing as a server side Java application, including placing new orders, payment, checking order status, processing orders for delivery, query stock level and generating customer reports. The last and most demanding workload is a DPI firewall that also uses additional I/O devices (network interface card (NIC)) driven by external load generators.

For each load level, a pre-measurement, measurement and post-measurement time of 30 s, 120 s and 10 s is used. For each workload, we evaluate PET's three composition mechanisms (see Section 6.1). We include the 95% confidence interval of each load level for all PET power measurement results, as well as the mean and maximum deviation and coefficient of variation (CV) over all load levels. We also analyze the ability to accurately trigger the target performance counters and evaluate the results for each of the three different composition mechanisms.

PI Workload When using a performance trigger composition that ignores side effects, power consumption behavior seems accurate on average. However, the maximum deviation is over 10% as shown in Table 12.7. If accumulating side effects, the CV is higher but it deviates less from the target power consumption on average and at its peak. Using simulated annealing to account for side effects does not perform as well and has a lower accuracy. When pruning performance triggers, all triggers except instructions retired and context switches can be removed (see Table 12.8).

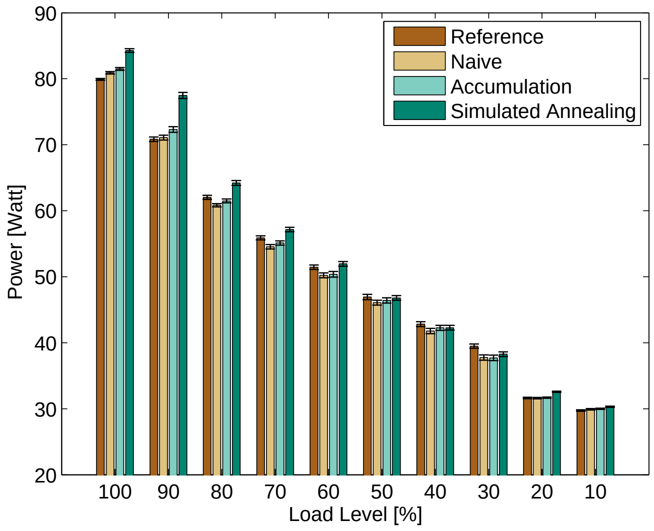


Figure 12.6: PI workload power consumption behavior with removed event triggers.

Pruning event triggers results in an overall improvement for all three composition methods, shown in Table 12.7 and Figure 12.6. The mean deviation from the target power consumption with the naive approach improves by a small margin while its CV and maximum deviation show a notable decrease. Accumulating side effects improves on mean and maximum deviation as well as CV. Simulated annealing also improves through pruning and in the same range as the naive and accumulation compositions. For PI, accumulating side effects for the pruned trigger composition is determined as the best solution, as

Table 12.9: XMLValidate mean and maximum deviation and CV.

		Power Deviation		
Measurement		Mean	Maximum	CV
Full	Naive	−10.48 %	−38.93 %	27.69 %
	Accumulation	−4.32 %	−11.29 %	5.04 %
	Sim. An.	−52.57 %	−67.78 %	26.81 %
Pruned	Naive	−23.31 %	−39.48 %	29.82 %
	Accumulation	−4.36 %	−11.57 %	5.40 %
	Sim. An.	−18.85 %	−27.02 %	8.61 %

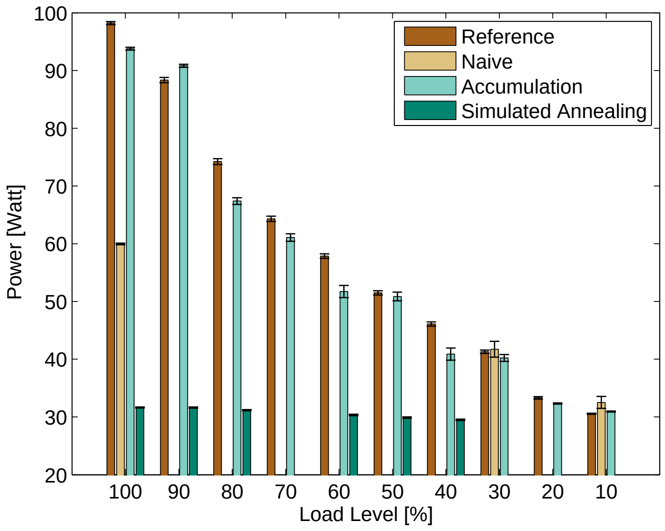


Figure 12.7: XMLValidate workload power consumption behavior.

its average deviation is closest to the target power consumption behavior with only marginally higher maximum deviation and CV. The results show that the local, CPU heavy, PI workload can be approximated with good accuracy and only minor deviations from the target power behavior with our approach.

XMLValidate Power measurements using PET compositions without event trigger pruning are shown in Figure 12.7 and the deviations are shown in Table 12.9. Not all load levels reach a steady state, but lower load levels with a steady state achieve the target power consumption with minor deviations. At

Table 12.10: XMLValidate performance counter results.

Perf.		Background	Achieved / Target Value		
counter	Corr.	noise factor	Naive	Accu.	Sim.An.
L3MISS	0.99	12.454	0.038	0.015	0.845
L3HITS	0.99	1.291	0.062	0.014	0.967
READ	0.98	463.950	1.880	0.049	0.320
WRITE	0.98	865.172	0.684	0.065	0.342
INST	0.98	5.868	0.069	0.666	0.084
irq	0.47	1.040	0.454	6.855	16.093
ctxt	0.96	5.928	0.053	114.533	5.680

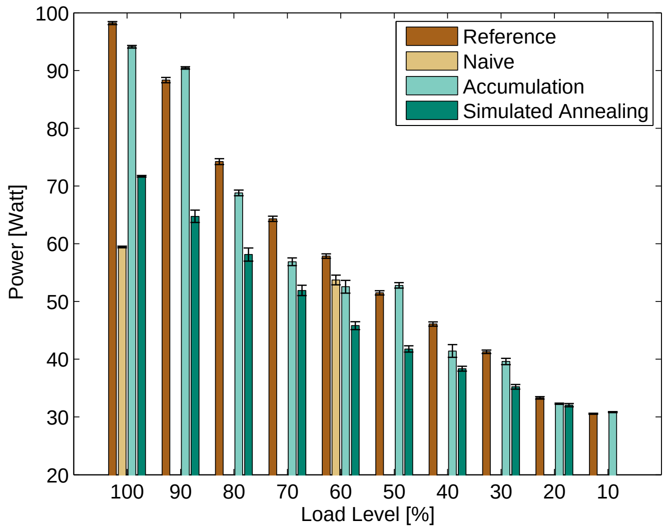


Figure 12.8: XMLValidate workload power consumption behavior with pruned performance event trigger.

the 100 % load level, the naive method features the largest deviation and it is therefore unlikely that it could reach other load levels with reasonable accuracy. Accumulation works well and reaches deviations and a CV comparable to the PI workload.

Based on Table 12.10, L3 cache misses and hits, bytes read and written, and the retired instruction count are not pruned for all measurements. The interrupts are pruned from all configurations due to low correlation. Context switches are removed from accumulation due to a significant overcount. However, some

Table 12.11: SSJ mean and maximum deviation and CV.

Measurement		Power Deviation Mean	Power Deviation Maximum	CV
Full	Naive	12.35 %	26.44 %	19.37 %
	Accumulation	13.28 %	27.61 %	7.03 %
	Sim. An.	−5.52 %	−9.35 %	3.66 %
Pruned	Naive	−3.35 %	20.91 %	12.99 %
	Accumulation	2.97 %	27.84 %	9.92 %
	Sim. An.	−9.63 %	−17.37 %	7.38 %

Table 12.12: SSJ performance counter results.

Perf. counter	Corr.	Background noise factor	Achieved / Target Value Naive	Accu.	Sim.An.
L3MISS	0.98	63.033	0.028	0.091	0.891
L3HITS	0.98	6.500	0.119	0.117	1.164
READ	1.00	706.852	1.071	0.164	0.927
WRITE	0.98	531.555	0.450	0.008	2.123
INST	0.98	1.652	0.186	1.563	0.799
irq	0.97	0.475	1.329	6.271	1.664
ctxt	0.99	0.597	0.291	65.128	54.963

load levels still do not achieve steady states, even with event trigger pruning. As with the PI workload, simulated annealing works better with fewer event triggers. Using a more complex workload does not necessarily result in a higher deviation. The accumulation measurement is in close proximity to the PI workload without pruning, with a mean deviation of below 5 %. Yet pruning performance triggers does not necessarily yield better results. This is mainly due to the workload stressing more hardware components, which in turn leads to fewer event triggers that can be removed.

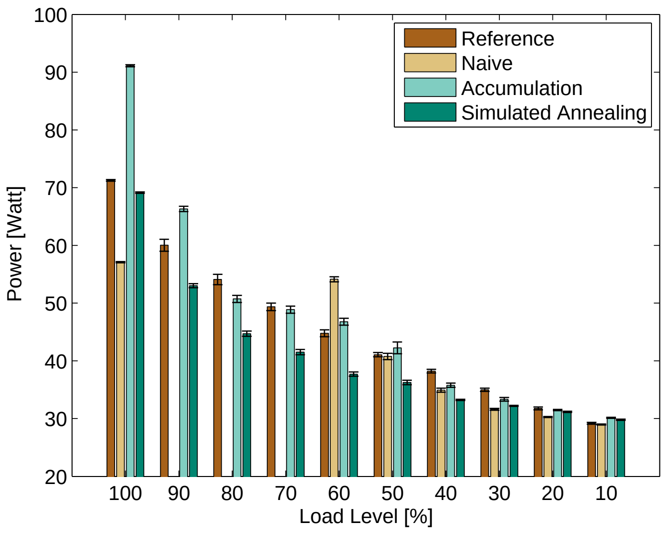


Figure 12.9: SSJ power consumption behavior with removed event triggers.

SSJ The results for SSJ are shown in Table 12.11. Ignoring side effects does not yield good results for the SSJ workload. Accumulation exceeds the reference power consumption whereas simulated annealing achieves a reasonable approximation that deviates $5\,\%$ on average with the maximum deviation staying below $10\,\%$. When trigger pruning is applied, only interrupts and context switches are a viable option to remove, as they are below the background noise as shown in Table 12.12.

Results for the pruned configuration are presented in Figure 12.9. It shows an improvement for the naive approach but above the $50\,\%$ load level, power behavior emulation accuracy decreases. Composition using accumulation improves power accuracy on average, but not the CV and maximum deviation. The measurement still seems affected by the same effects observed without pruning and exceeds the target power consumption at full load. Simulated annealing does not improve this effect either.

We show that even a complex workload can be emulated with a reasonable accuracy below $10\,\%$ on both mean and maximum deviations. Yet a mean deviation of less than $5\,\%$ is not achieved. We also conclude that reducing the amount of event triggers may not always improve the emulation for complex workloads.

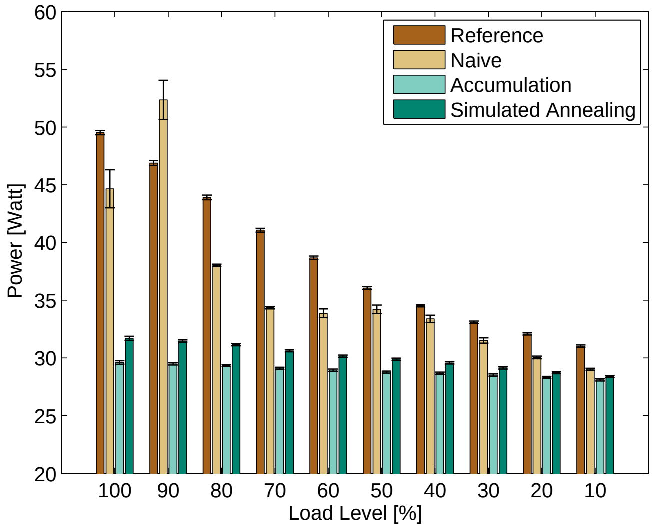


Figure 12.10: NFV workload power consumption behavior.

Table 12.13: NFV mean and maximum deviation and CV.

Measurement		Power Deviation Mean	Power Deviation Maximum	CV
Full	Naive	−6.63 %	−16.35 %	8.27 %
	Accumulation	−23.69 %	−40.19 %	14.33 %
	Sim. An.	−20.77 %	−35.97 %	12.33 %
Pruned	Naive	−4.52 %	27.18 %	13.42 %
	Accumulation	−23.38 %	−39.83 %	14.28 %
	Sim. An.	−20.83 %	36.20 %	12.28 %
Long	Naive	−8.84 %	−16.47 %	5.86 %
	Accumulation	−23.68 %	−40.23 %	14.33 %
	Sim. An.	−21.00 %	−36.19 %	12.32 %

NFV The DPI firewall stresses hardware parts otherwise not used by the previously employed CPU and memory bound workloads. Using the firewall, we evaluate the ability to emulate the power consumption-relevant behavior even though specific hardware components (NICs) remain unused. The results in Figure 12.10 and Table 12.13 show that this can be achieved. The naive composition that ignores side effects works best, deviating the least from the target in both mean and maximum power consumption. Yet it slightly underestimates power consumption consistently with one exception at the 90 % load level. All PET compositions favor underestimation. This behavior is expected considering that PET does not utilize NICs. The accumulation and simulated annealing compositions do not achieve the naive method's good accuracy with higher deviations and CVs. For pruned compositions only the L3 cache misses can be removed according to Table 12.14.

Pruning performance events to trigger does not result in an overall better

Table 12.14: NFV workload performance counter results.

Perf. counter	Correlation	Background noise factor	Achieved / Target Value Naive	Achieved / Target Value Accu.	Achieved / Target Value Sim.An.
L3MISS	0.97	2.776	1.380	0.019	1.572
L3HITS	0.97	0.260	1.748	0.021	1.729
READ	0.94	72.297	9.215	0.009	0.996
WRITE	0.97	91.529	5.092	0.010	1.395
INST	0.98	1.927	0.267	0.146	0.257
irq	1.00	44.395	0.070	0.506	0.311
ctxt	0.99	103.710	0.082	0.364	0.305

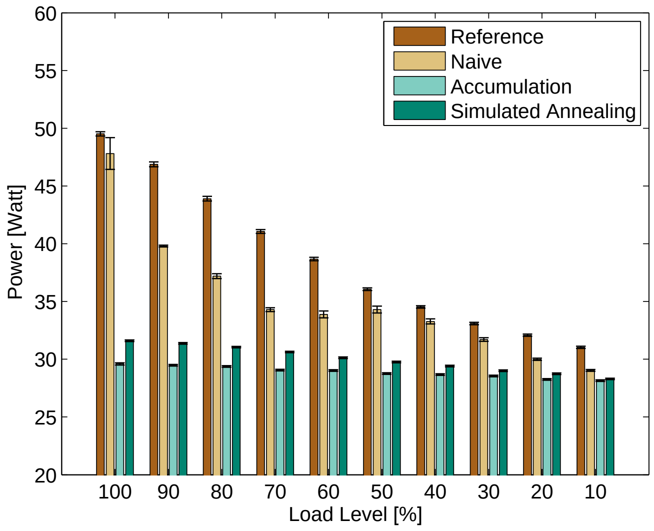


Figure 12.11: NFV power consumption behavior with 240 s measurement.

approximation. This coincides with the results for the XMLValidate and SSJ workloads. Accumulation and simulated annealing show only minor effects on the efficacy of pruning. Pruning improves naive compositions on average but at the cost of a higher CV and maximum deviation. The pruned naive composition still consumes too much power at the 90 % load level, which could be the result of the low transaction count in total.

To achieve more stable measurements for load levels where steady states are not reached, we repeat measurements with an elongated measurement phase of 240 s without pruning. We expect that a longer measurement phase might reduce the CV as well as increasing the total transaction count, as a longer measurement phase allows for a more stable measurement. Figure 12.11 and Table 12.13 show the results. The mean deviation for the accumulation composition increases but is still below 10 %. Its CV and maximum deviation improve significantly. Accumulation and simulated annealing compositions are not influenced by the longer measurement phase. We expect the naive composition results to converge towards the results of simulated annealing and accumulated compositions, indicating that NICs remain unused by PET. However, the naive composition approximates the I/O heavy load's power with a reasonable accuracy of less than 10 % on average. Simulated annealing and accumulation are not as accurate.

Summarizing, we evaluate our power profile emulation approach using performance event triggers, addressing **EQ A.6** ("Can we accurately emulate the power profile of more complex workloads using performance counters?"). We evaluate these performance triggers separately and determine the best performance trigger implementations in order to accurately trigger performance counter events. We also evaluate our overall PET framework, which emulates entire real-world workloads using the separate trigger implementations. We show that PET is capable of emulating the power consumption behavior of realistic workloads with a mean deviation down to 0.19 W (1%) and even show that it is able to emulate the power profile of I/O intensive applications, such as virtual network functions.

12.2 Use-Cases for the TeaStore Reference and Test Application

We demonstrate the use of TeaStore, which is introduced in Chapter 6.2. Specifically, we show its application in three use-cases. All use-cases are designed to show that TeaStore can be used as a software that exhibits the characteristics that are addressed or needed for the respective field of research. Specifically, we show TeaStore's use in three areas of research. We show that it can be used as a distributed application for evaluating and extracting software performance models, for testing single and multi-tier auto-scalers, and for software energy efficiency analysis and management, which addresses **EQ A.7** ("Can our distributed reference application be used to evaluate the quality of micro-service placements regarding energy efficiency and power?"). The use-cases are constructed as examples of the state-of-the art that current research has to compare itself against. For example, the auto-scaling example uses the standard state-of-the-art reactive auto-scaler, against which more advanced research scalers have to be compared.

For all of the experiments, we utilize much of the standard testing tools and profiles found in TeaStore's documentation. We generate load using an HTTP load generator, which sends requests based on an open workload model. Requests are defined using a load profile, in which the request rate may vary over time. For our measurements, the load profile may be constant, linearly increasing (stress test profile), or based on a real-world load trace. We extract the real world trace using the LIMBO load intensity model extraction mechanism, introduced in Chapter 5.1, and modify it to describe the load intensity in a range that can be handled by our SUT, varying between almost no load and the maximum throughput capacity. On the time scale, the real-world traces are modified to execute the load variations of the original multiple-day-long trace within one hour.

The content of the requests (meaning, the user actions) are defined using a stateful user profile. Each time a request is sent, an idle user from the pool of users is selected to execute a single action on the store. The user then performs that action and returns to the pool. This means that the user state and actions are chosen as they would be in a closed workload model (Schroeder et al., 2006), whereas the arrival times of the single requests are chosen according to an open workload model. We use a cyclical user profile, in which users browse the store. Figure 12.12 shows this profile. Users log in, browse the store for products, add these products to the shopping cart and then log out. The number of users is chosen depending on the maximum load.

We place TeaStore's primary services on several HPE ProLiant DL160 servers, each equipped with a Xeon E5-2640 v3 processor with 16 logical cores at 2.6

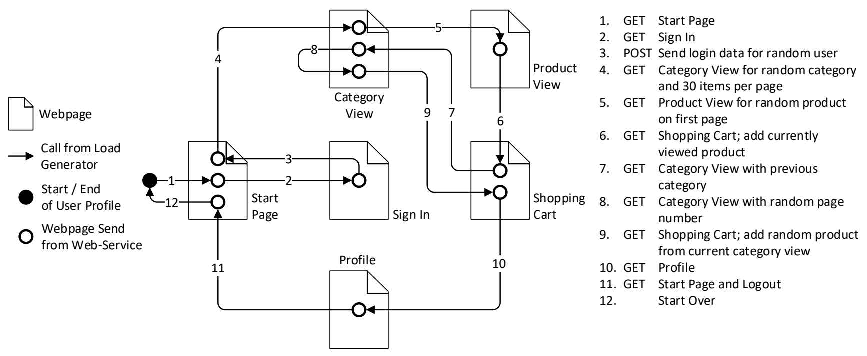

Webpage
Call from Load Generator
Start / End of User Profile
Webpage Send from Web-Service
Category View
Product View
Start Page
Sign In
Shopping Cart
Profile
1. GET Start Page
2. GET Sign In
3. POST Send login data for random user
4. GET Category View for random category and 30 items per page
5. GET Product View for random product on first page
6. GET Shopping Cart; add currently viewed product
7. GET Category View with previous category
8. GET Category View with random page number
9. GET Shopping Cart; add random product from current category view
10. GET Profile
11. GET Start Page and Logout
12. Start Over

Figure 12.12: "Browse" user profile configured in HTTP load generator for our use-cases, including web pages delivered and HTTP request type. Unused web pages are omitted for clarity.

GHz and 32 GB RAM. The servers run Debian 9 and Docker 17.12.1-ce. The service registry is executed on an additional physical host. We run the front-end load balancer and load generator on separate machines with network links to each of the service hosts.

12.2.1 Performance Modeling

Performance models provide a powerful tool for prediction of performance metrics, such as utilization or response time. These predictions enable smart capacity planning, especially in a micro-service environment where containers can be added or removed within seconds. Examples for such performance models are RESOLVE (Sitaraman et al., 2001), ROBOCOP (Bondarev et al., 2005), PCM (Becker et al., 2009), SAMM (Becker et al., 2017b), CACTOS (Groenda et al., 2017) and UML MARTE (Gérard and Selic, 2008). These models have a limited number of real world case studies, making it difficult to evaluate their applicability for real world scenarios. Additionally, quantitative comparison of performance models is challenging, as common case studies do not exist.

We showcase that TeaStore is well suited as a case study for advanced performance modeling concepts. We use a novel modeling mechanism, modeling TeaStore. Using TeaStore, we show that this novel mechanism can improve modeling accuracy when compared to standard state-of-the-art approaches. This, in reverse, highlights TeaStore's ability to evaluate the applicability and modeling accuracy of novel formalisms. Specifically, we evaluate the modeling concepts for parametric dependencies, see (Eismann et al., 2018), using TeaStore. This modeling formalism has the capability of modeling and predicting the impact of the changes within a workload profile, which can be mapped to a parameter in the underlying model. We compare this novel formalism to a standard state-of-the-art modeling method, by also constructing a model of TeaStore using the Descartes Modeling Language (DML) (Huber et al., 2017). Both models are used to predict the utilization of previously unseen deployments under different load levels. Using TeaStore, we aim to show that the ability of the novel modeling formalism to capture parametric changes in the workload profile increases prediction accuracy in comparison to the state-of-the-art method, which does not model this aspect. This comparison shows how TeaStore can be used as a case study to highlight the benefits of advanced performance modeling concepts.

The category page can display different amounts of products per page, based on user preference. We investigate the question of how changing the default number from five products per page to ten products per page impacts the performance. We assume that for a default of five products per page, some users manually switch to ten or twenty products per page, leading to the distribution shown in Equation 12.1, where $P_5(x)$ denotes the probability of any given user displaying the amount of x products per page, when a default value of 5 is pre-configured.

$$P_5(x) = \begin{cases} 0.9 & \text{if } x = 5 \\ 0.09 & \text{if } x = 10 \\ 0.01 & \text{if } x = 20 \\ 0 & \text{else.} \end{cases} \tag{12.1}$$

Equation 12.2 shows the assumed distribution for a default of ten products per page, where $P_{10}(x)$ denotes the probability of any given user displaying the amount of x products per page, when a default value of 10 is pre-configured. We aim to predict the impact of this change to the products per page on the performance of TeaStore.

$$P_{10}(x) = \begin{cases} 0 & \text{if } x = 5 \\ 0.99 & \text{if } x = 10 \\ 0.01 & \text{if } x = 20 \\ 0 & \text{else.} \end{cases} \tag{12.2}$$

We model the software architecture based on TeaStore architecture shown in Fig. 6.1. To parameterize the service demands in the static model, we deploy each service on a bare-metal server as shown in Fig. 12.13a, measure the utilization for a load of 1000 requests per second, using the usage distribution of Equation 12.1, and calculate the service demands according to Equation 12.3. The service demand defines the average time the CPU spends serving one request at the respective service (Spinner et al., 2015).

$$SerD = U/\lambda, \tag{12.3}$$

where U is the utilization and λ is the arrival rate. For the parametric model, we use linear regression to derive the Equations 12.4-12.6 for the service demands of the WebUI, ImageProvider and Persistence services, respectively.

$$SerD_{Web} = 0.0034 + 0.00016 * ProductsPerPage. \tag{12.4}$$

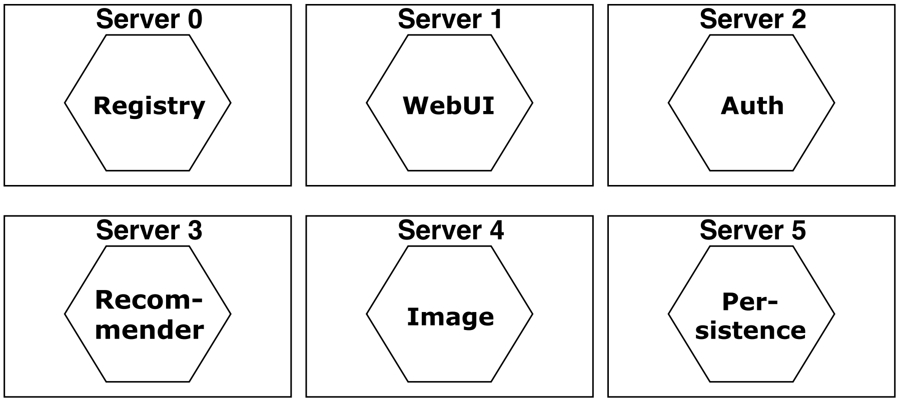
$$SerD_{Img} = 0.0012 + 0.00011 * ProductsPerPage. \tag{12.5}$$

$$SerD_{Per} = 0.0022 + 0.00005 * ProductsPerPage. \tag{12.6}$$

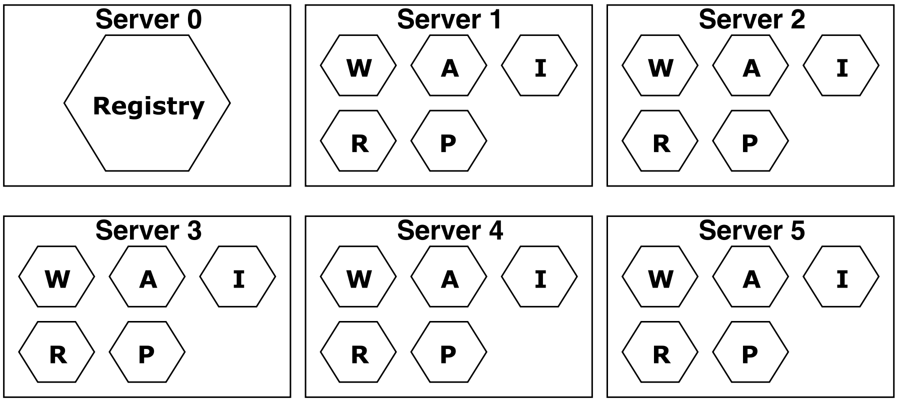
The service demand for the Authentication service remains static, as it is not influenced by the number of products per page. In both models, the recommender service was not included, since no recommendations are displayed on the categories page.

To evaluate the resulting models, we deploy an instance of every service on five servers to represent a production system, as shown in Figure 12.13b. For each server, we measure the CPU utilization under a load of 1000, 2000, 3000, 4000 and 5000 requests per second for the usage distributions from Equations 12.1 and 12.2. We compare the measured utilizations with the predicted utilizations using the static model and the parametric model for both distributions. Fig. 12.14 shows the absolute prediction errors for both models. For a default of five products per page, both the static and the parametric model have an absolute utilization prediction error of $< 5\%$. This is expected, as this is the scenario the static model was built for and calibrated with. However, for a default of ten products per page, the parametric model significantly outperforms the static model. Using TeaStore as a case study, we are able to highlight the benefit of modeling parametric dependencies. This illustrates TeaStore's applicability for model evaluation in one specific context.

Aside from parametric dependencies, TeaStore provides many opportunities to evaluate performance modeling concepts due to its performance properties. In the following, we list a number of challenges, which could be evaluated using TeaStore. We also list some example approaches that motivate or tackled these issues in the past:



(a) Calibration deployment.



(b) Evaluation deployment.

Figure 12.13: Deployments for model prediction. Services are abbreviated to their first letter.

Deployment options TeaStore offers a large variety of deployment options, as each service can be individually deployed and scaled. The prediction of resource utilizations and response times for previously unseen deployments and configurations is the most common use case for performance engineering. This task overlaps with the remaining challenges and therefore offers itself as a scenario for benchmarking of performance modeling approaches.

Internal state For some services, the performance does not only depend on the request, but also the internal state of the service. In TeaStore, the database size influences the service demand of the persistence provider service.

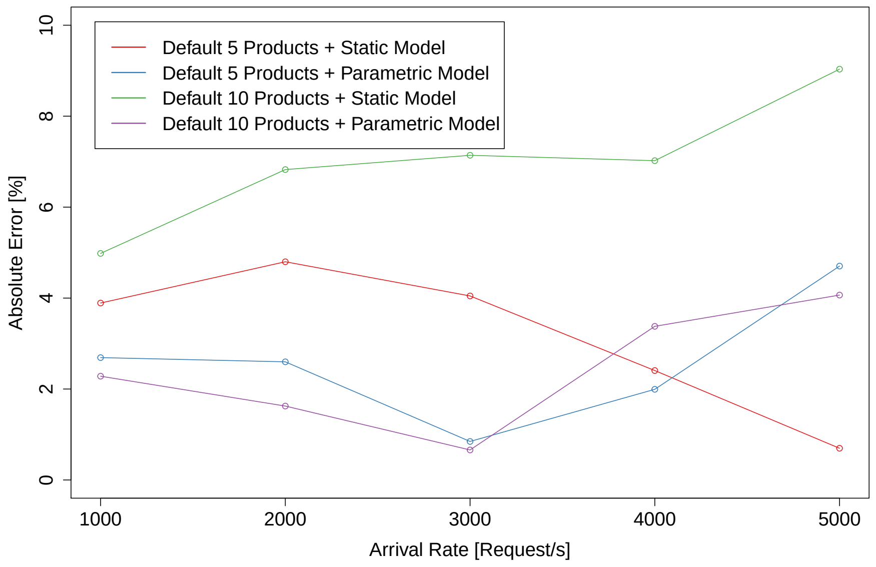


Figure 12.14: Absolute prediction error of the static and parametric model for two workload profiles and five workload intensities.

The number of entries in the database can dynamically change during operation, leading to changing persistence provider service demands. See e.g., (Happe et al., 2014).

Caching Caching mechanisms are challenging to model, as the behavior of a service with caching depends on its workload profile. The more frequent a small subset of items is requested, the more effective caching becomes. The Persistence and Image Provider of TeaStore implement caching. The implementation of the image provider cache is known and can be modeled as a white-box, whereas the persistence provider cache is part of the JPA implementation and has to be modeled as a black-box. See e.g., Bianchi et al., 2013 and Garetto et al., 2016

Network For distributed applications, network delays can influence their response time and can be a bottleneck which limits the maximum throughput. This means, the network topology and limitations need to be taken into account in a performance model to prevent significant prediction errors. As instances of each TeaStore service can be deployed on different machines, potentially even in different data centers, network delays can significantly influence the system response times. The Image Provider transmits potentially large images over the network, which can lead to a network bottleneck. See e.g., Rygielski et al., 2016.

Load types The CPU of a server is the most common bottleneck for applications, however the I/O capacity or the available memory can also be a bottleneck. Additionally, the load profile of two services deployed on the same machine influences how well they run in parallel. The authentication service causes CPU load, the Image provider I/O load and both the Image and Persistence provider caches cause memory load. Therefore, modeling of different load types is necessary to accurately predict the performance of TeaStore. See e.g., Huber et al., 2017.

Startup behavior Some systems dynamically add or remove service instances in order to adapt to changing workload. Whether a service starts instantly or takes some time until it is available, is an important factor when predicting the response times of such systems. The TeaStore services cover both cases. New Recommender instances need to be trained before they can process requests and Image provider instances need to generate the image files upon startup. For the remaining services new instances can be added and removed within seconds. See e.g., Papadopoulos et al., 2016.

Alternate Implementations Deciding between multiple implementations of the same component or service is a classic performance modeling challenge. TeaStore provides multiple recommender algorithms and image provider caches, which provides the opportunity for configuration optimization case studies. See e.g., Koziolek et al., 2013.

Timed tasks In most cases, the load of a system depends on the number of requests it receives. Sometimes, a system also regularly performs some tasks without user input, usually some kind of maintenance tasks. Modeling this behavior explicitly is important, as this load occurs independently of the user behavior. The Recommender service can be configured to be retrained at a regular interval, which is a realistic example of such a maintenance task.

We show that TeaStore can be used as a case-study for performance modeling approaches. It provides sufficient complexity to challenge existing approaches. The open source nature of TeaStore enables white-box modeling and any measurements using TeaStore can be easily replicated, since we provide docker containers with integrated monitoring. Therefore, TeaStore is a suitable case-study for a quantitative comparison between performance modeling approaches.

These comparisons can be done using manually created models or automatically extracted models using approaches such as the approaches of Walter et al., 2017a; Willnecker et al., 2015a; Krogmann et al., 2010 and Walter et al., 2017b. Using automatically extracted models would allow comparisons of full tool-chains, provided by the respective modeling formalisms, in a realistic usage scenario.

12.2.2 Auto-Scaling

In order to show that TeaStore works in an elastic manner and can thus be used for auto-scaling experiments, we stress an early development version of TeaStore using workloads derived from two different real-world traces (FIFA World Cup 1998, see Arlitt and Jin, 2000, and BibSonomy, see Benz et al., 2010). We employ a common, generic auto-scaler (Chieu et al., 2009) to automatically scale the store at run-time as the load intensity varies. We evaluate the quality of the scaler's decisions using a set of standard auto-scaler evaluation metrics.

12.2.2.1 Workload

We stress TeaStore using load intensity profiles based on different real-world workloads: (i) BibSonomy and (ii) FIFA World Cup 1998. We select a sub-set lasting four days from each of those traces. The BibSonomy trace represents HTTP requests to the social bookmarking system BibSonomy (see Benz et al., 2010) during April 2017. The FIFA[1] trace is a popular trace that was analyzed by Arlitt and Jin, 2000. The FIFA trace represents HTTP requests to the FIFA servers during the world championship between April and June 1998. We modify the traces to cover the load intensity range between a low load that can be covered using a minimal TeaStore deployment and a high load level that reaches the maximum capacity of the potential deployments for this scenario. On the time scale, the experiment is modified so that the load intensity variations of the original four days are executed within one hour.

12.2.2.2 Auto-Scaler

In 2009, Chieu et al., 2009 present a reactive scaling algorithm for horizontal scaling. This mechanism provisions VMs based on an application's scaling indicators. Among other things, the indicators consist of the number of active connections or the number of requests per second. The auto-scaler monitors these indicators for each VM and calculates the moving average. Next, the

[1]FIFA Source: `http://ita.ee.lbl.gov/html/contrib/WorldCup.html`

Table 12.15: Mapping of config. number and used containers.

	Configuration								
Service	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**
WebUI	1	1	2	3	3	4	5	5	6
Image Provider	1	1	2	3	3	4	5	5	6
Authentication	1	2	3	4	5	6	7	8	9
Recommender	1	1	1	2	2	2	3	3	3
Persistence	1	2	2	3	4	4	5	6	6

virtual machines with active sessions above or below given thresholds are determined. Finally, if all virtual machines have active sessions above a given threshold, new instances are provisioned. Upon detecting VMs with active sessions below the threshold and with at least one virtual machine that has no active session, idle instances are removed.

In these experiments we aim to use a representative auto-scaler. Consequently, we choose this reactive technique, as the underlying mechanism is simple and straight-forward. Based on this simplicity, this reactive generic auto-scaling-mechanism can and does serve as baseline approach when comparing state-of-the-art research auto-scalers. For our experiments, we use a re-implementation of this reactive algorithm that scales docker containers and name it *React*. In the evaluation framework of Ilyushkin et al., 2018, React shows a generic and stable scaling performance.

Scaling distributed applications is still an open research challenge and currently not supported by many state-of-the-art auto-scalers and not by our baseline auto-scaler. To account for this, we provide a best-effort list of which service to scale at which stage. This list acts as a mapping between the distributed services of TeaStore and the expected single service instances (scaling units) typically addressed by auto-scalers. Specifically, in our case, the auto-scaler under test always deploys a full-stack as the first scaling unit, then one Authentication and one Persistence instance as the second unit and, finally, one WebUI, Authentication, and Image instance each as the third scaling unit. Each service instance is limited to one virtual CPU core on the host machine. For additional scaling units, these three steps are repeated on the next available physical host (see Table 12.15). We use up to three physical server, resulting in a maximum configuration with a total of 30 service containers.

12.2.2.3 Quantifying Scaling Behaviour

In order to evaluate the scaling decisions made by React, we consider the proportion of failed transactions, the average response time and a set of system-oriented elasticity metrics endorsed by the SPEC RG (see Herbst and more, 2016). In particular, we focus on the *wrong provisioning time share*.

The wrong provisioning time share captures the time in which the system is in an under-provisioned (or over-provisioned) state during the experiment interval. That is, the *under-provisioning time share* τ_U describes the time relative to the measurement duration, in which the system is under-provisioned. Similarly, the *over-provisioning time share* τ_O describes the time relative to the measurement duration in which the system is in an over-provisioned state. The range of this metric is the interval $[0, 100]$. The best value of 0 is achieved, when the system features no over- or under-provisioning during the measurement. We define both metrics τ_U and τ_O as follows:

$$\tau_U[\%] := \frac{100}{T} \cdot \sum_{t=1}^{T} max(sgn(d_t - s_t), 0)\Delta t$$

$$\tau_O[\%] := \frac{100}{T} \cdot \sum_{t=1}^{T} max(sgn(s_t - d_t), 0)\Delta t \text{ , where}$$

d_t is the minimal amount of scaling units required to meet the Service Level Objectives (SLOs) under the load intensity at time t, s_t is the resource supply at time t, and T is the experiment duration. Δt denotes the time interval between the last and the current change either in demand d or supply s.

To know whether or not the system is in an under-provisioned or over-provisioned state, we execute separate off-line calibration measurements, which measure the average throughput of each target configuration in an over-loaded state. Based on this configuration, capacity information, and the load intensity of our traces, we are able to derive the required resource configuration for each point in time, which we can then use for calculating the time share metrics.

12.2.2.4 Evaluation

The scaling behavior of React on both the FIFA and BibSonomy traces is shown in Figure 12.15 and in Figure 12.16. Both figures are structured as follows: The horizontal axis shows the experiment time in minutes; the vertical axis represents the current number of scaling units (the configuration). Table 12.15 shows the mapping between the number and used containers. The red, dashed curve represents the required configuration, which has been derived using the

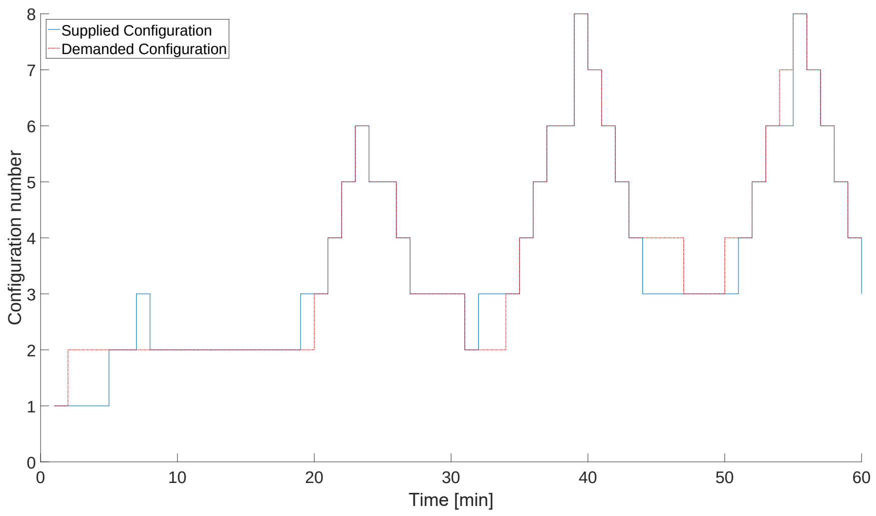


Figure 12.15: Scaling behavior for the FIFA trace.

separate off-line calibration tests, and the blue curve shows the supplied configuration, as measured during the experiment. For both traces, React scales the system in a similar fashion. That is, in both traces, the supplied configuration matches the required configuration for long periods. However, some deviations between supplied and required configurations occur. In Figure 12.15, for example, the system is in an under-provisioned state in the entire interval between minute 2 and 5. Overall, the under-provisioning and over-provisioning time-shares are equal or below 15% in both traces (see Table 12.16), indicating good scaling behavior. In addition to the low time-share metrics, the assertion of good scaling behavior is backed up by the observation that the proportion of failed transactions is below 2% and the average response time is lower than 0.19 s for both traces.

The results of our auto-scaler tests indicate that TeaStore can be used to compare the elasticity and performance of state-of-the art auto-scalers. Specifically, they show that TeaStore can even be used for comparing commonly used and state-of-the art autoscalers, even though these scalers are usually limited to single tier scaling. In addition, the results show that TeaStore exhibits robust behavior during run-time scaling, as the proportion of failed transactions is below 2%. This characteristic is achieved through the micro-service architecture and the client-side load balancers. The results also indicate that TeaStore is sufficiently scalable to allow for experiments of this kind.

Many open challenges remain, despite the good performance of the *React*

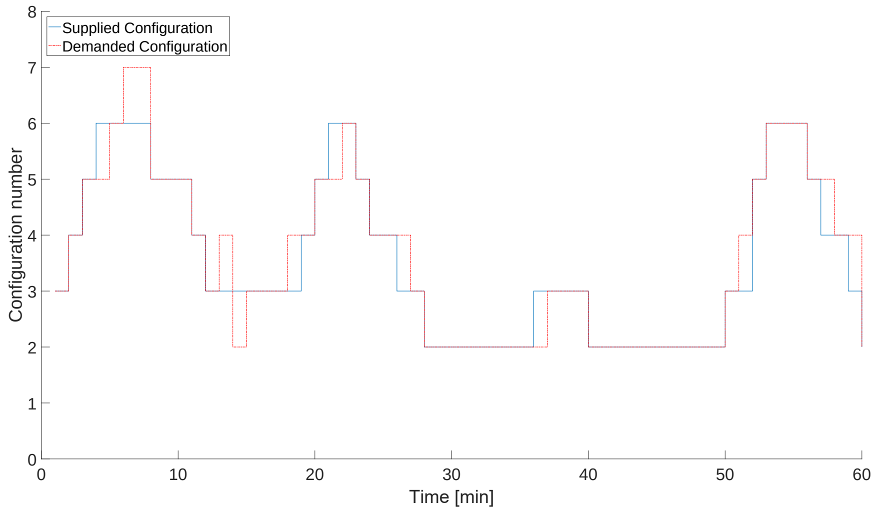


Figure 12.16: Scaling behavior for the BibSonomy trace.

Table 12.16: Result metric overview for both traces.

Metric	Fifa	BibSonomy
τ_U under-provisioning time share	15%	13%
τ_O over-provisioning time share	7%	6%
Proportion of failed transactions	1%	1%
Average response time	0.18 s	0.15 s

auto-scaler. In these experiments, the service deployment order is fixed and the auto-scalers do not have to decide which service to place on which machine, but rather when to add or remove the next configuration from the pre-defined list of configurations. Distributed application deployment decisions of this kind remain an open challenge. Furthermore, load profiles may be more complex, as the used profile does not change the database and does not change in the user actions performed over time. Real-world auto-scalers face the challenge of evolving user-behavior and changes in request service demands over time.

12.2.3 Energy-Efficiency Analysis

Energy efficiency and power prediction methods, such as the ones of Beloglazov et al., 2012 and Basmadjian et al., 2011 are often employed to solve a placement problem for services in distributed systems. The underlying challenge is that

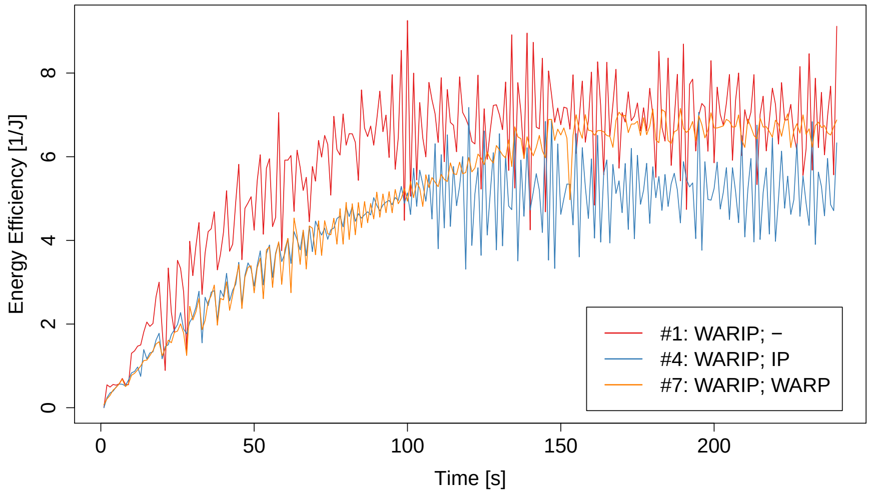


Figure 12.17: Energy efficiency for linearly increasing load. Services are abbreviated to their first letter.

different distributions of application services across physical hosts may not only result in different performance behavior but also in differences in overall power consumption. This section demonstrates this effect using TeaStore, addressing **EQ A.7** ("Can our distributed reference application be used to evaluate the quality of micro-service placements regarding energy efficiency and power?"). We show that different distributions of TeaStore's services can result in different performance and in different power consumption both on homogeneous and heterogeneous systems.

For these experiments, we use an increasing load intensity profile. The load profile starts at 8 requests per second and increases to 2000 requests per second over the time of four minutes. Request content is, again, specified using the user browse profile for the 128 users accessing the store. Depending on the current SUT configuration, some of the four minutes are spent in an under-provisioned state, in which the load intensity exceeds the capacity of the SUT. We measure the power consumption of the physical servers and throughput of TeaStore during the entire run. However, we only take those measurements into account, which are measured during the time in which load arrives at the system. Each measurement is taken on a per-second basis and thus tightly coupled to the current load intensity.

We calculate the following metrics based on the throughput and power consumption:

Table 12.17: Energy efficiency on homogeneous servers. Services are abbreviated to their first letter.

#	Serv.1	Serv.2	Capacity	Max Pwr	Eff.
1	WARIP	-	779.7 ±[29.7]	**114.4 W**	**5.3**
2	WI	ARP	1177.5 ±[31.5]	193.6 W	4.2
3	WAI	RP	883.4 ±[39.4]	175.8 W	3.8
4	WARIP	IP	863.0 ±[40.5]	173.5 W	3.9
5	WARIP	AIP	1228.7 ±[18.9]	208.4 W	4.2
6	WAIP	WARP	1231.8 ±[18.7]	203.7 W	4.3
7	WARIP	WAIP	1404.1 ±[14.5]	217.9 W	4.3
8	WARIP	WARIP	**1413.2** ±[14.7]	217.7 W	4.3

1. **Energy Efficiency**: We use the energy efficiency metric of our methodology in Chapter 4:

$$\mathit{Efficiency}[J^{-1}] = \frac{\mathit{Throughput}[s^{-1}]}{\mathit{Power}[W]}$$

   As energy efficiency is a ratio, we aggregate multiple energy efficiency scores using the geometric mean.

2. **Estimated Capacity**: We estimate the throughput capacity of each configuration by averaging the last 50 seconds of our load profile. Note that all configurations are operating at maximum load (capacity) at this time.

3. **Maximum Power Consumption**: The maximum power consumption measured (in watts). It indicates the power load that the configuration can put on the SUT.

12.2.3.1 Energy Efficiency on Homogeneous Systems

We run TeaStore with un-restrained docker containers on up to two of our testing servers. Table 12.17 shows the estimated capacity, maximum power, and geomean energy efficiency for different TeaStore deployments on those servers (Service names in the table are abbreviated to their first letter). The table confirms our previous assertion that the store performs differently depending on the service distribution. Capacity (maximum throughput) varies significantly for the different deployments, with some two-server deployments barely exceeding the capacity of the single server deployment and others almost doubling it.

The single server deployment (deployment #1) features the lowest performance, but also the lowest power consumption. As a result, it has the highest energy efficiency among all tested configurations. This is mostly due to our increasing stress test profile. At low load, as the load increases, the single system is still capable of handling all requests, while also consuming less power. At high load, it operates at capacity, but still consumes less power than the two-server setups. Figure 12.17 visualizes the energy efficiency over time for the single-server and two selected two-server deployments. The figure shows that an efficient two-server deployment can reach a similar energy efficiency as the single-server deployment at maximum load. However, some low performance deployments are incapable of reaching this efficiency and are overall less efficient due to the power overhead of the second server.

Among the two-server deployments, maximum power consumption is usually greater for those deployments with greater capacity, but some notable differences exist, which indicate room for power and efficiency optimization, even on a homogeneous system. Two notable examples emerge: Comparing deployment #2 with deployment #3, shows that deployment #2, which deploys the WebUI and Image services on one and the Auth, Recommender, and Persistence services on the other server has both a better performance, as well as smaller power footprint than deployment #3, which deploys the WebUI, Auth, and Image services on one server and the Image and Persistence services on another server. Consequently, deployment #2 features a better energy efficiency. In this example, one deployment is obviously better than another and TeaStore could be used to evaluate if a prediction or management mechanism actually selects the better option. However, in some cases power does not scale the same as performance. This case is hinted at when comparing deployment #5 and #6. Both deliver equal performance, but deployment #6 consumes slightly less power and is therefore a bit more efficient.

12.2.3.2 Energy Efficiency on Heterogeneous Systems

For our measurements on heterogeneous systems we replace the second server with an HP ProLiant DL20 system, which features an Intel Xeon E3-1230 v5 processor with 4 cores at 3.5 GHz and 16 GB RAM. This second server does not offer as much performance and consumes less power compared to its 8 core counterpart. Naturally, when deploying on this heterogeneous system, the order of deployment matters, as servers differ in power and performance.

Table 12.18 shows the measurement results of the heterogeneous system. It shows the performance, power, and energy efficiency for selected deployments. It illustrates the effect that deployment order has on the heterogeneous

Table 12.18: Energy efficiency on heterogeneous servers. Services are abbreviated to their first letter.

#	8 core	4 core	Capacity	Max Pwr	Eff.
1	WARIP	-	779.7 ±[29.7]	114.4 W	5.3
2	AIP	WARIP	781.1 ±[11.1]	**163.1** W	3.9
3	WARIP	AIP	**1207.3** ±[23.4]	189.5 W	**4.6**
4	WAIP	WARIP	1011.9 ±[24.7]	179.6 W	4.4
5	WARIP	WAIP	1067.7 ±[26.7]	187.0 W	4.3
6	WARIP	WARIP	1003.9 ±[24.9]	179.7 W	4.1

system, especially regarding deployments #2 and #3, which are the same deployment, except that they deploy each respective stack on the different server. Deployment #2 deploys the full stack on the smaller server and replicates some components on the larger machine, wheres deployment #3 does the reverse. Although deployment #3 consumes more power than #2, it has a far better performance and greater overall efficiency. It should also be noted that deployments with fewer services on the smaller machine seem to be more efficient in the heterogeneous environment compared to the respective deployments in the homogeneous environment, which deploy the smaller stack on an equivalent machine. Deployment #5 corresponds to deployment #7 on the homogeneous system (see Table 12.17), which is the most efficient system in that context. However, on the heterogeneous system, it is trumped in performance and efficiency by deployment #3, which places fewer services on the smaller machine.

In addition, the heterogeneous system demonstrates an efficiency – performance trade-off when compared to the homogeneous system. The most efficient heterogeneous deployment has a slightly lower performance capacity than the best homogeneous one, yet consumes less power and has a better energy efficiency.

Overall, our experiments show that TeaStore exhibits different performance and power behavior depending on deployment, both on heterogeneous and homogeneous systems. Due to this, it can be used to evaluate the prediction accuracy of power prediction mechanisms. In addition, some of our configurations feature a performance – efficiency trade-off, which is highly relevant for power and performance management, showing that TeaStore can be used to evaluate such management approaches.

Summarizing, we demonstrate the use of TeaStore, introduced in Chapter 6.2 in three application scenarios: Performance model evaluation, auto-scaler evaluation, and evaluation of energy efficiency of service placements, addressing

EQ A.7 ("Can our distributed reference application be used to evaluate the quality of micro-service placements regarding energy efficiency and power?"). Researchers may decide to use TeaStore and the corresponding testing tools and profiles in any of these three primary application scenarios. TeaStore can also be used in additional settings to achieve evaluation results that demonstrate the applicability of their work, while also enhancing comparability of their results.

12.3 Summary

This chapter addresses **EQ A.6** ("Can we accurately emulate the power profile of more complex workloads using performance counters?") and **EQ A.7** ("Can our distributed reference application be used to evaluate the quality of micro-service placements regarding energy efficiency and power?") by evaluating our performance counter based workload emulation approach and demonstrating the use of TeaStore.

We evaluate our power profile emulation approach using performance event triggers, addressing **EQ A.6**. To this end, we evaluate these performance triggers separately and determine the best performance trigger implementations in order to accurately trigger performance counter events. We also evaluate our overall PET framework, which emulates entire real-world workloads using the separate trigger implementations. We show that PET is capable of emulating the power consumption behavior of realistic workloads with a mean deviation down to 0.2 W (1%) and even show that it is able to emulate the power profile of I/O intensive applications, such as virtual network functions.

We demonstrate the use of TeaStore, introduced in Chapter 6.2 in three application scenarios: Performance model evaluation, auto-scaler evaluation, and evaluation of energy efficiency of service placements, addressing **EQ A.7**. Researchers may decide to use TeaStore and the corresponding testing tools and profiles in any of these three primary application scenarios. TeaStore can also be used in additional settings to achieve evaluation results that demonstrate the applicability of their work, while also enhancing comparability of their results.

Chapter 13

Accuracy of Power Interpolation

We evaluate the accuracy of our interpolation methods from Chapter 7 based on measurements using the SERT implementation of our methodology of Chapter 4, addressing **EQ B.1** ("Can interpolation be used to accurately model power consumption per load level on servers?"). The goal of using the SERT is to utilize its many worklets and load levels for a thorough evaluation of our interpolation method on the function of server power per load.

We use all of SERT's worklets, except for the memory *Capacity* worklet, as it doesn't scale with load levels, and the *XMLvalidate* worklet, which didn't scale correctly for fine-granular target load levels. The used worklets are: six different CPU worklets (*Compress*, *CryptoAES*, *LU*, *SHA256*, *SOR*, and *SORT*), the two storage worklets (*Random* and *Sequential*, see Lange et al., 2012), the memory *Flood* worklet, described by Lange et al., 2013, and the hybrid *SSJ* worklet. We exercise each of the worklets at 100 different load levels, with a separate idle power measurement serving as a 101st measurement at the 0% load level. For each of these intervals, throughput (in s^{-1}) and power consumption (in W) are measured. Energy efficiency (in $\frac{s^{-1}}{W} = \frac{1}{J}$) can be derived from these measurements by dividing throughput by power consumption.

We select both evenly distributed as well as scattered subsets of measurements from the original 101 results. Evenly distributed subsets are selected using the data points from 5, 6, or 11 equi-distant load levels (i.e., the 0%, 25%, 25% ... load levels for five data points, the 0%, 20%, 40% ... load levels for six data points, etc.). As a result, our data sets have load level intervals of 25%, 20%, or 10% between the considered levels. The scattered subset is selected based on 8 scattered load levels. These scattered levels stay the same during the entire evaluation to ensure comparability of results. All models are evaluated based on their ability to accurately reconstruct the entire original measurement for the given workload using no additional data. We compute the relative absolute error ($|p_{model} - p_{preferece}|/p_{preference}$) for each data point in the reference measurement. The mean and median of this relative point-wise difference in

Table 13.1: Mean modeling errors of interpolation and reference methods for the power over load level function of the SSJ workload.

	Interval sizes			
Model	**10%**	**20%**	**25%**	**scattered**
Cubic Spline	0.172%	0.222%	**0.224%**	**0.19%**
Max. degree Polynomial	0.35%	0.249%	0.304%	0.215%
Linear	0.169%	0.297%	0.296%	0.175%
Nearest Neighbor	0.704%	1.419%	1.772%	1.117%
Piece-Wise Polynomial (degree 2)	**0.168%**	0.225%	0.255%	0.191%
Piece-Wise Polynomial (degree 3)	0.19%	0.243%	0.317%	0.26%
Piece-Wise Polynomial (degree 4)	0.204%	0.246%	-	0.218%
Shepard (weight 1)	1.325%	1.449%	1.639%	1.434%
Shepard (weight 2)	0.353%	0.651%	0.793%	0.58%
Split Polynomial (1 break)	0.35%	0.249%	0.304%	0.215%
Split Polynomial (2 breaks)	0.305%	0.246%	0.317%	0.197%
Split Polynomial (3 breaks)	0.294%	**0.217%**	0.241%	0.191%
Split Polynomial (4 breaks)	0.284%	0.233%	0.296%	0.26%
Linear Power Model	2.757%	2.757%	2.757%	2.757%
Exponentially Corrected Model	4.023%	3.96%	3.938%	4.042%
Linear Regression	0.344%	0.255%	0.317%	0.199%

predicted and measured data serve as error metrics for the overall model. Of course, a smaller relative error corresponds to a more accurate model.

13.1 Models for Comparison

We compare the accuracy of our interpolation functions with three common power modeling approaches:

- **Linear Power Model:**
  The linear power consumption model is a common model in literature. It is relatively simple, using only two data points, yet features a surprising accuracy and robustness (see Rivoire et al., 2008). It calculates power consumption at a target load level $u \in [0, 1]$ as seen in Eq. 13.1:

$$p(u) = p_{idle} + (p_{max} - p_{idle})u \tag{13.1}$$

- **Linear Power Model with Exponential Correction:**
  This power model, introduced by Fan et al., 2007, modifies the linear

Table 13.2: Median modeling errors of interpolation and reference methods for the power over load level function of the SSJ workload.

	Interval sizes			
Model	**10%**	**20%**	**25%**	**scattered**
One single max. degree Polynomial	0.18%	0.185%	**0.149%**	0.144%
Piece-Wise Polynomial (degree 2)	**0.09%**	**0.143%**	0.205%	0.154%
Split Polynomial (1 break)	0.18%	0.185%	**0.149%**	0.144%
Split Polynomial (2 breaks)	0.151%	0.182%	0.194%	**0.127%**
Split Polynomial (4 breaks)	0.171%	**0.143%**	0.279%	0.186%
Linear Regression	0.186%	0.185%	0.267%	0.128%

power model using an exponential correction factor r, accounting for the curvature in power per utilization functions (Eq. 13.2):

$$p(u) = p_{idle} + (p_{max} - p_{idle})(2u - u^r) \tag{13.2}$$

We determine r by reforming the problem into a linear one and then fitting r by minimizing the square errors using linear regression.

- **Polynomial Fitting using Regression:**
  We create polynomials of varying degrees to fit the measured results. These functions model either power or performance with the load levels serving as the input parameter. Regression is applied to fit the coefficients $a_0, a_1, ..., a_n$ of the polynomial interpolation function $p(u) = a_n x^n + ... a_1 x + a_0$. We use the Apache Commons Math OLS multiple linear regression implementation.

13.2 Comparison of Interpolation Methods

A comparison of the mean modeling accuracy of the interpolation and modeling methods for the power per load level function of the hybrid SSJ workload is shown in Table 13.1. As mentioned previously, the displayed modeling error metric is the mean of the relative absolute differences between each data point in the reference measurement and the corresponding model prediction. The table shows the modeling error for interpolation methods and the different power models. Interpolation methods with multiple configuration options (such as Shepard weights) are marked with their respective configuration. The optimal interpolation method changes depending on input dataset size. Table 13.1 shows the modeling errors for all of the evaluation datasets.

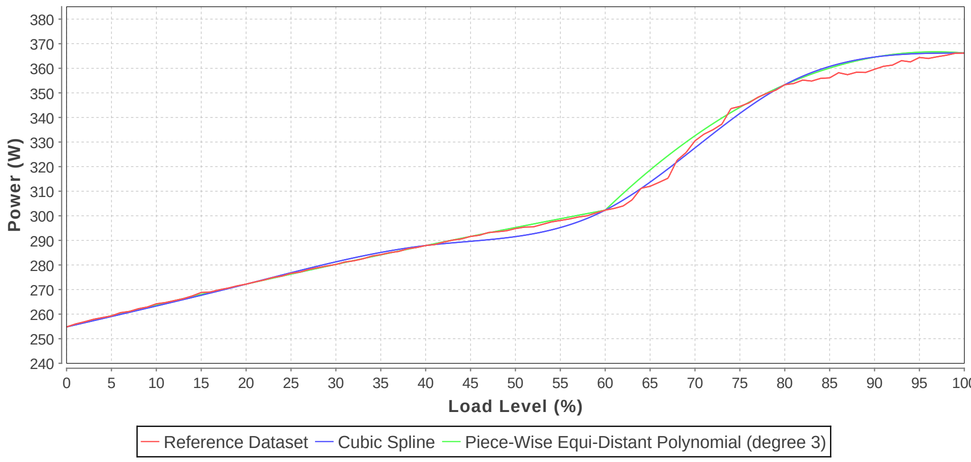


Figure 13.1: Accuracy of LU interpolation for interpolation set at 20% load level intervals.

Compared to the simple power models, all other interpolation functions are highly accurate, with modeling errors reliably less than 1% for all dataset sizes. The two exceptions are nearest neighbor and Shepard interpolation.

Polynomial interpolation provides the greatest accuracy, the optimal configuration (equi-distant splits, dynamic splits, or splines) depends on the size of the input dataset. Optimal configuration changes depending on whether accuracy is optimized towards mean or median accuracy. As Table 13.2 shows, the optimal interpolation strategy changes for all datasets except for the 10% interval dataset. However, median accuracy favors piece-wise polynomial interpolation as well.

The power per load level curve of select interpolation methods for the 20% interval subset and the reference dataset for SSJ is shown in Figure 13.2. This figure shows that the two most accurate interpolation methods for SSJ (cubic spline and equi-distant split polynomial with three splits) approximate the reference measurement almost perfectly with the exception of a few irregular spikes in the reference dataset's power consumption around the 25% and 90% load levels.

Regression is not as accurate as any of the piece-wise polynomial interpolation methods (equi-distant splits, dynamic splits, or splines) in cases of equi-distant data. For our scattered dataset, regression is only slightly less accurate than cubic spline interpolation.

It is also notable that the dynamically split polynomial interpolation and

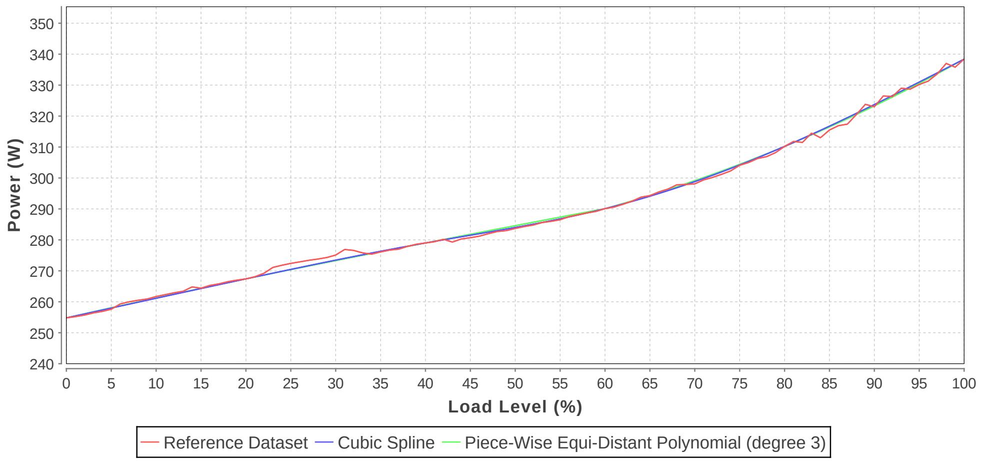


Figure 13.2: Accuracy of SSJ interpolation for interpolation set at 20% load level intervals.

interpolation using one polynomial of maximum degree feature identical accuracy for this workload. This effect is the result of the largest difference in power consumption for SSJ taking place right before full utilization. Consequently, the first breakpoint is set at full utilization, resulting in no change to polynomial interpolation without breakpoints.

This observation is specific to SSJ and does not repeat for other workloads. The CPU-bound LU workload's power scales differently with increased load, as can be seen in the power scaling behavior of the LU workload. Figure 13.1 shows that LU has a sharper increase in power consumption beginning at 60% load. This results in a visible impact on interpolation accuracy (see Table 13.3), as interpolation methods must correctly recognize this sudden increase in power draw. The accuracy gap between interpolation and regression is significantly higher than for SSJ, in addition cubic spline interpolation does not perform as well. However, piece-wise polynomials (equi-distant and dynamically split) are still the overall best choice in most cases and a good choice in all cases.

Table 13.4 shows the most accurate interpolation strategies for all worklet – subset combinations. Configuration parameters and model accuracy are displayed in parentheses behind each respective optimal strategy. Polynomial interpolation always outperforms regression, as some version of polynomial interpolation is always the most accurate of the evaluated interpolation methods (Keep in mind that linear interpolation is polynomial interpolation with piecewise polynomials of degree 1). The best interpolation method is our new

Table 13.3: Mean modeling errors of interpolation and reference methods for the power over load level function of the LU workload.

	Interval sizes			
Model	**10%**	**20%**	**25%**	**scattered**
Cubic Spline	0.198%	0.48%	0.901%	0.256%
One Polynomial	1.187%	1.37%	1.472%	1.391%
Linear	0.219%	**0.241%**	0.856%	**0.2%**
Piece-Wise Polynomial (degree 2)	0.226%	0.442%	0.922%	0.268%
Piece-Wise Polynomial (degree 3)	0.239%	0.412%	**0.744%**	0.331%
Split Polynomial (1 break)	0.156%	0.53%	**0.744%**	0.304%
Split Polynomial (4 breaks)	**0.127%**	**0.241%**	0.856%	0.257%
Linear Regression	0.526%	1.305%	1.774%	0.823%

method using the dynamically split polynomial with one breakpoint. It is the most accurate interpolation method in 11 cases, followed by linear interpolation, which is best in seven cases. In the end, though, the exact best polynomial method differs for a given problem and dataset. Finding a good configuration within this space remains a challenge.

13.3 Interpolation using Reference Dataset

Next, we evaluate the efficiency of automatic selection of a good interpolation function using an independent reference dataset. We choose the SSJ measurement set as our reference dataset and then choose the corresponding interpolation function for any given subset based on the optimal interpolation function of SSJ for this given subset.

The default SPEC SERT configuration actually offers a similar use-case as it samples SSJ at a greater number of load levels than for all other workloads. In order to achieve a more thorough evaluation, our sampling rate is higher than SERT's default rate, but the use-case remains similar.

The accuracy of interpolation based on automated interpolation selection and configuration using SSJ as the reference dataset is displayed in Table 13.5. Although interpolation selection based on an independent reference dataset is, of course, less accurate than interpolation using the optimal interpolation function, selected using the actual measurement set for the given workload (see Tabe 13.4), it is still more accurate than all other methods.

Compared to using regression, interpolation using an independent reference dataset features an improvement of 43.6% (mean over the relative improvements) in accuracy. Compared to always choosing the single, most commonly

Table 13.4: Mean modeling errors of best interpolation function for each workload.

Worklet	10% intervals	20% intervals	25% intervals	scattered
SSJ	Polynomial (degree 2) (0.168%)	Polynomial (3 breaks) (0.217%)	Spline (0.224%)	Linear (0.175%)
Compress	Polynomial (3 breaks) (0.165%)	Linear (0.308%)	Linear (0.385%)	Polynomial (3 breaks) (0.277%)
Cryp-toAES	Polynomial (2 breaks) (0.115%)	Linear (0.328%)	Polynomial (1 break) (0.824%)	Polynomial (1 break) (0.214%)
SHA256	Polynomial (degree 3) (0.156%)	Spline (0.186%)	Spline (0.181%)	Polynomial (degree 4) (0.112%)
LU	Polynomial (4 breaks) (0.127%)	Linear (0.241%)	Polynomial (1 break) (0.744%)	Linear (0.2%)
SOR	Polynomial (2 breaks) (0.099%)	Linear (0.22%)	Polynomial (1 break) (0.607%)	Polynomial (1 break) (0.167%)
Sort	Polynomial (2 breaks) (0.148%)	Polynomial (1 break) (0.425%)	Polynomial (1 break) (0.8%)	Polynomial (1 break) (0.271%)
Flood	Polynomial (3 breaks) (0.065%)	Polynomial (degree 2) (0.06%)	Polynomial (degree 2) (0.08%)	Nearest Neighbor (1.023%)
Sequential	Polynomial (1 break) (0.099%)	Polynomial (1 break) (0.226%)	Polynomial (1 break) (0.129%)	Polynomial (2 breaks) (0.089%)
Random	Polynomial (3 breaks) (0.037%)	One Polyno-mial (0.049%)	One Polyno-mial (0.058%)	Polynomial (3 breaks) (0.049%)

best interpolation method (dynamically split polynomial interpolation with one breakpoint), automated selection and configuration still improves model accuracy by 20.025% (mean over the relative improvements).

Table 13.5: Mean modeling errors of independent reference based interpolation for each workload, using SSJ as the reference dataset.

Worklet	**10% intervals** Piece-wise Polynomial (degree 2)	**20% intervals** Split Polynomial (3 breakpoints)	**25% intervals** Cubic Spline	**scattered** Piece-wise Polynomial (degree 1)
Compress	0.229%	0.581%	0.457%	0.278%
CryptoAES	0.165%	0.404%	0.904%	0.296%
LU	0.226%	0.46%	0.901%	0.2%
SHA256	0.197%	0.232%	0.181%	0.174%
SOR	0.156%	0.357%	0.682%	0.206%
Sort	0.207%	0.478%	0.948%	0.417%
Flood	0.898%	2.017%	2.529%	0.787%
Sequential	0.123%	0.255%	0.178%	0.109%
Random	0.039%	0.056%	0.061%	0.058%

13.4 Interpolation using Cross-Validation

Although cross-validation based configuration and selection lacks the additional data of its reference based counterpart, it is still fairly accurate. Specifically it is 31.36% more accurate than linear regression. The full results are displayed in Table 13.6.

The major drawback of the cross-validation based configuration and selection are the different scales between the cross-validation problem and the final dataset to be interpolated. For example, when removing a data point from the 10% utilization interval set, the interpolation function has to interpolate a 20% utilization gap. Its ability to do so is then judged to indicate its accuracy when interpolating equi-distant data-points at 10% intervals.

13.5 Summary

Our evaluation of interpolation accuracy answers **EQ B.1** ("Can interpolation be used to accurately model power consumption per load level on servers?"). It shows that interpolation features superior accuracy compared to regression for bounded problem spaces. In comparison to regression, our automated interpolation method configuration and selection approach improves modeling accuracy by 43.6% if additional reference data is available and by 31.4% if it is not. Our selection method also improves accuracy compared to simply

Table 13.6: Mean modeling errors of cross-validation based interpolation.

Worklet	10% intervals	20% intervals	25% intervals	scattered
SSJ	One Polynomial (0.35%)	Polynomial (degree 2) (0.225%)	Polynomial (1 break) (0.304%)	One Polynomial (0.215%)
Compress	Linear (0.24%)	Polynomial (3 breaks) (0.581%)	Polynomial (2 breaks) (0.509%)	Polynomial (degree 3) (0.277%)
CryptoAES	One Polynomial (0.459%)	Linear (0.328%)	Linear (0.91%)	Polynomial (degree 3) (0.334%)
LU	Polynomial (5 breaks) (0.16%)	Linear (0.241%)	Linear (0.856%)	Linear (0.2%)
SHA256	Spline (0.186%)	One Polynomial (0.351%)	Polynomial (2 breaks) (0.363%)	Polynomial (degree 2) (0.235%)
SOR	Spline (0.135%)	Linear (0.22%)	Polynomial (degree 1) (0.701%)	Polynomial (degree 2) (0.234%)
Sort	Polynomial (1 break) (0.204%)	Linear (0.443%)	Linear (1.026%)	Linear (0.417%)
Flood	Nearest Neighbor (0.727%)	Nearest Neighbor (1.96%)	Nearest Neighbor (2.678%)	Polynomial (5 breaks) (0.79%)
Sequential	Linear (0.138%)	Nearest Neighbor (0.316%)	Polynomial (1 break) (0.129%)	Polynomial (2 breaks) (0.089%)
Random	Polynomial (6 breaks) (0.043%)	One Polynomial (0.049%)	Linear (0.068%)	One Polynomial (0.051%)

choosing the most commonly accurate interpolation method (the method that is the most accurate method for most cases) by 20.0%. As a more general note, we show that power characteristics of a system under observation lend themselves well towards being interpolated if extrapolation is not required.

Chapter 14

Evaluation of Offline Power Prediction

We describe two use-cases for offline power prediction in Chapter 8, each with a corresponding prediction approach. The first approach predicts the power consumption of virtual machine hypervisors , wherease the second predicts the power consumption of a target application. Both models use rating results of the SERT implementation of our power rating methodology in Chapter 4 to predict the consumption of unavailable servers.

In this chapter, we evaluate both approaches., addressing **EQ B.2** ("Can we accurately predict the power consumption of hypervisors and applications on unavailable servers, based on results provided by our rating methodology?"). We train our models using SERT results and reference results measured on local servers and analyze the accuracy of the resulting power prediction. We also analyze potential different training sets and our approaches' configurations as to their impact on accuracy.

14.1 Evaluation of Power Prediction for Virtualized Environments

We evaluate the accuracy of our virtual machine hypervisor power prediction approach in Chapter 8, Section 8.1 by predicting the power consumption of a target system and then comparing the predicted consumption against the actually measured power draw. We perform this prediction for multiple workloads and investigate how prediction accuracy is impacted by the target workload choice. This evaluation addresses **EQ B.2** ("Can we accurately predict the power consumption of hypervisors and applications on unavailable servers, based on results provided by our rating methodology?") regarding prediction of hypervisors.

The systems used in our evaluation have significant differences, as they are from different generations. One system is from 2015 and uses a Haswell generation Intel processor, whereas the other one is an older machine from 2010 with a

Table 14.1: Hardware configuration of servers.

	Reference System	**Target System**
Model	HP ProLiant DL160 Gen9	HP ProLiant DL380 G5
CPU model	Intel Xeon E5-2640 v3	Intel Xeon E5420
Number of cores	8	4
Hardware threads	16	8
CPU frequency	2.60 GHz	2.50 GHz
Memory	2 x 16 GB	2 x 8 GB
Harddisk	1 x 460 GB	1 x 400 GBA3, 1 x 120 GB
Network	1 Gbit/s	1 Gbit/s

Harpertown generation processor. More details on the hardware configuration of the systems used for the evaluation are shown in Table 14.1. These two devices were selected for this evaluation because the power characteristics of the two servers are very different. The reference system has a power consumption in the range between 36.7 W and 141.8 W whereas the target system features an idle power of 237.6 W and a highest measured power consumption of 334.8 W. This means that, because of the significant differences between the devices, there is no intuitive way to guess the power consumption based on simple assumptions and a more complex prediction method is needed.

Both systems run XenServer 6.5 as the hypervisor for their virtualized environments. The virtual machines themselves are running Ubuntu Server (64-bit) 14.04.2 as their guest operating system with Oracle Java Runtime Environment 1.7.0_79. The operating system for the non-virtualized configuration is Debian GNU/Linux 8.0 (Jessie) on the reference system and Ubuntu Server (64-bit) 14.04.2 on the target system.

We measure power consumption of the non-virtualized environment for both systems using our regular measurement methodology from Chapter 4. We then measure the power consumption for each of the three virtualized environment configurations from Table 8.1 in Section 8.1.1.1 on both systems. Finally, using the measurement results from the reference system, we predict the power consumption for each of those configurations on the target system.

14.1.1 Prediction without Sub-Models

Firstly, we investigate if sub-model creation using the self-prediction error as a heuristic is needed. To this end, we predict power consumption using all workloads without any sub-models. Table 14.2 shows the relative predic-

Table 14.2: Relative difference between predicted and actual power (no sub-models).

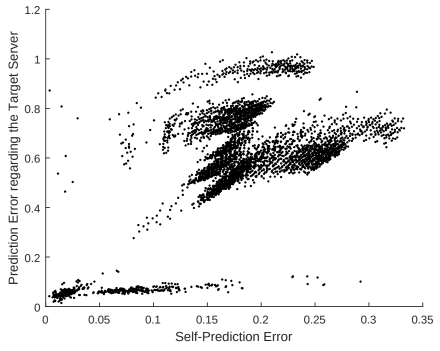
Workload	100% Load	50% Load
Idle	487%	481%
Compress	97%	111%
CryptoAES	95%	110%
LU	92%	108%
SOR	104%	116%
XMLValidate	95%	109
Sort	98%	112%
SHA256	100%	113%
Sequential	53%	55%
Random	447%	448%
SSJ	100%	113%
Flood	208%	203%

tion accuracy using this approach. Even using the most accurate regression implementation (stepwise regression for this specific case), the prediction accuracy without sub-model creation is poor. CPU workloads have a relative error around 100%, with some workloads exhibiting even worse prediction accuracy. This indicates that additional optimization (e.g., using sub-models) is necessary.

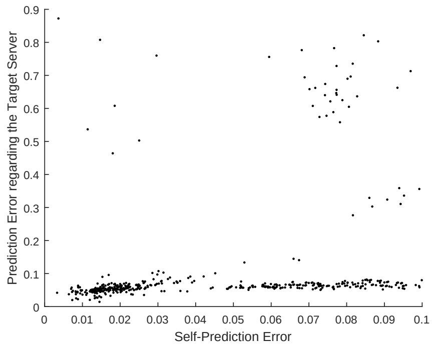
14.1.2 Self-Prediction Error and Actual Prediction Error

As an intermediate step of the evaluation, we investigate if the self-prediction error of the reference system is actually a good indicator for the final prediction error of a sub-model. The relationship between a sub-model's self-prediction error on the reference system and the actual prediction error regarding the target system for the virtualized environment configuration with one VM is shown in Fig. 14.1. The figure shows that the self-prediction error is a good indicator for sub-models, as long as the self-prediction error remains low. The great majority of sub-models with self-prediction errors of less than 10% also has an actual prediction error of less than 10%.

Sub-models with a self-prediction error of more than 10% are far more volatile and cover a wide range of actual prediction errors. Fortunately, this is not very relevant for our prediction approach, as it always selects the sub-models with the lowest self-prediction error. These observations repeat for the other two virtualized environment configurations.



(a) All sub-models.

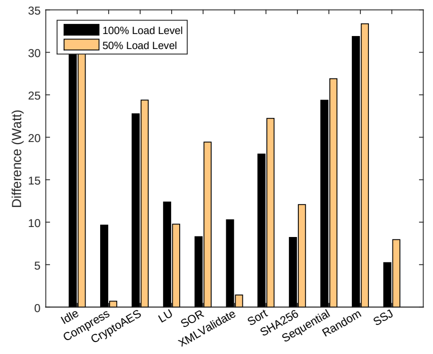


(b) Models with self-prediction error ≤ 0.1.

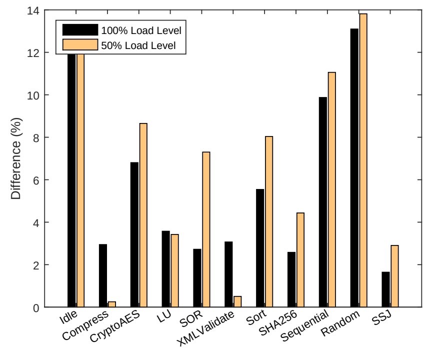
Figure 14.1: Relationship between self-prediction error and actual prediction error.

14.1.3 Prediction with Sub-Models

Finally, we evaluate the prediction error of sub-models obtained using the self-prediction heuristic. We measure the absolute and relative difference between the measured power consumption on the target system and the predicted power. The prediction errors for the virtualized environment configuration with one VM (configuration #1 in Table 8.1) are shown in Fig. 14.2.



(a) Absolute Difference



(b) Relative Difference

Figure 14.2: Prediction errors for configuration with 1 VM.

The CPU-heavy workloads all feature a prediction error of less than 10%, with Compress and XMLValidate even featuring a relative prediction error of less than 1% at 50% load. CryptoAES prediction is the least accurate among the CPU workloads, yet its prediction error does not exceed 10%. The relative prediction inaccuracy for CryptoAES is easily explained as the workload makes use of the specialized AES CPU instruction set on the reference system. The target system's CPU does not use this instruction set as it is too old.

Idle and the Storage workloads (Sequential and Random) are not as accurate as the CPU workloads with prediction errors between 10% and 14%. Storage power prediction is difficult, as the systems have very different storage configurations (1 HDD vs. 2 HDDs) and Idle prediction is hard, since no other workload in this collection resembles it. Under these conditions, the prediction is still quite accurate. The only workload that does not feature any accurate predictions is the Flood workload. It is missing in Fig. 14.2 because it features a prediction error of 117% and 118% for its two load levels. This can be attributed to two factors: The difference in system memory and the absence of any other workload that resembles Flood and that might help to increase its prediction accuracy.

The virtualized environments configurations with two VMs can also be predicted with good accuracy. CPU and storage workloads still remain in the accuracy range of prediction errors less than 15%. Idle is predicted with slightly diminished accuracy (18% relative error). The only exception is the CryptoAES workload. Due to its use of a specialized hardware instruction set, it struggles severely with virtual machine over-provisioning (configuration #2 in Table 8.1). Performance on the reference system itself is already impacted significantly by the virtualized environment configuration. As a result, the prediction accuracy for this configuration suffers and drops to 22% and 24%, respectively.

Summarizing, we evaluate our offline prediction mechanism for virtual machine hypervisor power consumption, answering **EQ B.2** ("Can we accurately predict the power consumption of hypervisors and applications on unavailable servers, based on results provided by our rating methodology?") for this context. We evaluate the benefits of partitioning our overall model into sub-models using self-prediction accuracy as a heuristic. The results show that power consumption of CPU and storage loads can be reliably predicted with a prediction error of less than 15% across all tested virtualized environment configurations.

14.2 Evaluation Offline Power Prediction for Target Applications

We evaluate our offline prediction model in Chapter 8, Section 8.2 using three target applications and three different real-world, physical servers. This evaluation addresses **EQ B.2** ("Can we accurately predict the power consumption of hypervisors and applications on unavailable servers, based on results provided by our rating methodology?") regarding software applications on unavailable servers. We run the SPEC SERT on each of those servers to measure and characterize its power consumption. We then predict the target applications' power consumption when running on two of those servers using the respective target server's SERT and a training set consisting of a SERT result and target application power measurements from one of the servers. We compare measured application power consumption with the predicted consumption for all measured load levels and servers and calculate the aggregate prediction error using the mean absolute percentage error (MAPE).

We consider the following three applications:

- **Pi**: A worklet that ships with the SPEC ChauffeurWDK (Arnold, 2013) and computes Pi by calculating up to 100000 iterations of the Gregory-Leibniz series.

- **Friendgraph**: Friendgraph is supposed to emulate a simple social network graph of "friends", which store arbitrary numeric properties within a matrix. A transaction calculates "friend"-value by aggregating a friend's matrix with all of its first and second-order friends.

- **Dell DVD Store** (Dell, Inc., 2011): The Dell DVD Store is a test web application developed by Dell. We use its PHP implementation with a MariaDB database and deploy using Docker. We generate load using users who browse the store in a cyclical pattern. Users log in, search for items, add one item to the cart, and then log out. We choose the DVD Store instead of the TeaStore from Chapter 6 considering that the DVD Store is primarily deployed as a single instance on a single machine, which fits the use-case of this evaluation.

All servers in our experiment run Debian 9.4 (Kernel 4.9.82), with Docker 18.03.0-ce, and Java HotSpot 64-Bit (build 25.161-b12). Table 14.3 shows the hardware configuration of our three testing devices, which we identify using the core-count.

Table 14.3: System under test specification including power characteristics measured using SERT.

	4 core	**8 core**	**10 core**
Model	HP ProLiant DL20	HP ProLiant DL160	HP ProLiant DL160
CPU	4 cores	8 cores	10 cores
Xeon Model	E3-1230 v5	E5-2640 v3	E5-2650 v3
Clock	3.4 GHz	2.6 GHz	2.3 GHz
Generation	Skylake	Haswell	Haswell
Memory	2 x 8 GB	2 x 16 GB	2 x 16 GB
Storage	1 x 460 GB	1 x 460 GB	1 x 460 GB
Idle Pwr	28.3 W	39.6 W	42.6 W
Max Pwr	106.1 W	139.7 W	151.8 W

14.2.1 Measuring Target Application Power and Performance

We measure the target application power consumption and performance using our power measurement methodology of Chapter 4. We do this by implementing our workloads within the SPEC ChauffeurWDK (Arnold, 2013) or by implementing a load driver in Chauffeur to drive the external workload in case of the DVD Store. Using this methodology and implementation, we measure the target applications' power and performance (throughput) at four target load levels: 25%, 50%, 75%, and 100%. We connect an external power measurement device to the AC power inlet of the SUT. An external director machine controls the experiment using a network connection to the SUT. The SUT runs Chauffeur's *host*-software, which launches client processes that execute the workload or delegate it to the DVD Store using HTTP.

Applying our power methodologies default durations, we perform a warmup run for 30 seconds and then calibrate the maximum load level of the target application by running it on parallel on every available hardware thread. We repeat this calibration two times, with each separate run having a pre-measurement duration of 15 seconds, a measurement duration of 120 seconds, and a post-measurement duration of 15 seconds.

After calibration, we measure performance and power consumption for the target load levels in descending order. The load levels' pre-measurement, post-measurement, and measurement times are equal to the calibration. Our reported per-load-level power and performance results are the average of the 120 second measurements. The maximum coefficient of variation during our

Table 14.4: Unoptimized, baseline prediction errors of base formalisms.

Formalism	Avg. MAPE
Gradient Tree Boost	32.94%
Regression Tree	19.42%
Random Forest	31.03%
Gaussian Regression	100%

measurement runs is the performance variation of the Dell DVD Store at full load on the 10 core machine. It features a CV of 1.6%, which is well below the maximum boundary of 5%.

14.2.2 Unoptimized Power Prediction

Firstly, we analyze the accuracy of our prediction formalisms without interpolation, optimization, and formalism selection. This analysis serves as a baseline to classify room for improvement for our method. When not using interpolation, we only use worklets with at least four load levels, meaning that the memory and storage worklets are discarded. For the SSJ worklet, we discard the four of the eight load levels that do not appear in the other worklets. Table 14.4 shows the prediction errors of this baseline method. Considering that parameter optimization is removed, we set each of the regression formalism's parameters to the minimum parameter of our parameter space.

The results in Table 14.4 show differences in the prediction accuracies of the underlying formalisms. Yet, even the more accurate methods suffer from poor accuracy when not optimized and without any interpolation. In general, the regression mechanisms seem to be able to capture the problem with some error. Gradient Tree Boost and Random Forest both feature a mean absolute percentage error of slightly more than 30%. Even though these two formalisms achieve similar average results, they differ in terms of variance. Gradiant Tree Boost's MAPE has a standard deviation of 11.8%, whereas the Random Forest has a lower variation of 7.7%. Both methods can not achieve the accuracy of the regular Regression Tree, though. It has an average MAPE of 19.42% with a standard deviation of only 2.6%.

We attribute this to the relatively little amount of training data available. Four load levels per worklet do not provide sufficient data for an accurate training of all formalisms. The Gaussian Regression and Neural Networks are most affected by this. Seven worklets with at least four load levels plus idle seems to not provide sufficient data to derive multiple Gaussian distributions.

Table 14.5: Formalism prediction error with adaptive interpolation and parameter optimization.

Formalism	Avg. MAPE
Gradient Tree Boost	32.94%
Regression Tree	10.89%
Random Forest	32.05%

Consequently, we do not consider Gaussian Mixture Models in the following tests.

14.2.3 Predicting Power using Interpolation and Optimization

Our baseline tests indicate that the amount of data provided by SERT results is too small for most prediction methods. Consequently, we enable interpolation in conjunction with our prediction formalism parameter optimization method. The interpolation mechanism ensures that no measurement must be discarded in worklets with high load level counts. Specifically, it creates artificial load levels for all worklets with fewer levels than SSJ, which is the worklet featuring the most load levels. As a result, all worklets are padded to a total of nine load levels (eight SSJ load levels plus idle power). We use all worklets with a minimum of four load levels, enabling the use of adaptive interpolation. Paramter optimization is left at default settings, meaning that it uses as many iterations as the underlying method has parameters for optimization. Our regression formalisms feature two parameters, each, resulting in an iteration count of two.

The prediction accuracy of our three regression formalisms used with adaptive load-level interpolation and parameter optimization is shown in Table 14.5. Gradient Tree Boost' accuracy improves, but only by 0.004% percentage points, which is not significant. Random Forest's accuracy even decreases by 1.2% points. However, Regression Tree improves significantly. With optimization and interpolation enabled, it features an prediction error of 10.89%, which is an improvement of 8.5 percentage points and relative improvement of 43.9% compared to our baseline experiment.

The amount of data provided by interpolated results seems to be sufficient for a tuned regression tree. In contrast, the other two regression methods do not improve significantly, which leads to conclude that the interpolated SERT results did still not produce sufficient data for training of accurate models for these formalisms.

Table 14.6: Formalism prediction error with parameter optimization, but not using any interpolation.

Formalism	Avg. MAPE
Gradient Tree Boost	32.94%
Regression Tree	19.42%
Random Forest	32.26%

14.2.4 Parameter Optimization and no Interpolation

The predictions using parameter optimization and interpolation in Section 14.2.3, significantly improve the prediction accuracy of the baseline formalisms. The part played by the parameter optimization and interpolation, respectively, in this improvement remains an open question. To investigate if parameter optimization caused this improvement by itself, we test our prediction formalisms using parameter optimization only. That means that we use four load levels only, discarding excess SSJ load levels and worklets with fewer load levels, similar to the baseline approach in Section 14.2.3.

Table 14.6 shows the prediction errors of using parameter optimization without interpolation. In general, the differences to baseline prediction are very small. Gradient Tree Boost and Regression Tree do, in fact, see no change. Random Forest's accuracy, on the other hand, decreases. This would indicate that parameter optimization is of no help for such a small dataset and the chosen prediction formalisms. However, accuracy increased in the results of Section 14.2.3, where optimization was used. We investigate this further in the following tests.

14.2.5 Interpolation with Baseline Parameters

Our tests using optimization without any interpolation in Section 14.2.4 do not show any significant improvement in prediction accuracy. Consequently, we investigate if interpolation achieves the improvement of Section 14.2.3 by itself. To test this, we again apply adaptive interpolation to all worklets with four load levels in order to match the load level count of SSJ (eight plus idle). We also, again, interpolate the reference workload to nine load levels.

Table 14.7 shows the prediction accuracy when using the baseline parameter set on interpolated training and prediction data. Interestingly, these results do also not differ significantly from the baseline. Gradient Tree Boost and Random forest improve marginally by less than 1 percentage point. Due to the lack of

Table 14.7: Formalism prediction error with interpolation, but not using any parameter optimization.

Formalism	**Avg. MAPE**
Gradient Tree Boost	32.91%
Regression Tree	20.40%
Random Forest	30.41%

optimization, Regression Tree's accuracy decreases when using interpolation without any parameter optimization.

These results indicate that neither parameter optimization nor interpolation increase prediction accuracy by themselves. In addition, they disprove the potential conclusion from the optimization-only results in Section 14.2.4 that parameter optimization does not improve prediction accuracy and improvements are instead due to interpolation. Instead, the separate tests using interpolation and parameter optimization only clearly indicate that the combination of both is responsible for the improvements in prediction accuracy. Interpolation ensures that sufficient data is available to run optimization against and parameter optimization configures the prediction formalisms for the interpolated dataset.

14.2.6 Prediction Accuracy depending on Interpolation Method

Our results in Section 14.2.3 show that interpolation helps in increasing prediction accuracy, particularly when using the Regression Tree prediction formalism. Those experiments used an adaptive interpolation approach to generate additional data for training. We investigate the choice of the specific interpolation method and its impact on accuracy. We focus on the Regression Tree prediction formalisms, as it shows the greatest sensitivity to interpolation and parameter optimization in addition to usually providing the most accurate predictions. Table 14.8 shows the prediction errors of using Regression Tree with parameter optimization based on datasets prepared with the different interpolation methods.

We investigate *zero padding* as an alternate approach to the baseline and to interpolation. Zero padding of missing results allows us to not discard excess SSJ load levels and still meet the constraints of the prediction methods. As seen in Table 14.8, using zero padding results in a Regression Tree prediction error of 19.91%, which is slightly less accurate than the baseline prediction. In contrast, the prediction errors when using the interpolation methods are all significant improvements compared to the baseline. *Nearest Neighbor Interpolation* features

Table 14.8: Regression Tree prediction error depending on interpolation method.

Interpolation Method	Avg. MAPE
Zero Padding	19.91%
Nearest Neighbor	15.49%
Linear Interpolation	9.49%
Adaptive Interpolation	10.89%

the smallest improvement in accuracy, improving the average prediction MAPE by only 3.92 percentage points. Adaptive Interpolation improves the prediction by 8.5 percentage points,which is a relative improvement of 43.9% compared to our baseline experiment.

Interestingly, linear interpolation provides the most accurate predictions with a MAPE of 9.49%. This is a relative improvement of 51% compared to the baseline Regression Tree prediction error. Again, we surmise that the small size of the dataset is a cause for this, as the adaptive interpolation has too few datapoints (usually four) to adapt itself. Instead, simply picking an interpolation option that always delivers good results and is applicable to such small sets, seems to lead to more accurate predictions. Figure 14.3 shows the results of power prediction for the Friendgraph worklet using linear interpolation. As a sidenote, including the storage worklets in predictions using linear interpolation does not increase prediction accuracy, leading to a prediction error of 14.09%.

Summarizing, the evaluation of our offline application power prediction approach addresses **EQ B.2** ("Can we accurately predict the power consumption of hypervisors and applications on unavailable servers, based on results provided by our rating methodology?") regarding applications on unavailable servers. It shows that our offline power prediction method can accurately predict the power consumption of a target server using SERT results. It also shows that the combination of parameter optimization and interpolation can significantly increase the prediction accuracy compared to baseline formalisms by as much as 51%, even though both interpolation and parameter optimization do not achieve significant improvements in accuracy on their own.

14.3 Summary

The evaluations of this chapter address **EQ B.2** ("Can we accurately predict the power consumption of hypervisors and applications on unavailable servers,

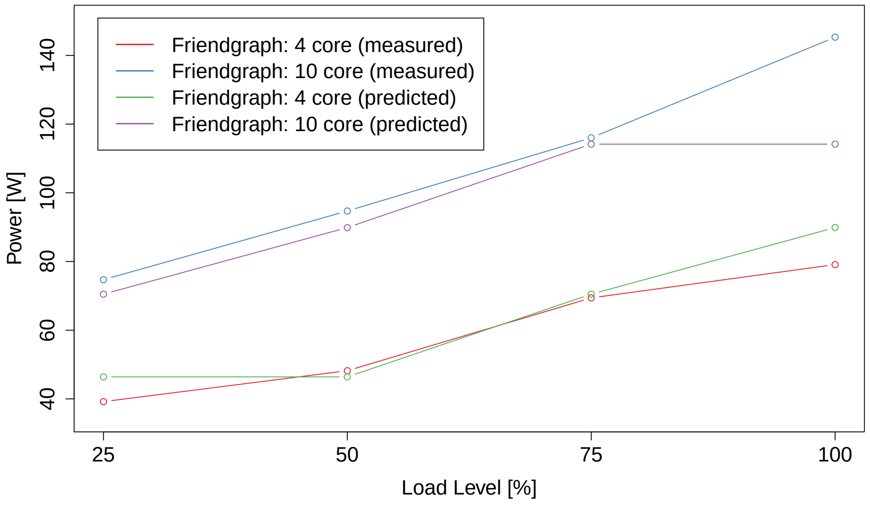


Figure 14.3: Measurements and Regression Tree prediction (using linear interpolation) of Friendgraph worklet power.

based on results provided by our rating methodology?”). We evaluate our offline application power prediction approach and show that it can accurately predict the power consumption of a target server using SERT results. It also shows that the combination of parameter optimization and interpolation can significantly increase the prediction accuracy compared to baseline formalisms by as much as 51%, even though both interpolation and parameter optimization do not achieve significant improvements in accuracy on their own.

In addition, we evaluate our offline prediction mechanism for virtual machine hypervisor power consumption. We evaluate the benefits of partitioning our overall model into sub-models using self-prediction accuracy as a heuristic. The results show that power consumption of CPU and storage loads can be reliably predicted with a prediction error of less than 15% across all tested virtualized environment configurations.

Chapter 15

Applying Online Power Prediction using Real-World Workloads

We evaluate the applicability and accuracy of our online power prediction models from Chapter9, addressing **EQ B.3** ("Can we accurately predict the power consumption of a potential software component placement based on run-time data?"). We test their ability to predict power consumption in a run-time scenario under varying load in a heterogeneous environment. We run two common web applications in a distributed environment consisting of two different servers using CPUs of different architectures.

15.1 Methodology

The two test applications are deployed on two separate physical servers, which are put under load by a load generator running on a separate machine. The load generator collects application level performance data and is controlled by a dedicated experiment controller. The controller also collects power measurements using external measurement devices.

15.1.1 Experiment Setup

The test applications are executed on two different servers from different generations and using different processor architectures. Their specifications are shown in Table 15.1. The two SUTs feature different memory sizes and CPUs of different sizes and architectures. The smaller system (SUT 1) has a quad-core CPU of the Skylake architecture, running at 3.4 GHz, whereas the larger system (SUT 2) has an 8-core CPU of the older Haswell generation running at 2.6 GHz. The two systems differ in their performance and power consumption. Table 15.1 shows the differences in power consumption measured by running SERT on the SUTs.

Table 15.1: System under test specification including power characteristics as measured using SERT.

	SUT 1	**SUT 2**
Model	HP ProLiant DL20	HP ProLiant DL160
CPU	Intel Xeon E3-1230 v5, 4 cores at 3.4 GHz	Intel Xeon E5-2640 v3, 8 cores a 2.6 GHz
Memory	2 x 8 GB	2 x 16 GB
Storage	1 x 460 GB	1 x 460 GB
Idle Power	27.1 W	36.4 W
Max Power	98.1 W	140.6 W

The SUTs are connected to a load generator machine that can send requests to either or both machines, depending on the current component deployment. In addition, the SUTs are connected using a separate network for communicating with each other. The load generator and SUTs are connected to an external experiment controller. The controller collects application level performance metrics from the load generator, as well as system metrics (CPU performance counters) from the SUTs themselves. The system metrics are needed for training the single server power prediction model. The controller also communicates with the external Yokogawa WT310 power measurement devices, which are connected to the SUTs and measure their DC power consumption at the power inlet. Fig. 15.1 shows the overall experiment setup.

15.1.2 Test Applications

We use two common 2-tier, PHP-based, web applications for testing. The advantage of using PHP-based web applications is that we can emulate the effect of different (functionally identical) implementations by exchanging the web server (and PHP module) on which the applications are run. We use the following two applications:

- Dell DVD Store 6.1.1.4: A test application developed by Dell that models a web store for DVDs (Dell, Inc., 2011).
- RUBiS 2.3.2: The Rice University Bidding System that emulates an Ebay-like auction platform (*RUBiS User's Manual* 2008).

We use three different web servers to execute the web tier of the applications: Apache HTTP Server 2.4.10, Lighttpd 1.4.35, and Nginx 1.6.2, each with their corresponding PHP modules. We use MySQL Server 5.5.54 as the database

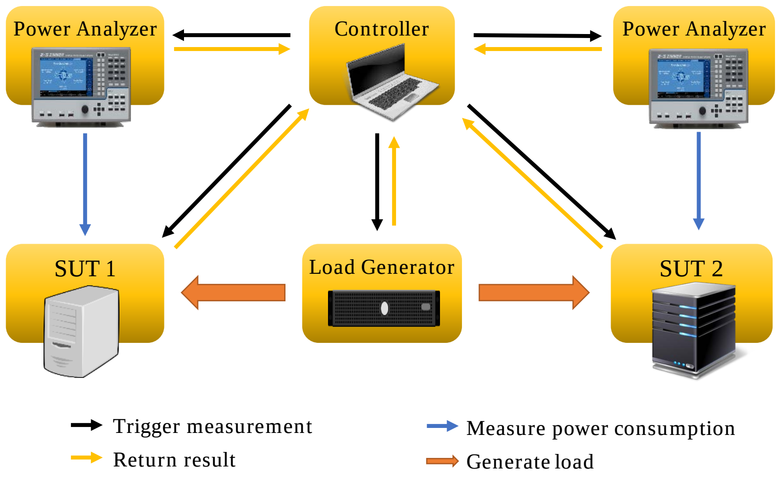


Figure 15.1: Model evaluation device setup.

for our data tier. We design our usage profiles to be read-only, thus posing no constraints on database replication in order to allow evaluation of as many replication strategies as possible. This means that the web servers and the database may run on either one or on both SUTs, and different web servers may run on the two SUTs in case both of them are used for the web tier.

We derive 15 different deployment options from the three different web tier implementations and the database, with the two SUTs as deployment targets.

15.1.3 Load Generation

The load generator is designed to generate loads with a varying load intensity. To this end, we assume an open workload model in which requests arrive with an independent and variable arrival rate. The varying arrival rate is specified using the DLIM formalism implemented in the LIMBO tool (see Chapter 5). We use LIMBO to specify a varying load intensity for our training scenarios. This generated load intensity may be continuous and cover all load intensity ranges between system idle and the maximum throughput. In some scenarios, we use a load intensity definition where certain load intensity ranges are omitted on purpose. We do this when training for evaluation of the prediction accuracy when predicting power consumption for an unknown load intensity.

For load generation, the arrival rate is sampled from the load intensity description with a sampling interval of one second. At each point in time the load generator reaches the target arrival rate by generating the target count of requests during the given second. During each second, it proceeds according to the standard load generation behavior defined in our measurement and rating methodology of Chapter 4. Work units are scheduled in batches and random, exponentially distributed wait times are added in between the scheduling of each batch. Wait times and batch sizes are chosen so that the target arrival rate is reached as accurately as possible.

The request types (determining the actual content of requests) sent to the web applications follow a cyclic pattern, which is based on a closed workload model. This means that, whereas the time of each request is chosen according to an open workload model, the request type and content are stateful and depend on the previous responses. The type of generated requests are based on a user profile that emulates a user browsing the store(s), viewing random items, and adding them to the shopping cart.

The load generator distributes load according to the round robin scheme when generating load for multiple web-tier instances.

15.1.4 Measurement Methodology

We collect aggregate performance and power measurements on a per-second basis. Power measurements are made at the DC power inlets of the servers. The Yokogawa WT310 power measurement devices report average DC power in watts with a maximum measurement uncertainty of 1%. Application level performance (load intensity and throughput) is measured directly at the load generator and collected by the experiment controller.

When collecting performance counters, performance counter measurement daemons based on the Intel Performance Counter Monitor (Intel PCM, see Willhalm et al., 2012) are executed on the SUTs. These daemons aggregate performance counters on a per-second basis and send the aggregate measurements to the controller.

15.1.5 Metrics

We use the following metrics for evaluating the accuracy of the power prediction models in our prediction scenarios:

- Mean Absolute Error (MAE) in watts
- Median Absolute Error (MdAE) in watts
- Mean Absolute Percentage Error (MAPE)
- Median Absolute Percentage Error (MdAPE)

Table 15.2: Power consumption of the most and least power consuming deployment configuration at full load.

Deployment Config.	**SUT 1**	**SUT 2**	**Total**
SUT 1: Apache; SUT 2: MySQL	81.56 W	54.45 W	136.01 W
Both SUTs: Lighttp + MySQL	45.83 W	59.27 W	105.1 W
Difference	35.73 W	-4.82 W	**30.91 W**

- Minimum and Maximum Absolute Errors in watts

15.2 Power Saving Potential

To demonstrate the range of potential power savings we execute all deployment variants and measure their power consumption at full load. The difference between the maximum and minimum power consumption is a strong indicator of the savings potential that can be achieved by an accurate prediction mechanism. Table 15.2 shows the single server and total power consumption and the specific deployments that use the most and least power at their respective maximum load.

Note that these are only measurements at maximum load, regardless of the throughput at maximum load. This means that the least power consuming deployment at maximum load does not necessarily have to be the least power consuming deployment at all load levels. Also note that the maximum load level differs for the different deployment configurations, as some deployments have a higher throughput capacity than others.

The observed difference in power consumption between the most and least consuming deployment configuration of 30.91 W (29.4%) also offers a bound for model prediction errors, as this is the maximum deviation of power consumption that can actually occur in practice in our scenario.

Table 15.3: Gradient Tree Boost prediction accuracy for # training vectors (total # vectors for 8 training deployments).

# Vectors	**MAE**	**MAPE**	**MdAE**	**Error Interval**
280	2.69 W	2.70%	1.84 W	[-0.50 W, 4.38 W]
560	2.63 W	2.48%	1.70 W	[-6.23 W, 3.47 W]
1600	2.08 W	2.17%	1.21 W	[-4.71 W, 4.51 W]
2400	1.05 W	1.25%	0.83 W	[-6.08 W, 3.09 W]

Table 15.4: Regression Tree prediction accuracy depending on # training vectors (total # vectors for 8 training deployments).

# Vectors	MAE	MAPE	MdAE	Error Interval
280	1.61 W	1.59%	0.52 W	[-5.02 W, 3.12 W]
560	3.46 W	3.40%	2.84 W	[-4.13 W, 9.94 W]
1600	5.01 W	5.11%	3.21 W	[-8.15 W, 7.75 W]
2400	0.90 W	1.12%	0.95 W	[-1.35 W, 3.72 W]

15.3 Workload Deployment Power Prediction

We evaluate the accuracy of the workload deployment power prediction model in two steps. Firstly, we evaluate the accuracy of the prediction depending on the amount of training data. We also use the opportunity to evaluate the accuracy and applicability of the different prediction methods introduced in Section 9.2.1.1. Next, we investigate the model's accuracy for the two primary application scenarios: prediction of power consumption for previously unobserved deployments and prediction of power consumption for previously unobserved throughput levels. We also investigate if both scenarios can be combined.

15.3.1 Number of Training Vectors and Prediction Accuracy

We evaluate the prediction accuracy of the different prediction methods that can be used to implement the workload deployment prediction model considering the influence of the number of training vectors. For this evaluation, we train the model with eight separate deployments for the Dell DVD Store. Each deployment is executed and instrumented for n seconds, where n is the number of target training vectors for this deployment. We generate a linearly increasing load intensity that covers all load levels from idle to the maximum load of the configuration with the highest load capacity (ca. 15000 transactions per second). We evaluate the mean prediction error when predicting the power consumption.

Tables 15.3 and 15.4 show the prediction errors for the gradient tree boost and regression tree prediction methods. Gradient tree boost exhibits an increase in prediction accuracy with the number of training vectors. The mean and median errors decrease as the number of vectors increases. All errors using gradient tree boost are well below our upper error bound of 29.4%.

Table 15.5: Prediction error for previously unobserved deployments depending on # trained deployments and # Web UI component implementations.

#Tr. Dep.	**#UI**	**MAE**	**MAPE**	**MdAE**	**Error Interval**
3	2	1.98 W	2.18%	1.54 W	[-3.9 W, 4.3 W]
5	2	12.4 W	12.48%	10.47 W	[-17.1 W, 1.2 W]
7	2	6.41 W	6.68%	5.89 W	[-11.1 W, 3.8 W]
11	3	3.1 W	3.17%	2.51 W	[-5.3 W, 4.2 W]
14	3	1.9 W	2.21%	1.62 W	[-4.9 W, 3.2 W]

The regression tree exhibits a different behavior with respect to the number of training vectors. Its accuracy decreases at first as the number of training vectors increases. This is a strong indicator for model overfitting in the case of lower number of training vectors. The accuracy of the regression tree increases for larger training data amounts, after manual reconfiguration for those amounts of training data. This behavior is expected, considering that the boosting mechanism in gradient tree boost is specifically designed to prevent overfitting.

The random forest exhibits the lowest accuracy of all prediction methods. It only manages to reach a minimum prediction error of 1.85 W (2.28%) at 2400 training vectors (300 per trained deployment). Because of the overfitting behavior of the regression trees and the lower accuracy of the random forest, we focus on the use of our models with the gradient tree boost as prediction method.

15.3.2 Predicting Previously Unobserved Deployments

To evaluate the prediction accuracy for previously unobserved deployments we train a varying number of deployments. Each of these deployments is trained using a linearly increasing load profile over 300 seconds (resulting in 300 training vectors per deployment). We then evaluate the accuracy of the resulting power prediction model for a deployment that is not in the set of training deployments.

Table 15.5 shows the power prediction errors using gradient tree boost depending on the number of training deployments as well as the number of different Web UI component implementations that occur in the set of training deployments. The table shows that prediction accuracy generally increases significantly with additional training deployments, with exception of one outlier. The predictions for all training data sets, except for the one with five deploy-

Table 15.6: Deployments in the deployment datasets of size three and five and deployment to predict.

Tr. Set	Max Thr.	Max power	SUT 1	SUT 2
3 & 5	14500	126.1 W	Apache	DB
3 & 5	14500	118.5 W	Lighttpd	DB
3 & 5	14500	104.2 W	Light+DB	Light+DB
5	9000	132.1 W	DB	Apache
5	12000	102.4 W	DB	Lighttpd
none	12000	108.7 W	Apache+DB	Apache+DB

ments, have an error of less than 10%, well below our upper bound of 29.4%. However, for our dataset, the prediction using only three training deployments is significantly more accurate than the prediction using a training set of size five or seven. This effect indicates that the three deployments used for the smallest training set are similar to the predicted one, leading to model overfitting. In addition, the greater prediction error for five training deployments is not due to outliers, as the median error increases as well.

We investigate the specific deployments used for the two prediction scenarios with three and five deployments, respectively. Table 15.6 shows the deployments for the training sets of size three and five and the deployment to predict (the deployment not in any of the training sets in Table 15.5). Table 15.6 also shows the maximum power consumption of the different deployments. None of the training deployment features a deployment in which the Apache WebUI is co-located with the database. However, the smallest training set features one deployment with co-location that consumes a similar amount of power compared to the target deployment. When training using this smallest training set, the gradient tree boost appears to overfit using this training deployment. In return, the two additional training deployments in the training set of size five are dissimilar to the target set both in power consumption and the deployment itself, resulting in a significant error. However, even the error of 12.48% is well below the upper bound of 29.4%.

15.3.3 Power Prediction for Previously Unobserved Throughput Levels

Another potential scenario for prediction using our run-time model is the prediction of power consumption for a load intensity that has not been observed in the training set. This type of prediction may be used to optimize the deployment in preparation for an anticipated load spike.

Table 15.7: Prediction error for previously unobserved throughput depending on # trained deployments.

#Train. Depl.	**Observed Depl.**	**Observed Thr.**	**MAE**	**MAPE**
3	Yes	Yes	0.86 W	0.93%
3	Yes	**No**	**1.29 W**	**1.35%**
14	No	Yes	0.9 W	1.04%
14	No	**No**	**1.1 W**	**1.22%**

For this evaluation, we use the same deployment training sets as in Section 15.3.2. Again, we use 300 training vectors per deployment. The training vectors include power and throughput values for all load levels, except the ranges between 3000 and 6000 requests per second and 10000 and 13000 requests per second. When predicting for the target deployments, we predict 35 power consumption samples for throughputs in the omited ranges. Specifically, we predict the power consumption for throughputs of 4500 and 11500 requests per seccond.

Table 15.7 shows the errors of the prediction. We compare prediction for throughput levels in the training set with prediction for throughput levels not in the training set; we also evaluate the prediction accuracy for scenarios where both the throughput level and the deployment are not in the training set. Our regression-based power extrapolates the power consumption for previously unobserved load intensity ranges with high accuracy. The prediction for observed and unobserved throughput levels differ by less than 0.4 watts. In addition, the prediction error for scenarios where both the throughput level and deployment are unobserved is not significantly higher compared to scenarios where only the deployment is unobserved. In our case, the difference is only 0.2 watts.

15.3.4 Evaluation Results for RUBiS

To show the more general applicability of our approach, we also evaluate its accuracy when applying it to the RUBiS web application. RUBiS is also a two tier web application, which allows the use of the same deployments we used for the DVD Store. As previously, we use gradient tree boost and collect 300 training vectors per deployment in the training set.

Our model exhibits slightly better accuracy for RUBiS compared to the DVD Store. Table 15.8 shows this for the prediction results using the full training

Table 15.8: Prediction error comparison for RUBiS and Dell DVD Store.

Application	**#Train. Depl.**	**Observed Depl.**	**MAE**
Dell DVD Store	14	No	1.9 W
RUBiS	14	No	1.25 W

set. The power consumption of the remaining deployment is predicted with a slightly better accuracy exhibiting an error of 1.25 watts. Overall, our measurements using RUBiS confirm our previous observations in the context of the DVD Store.

15.4 Single Server Power Prediction

The single server power prediction model is designed to enable the use of our workload power prediction model without requiring the use of power measurement devices at run-time. Instead it uses CPU performance counters and learns their relationship to power consumption during a SERT run. The question which SERT worklets and which specific performance counters should be used is not trivial and poses a research challenge. We evaluate multiple combinations of SERT worklets and performance counters against three target workloads, each deployed in a non-distributed setting on a single system: DVD Store, RUBiS and SSJ. SSJ is the hybrid worklet in SERT designed to resemble a real world enterprise application.

We evaluate the mean prediction error for both servers (SUTs) in our testbed considering multiple prediction scenarios at various load levels between 100 and 1400 transactions per second.

Table 15.9 shows the prediction errors for single server power consumption for selected performance counter – worklet combinations. The results support our decision to use the *CPU power* and *Memory Bytes Write* counters, as using these counters leads to results with the smallest errors. Using only the CRYPTO-AES worklet for training produces the most accurate prediction results using these two counters. This may be an indicator for some overfitting. However, training using CRYPTO-AES produces the most accurate results for all of our prediction workloads (with exception of the case where we predict SSJ with a training set that includes SSJ). We use this configuration in the rest of this evaluation, but admit that potential overfitting might require futher investigation.

Table 15.9: Single server power prediction error depending on training worklets and used performance counters.

Worklets	**CPU Performance Counters**	**MAE**		
		DVD Store	RUBiS	SSJ
CRYPTO-AES, SSJ, SOR, LU	CPU-Power, IO-Bytes, Memory-Bytes-Write, CPU-Cycles, CPU-Ref-Cycles, Memory-Bytes-Read, CPU-Frequency	4.97 W	4.48 W	3.85 W
CRYPTO-AES	CPU-Power, IO-Bytes, Memory-Bytes-Write, CPU-Cycles, CPU-Ref-Cycles, Memory-Bytes-Read, CPU-Frequency	2.65 W	2.51 W	2.81 W
COMPRESS, Crypto-AES, SSJ, XML_Validate	CPU-Power, Memory-Bytes-Write	2.16 W	2.47 W	2.35 W
CRYPTO-AES	CPU-Power, Memory-Bytes-Write	**1.11 W**	**0.94 W**	**1.63 W**
CRYPTO-AES	CPU-Power	1.34 W	1.15 W	1.71 W
SSJ	CPU-Power, Memory-Bytes-Write	1.56 W	1.76 W	1.08 W
CRYPTO-AES	CPU-Power, Memory-Bytes-Write, Ref-Cycle	3.14 W	2.92 W	3.74 W
CRYPTO-AES	CPU-Power, Memory-Bytes-Write, Memory-Bytes-Read	2.41 W	2.44 W	2.53 W

15.5 Combined Models Prediction Accuracy

The end-to-end scenario for applying our modeling approach includes using both the workload deployment power prediction model and calibrated single server power models for all SUTs. We evaluate the prediction error of the workload deployment power prediction model when using the calibrated single server power models to estimate the power consumption of each SUT. Both SUTs are instrumented with performance counter listeners, based on Intel PCM (Willhalm et al., 2012), to collect performance counters at run-time. These performance counters are used to estimate the current power draw of the SUT.

To evaluate the impact of the error added by using the single server power prediction models, we choose a self-prediction scenario for the workload deployment model. The deployment model is trained with all of our 15 deployment

Table 15.10: Error of using both models vs. deployment only.

System Power	MAE	MAPE
Measured	1.38 W	1.46%
Model	1.8 W	2.01%

options and we evaluate its accuracy when predicting the power consumption of one of the training models for various target throughput levels. We compare the relative and absolute errors of the power prediction when using both models together against the power consumption measured by the dedicated power measurement devices for the predicted deployment.

Table 15.10 shows the relative and absolute mean self-prediction errors. The MAE when using power measurements of the power analyzer as input to the run-time model is 1.38 watts. When substituting the power analyzer with the single server power model, the MAE increases by 0.42 watts to 1.8 watts. This is significantly less than the 1.11 W prediction error of the single server power prediction model measured in the separate model evaluation (see Table 15.9). Generally, we can conclude that using both models together results in acceptably small errors warranting their use in case of the unavailability of hardware power analyzers.

15.6 Predicting Power for Containers

Modern distributed applications assembled from components or (micro-)services that can be freely deployed at run-time are usually executed in virtualized environments. More recently, the use of containers instead of full virtualization has also gained popularity. We investigate if our power prediction models can be used in a containerized environment and evaluate the impact this has on their accuracy.

We deploy two virtual SUT containers on the physical SUT 1. Both virtual SUT containers contain a full Debian stack on which the application components can be deployed. We can deploy all of the 15 deployment configurations from the previous experiments on the two virtual SUTs. Doing so introduces a discrepancy between the single server power model and the workload deployment power prediction model as the models consider different systems than the systems for which they predict power. The single server power model predicts power for *physical* servers, whereas the run-time power model predicts power for *hosts*, which in this case are the *virtual* systems. However, the single server

Table 15.11: Prediction error when running in containers.

# Phys. SUTs	Containers	MAE	MAPE
2	No	1.8 W	2.01%
1	Yes	1.25 W	2.02%

model is workload agnostic, requiring only the set of performance counters per physical server, which means that the two models can still be used together.

We instrument one of the containers using the performance counter listener (alternatively, the host operating system could be instrumented instead) and compare the mean self-prediction error when predicting based on the full training set, as used in Section 15.5. Table 15.11 shows that the mean percentage error remains very similar when using containers vs. running on a native system. From this, we conclude that our approach can be readily applied in containerized environments.

15.7 Summary

This chapter evaluates our online power prediction mechanism, introduced in Chapter 9 and answers **EQ B.3** ("Can we accurately predict the power consumption of a potential software component placement based on run-time data?"). We evaluate both the online deployment power prediction method and the single server power prediction approach. We also evaluate the end-to-end scenario that uses both approaches together. The results show that our method can accurately predict the power consumption of previously unobserved deployments with a mean absolute percentage error of 2.2%.

Chapter 16

Conclusions

This chapter concludes the thesis and provides a summary of its contributions. We discuss the benefits of our work and give an overview of potential future work.

16.1 Summary

This thesis addresses two main goals: **Goal A** (create a comprehensive power and energy efficiency measurement and rating methodology for servers) is addressed by presenting a measurement and rating methodology for energy efficiency of servers and an energy efficiency metric to be applied to the results of this methodology. We also design workloads and load intensity and distribution models and mechanisms that can be used for energy efficiency testing. Based on this, we address **Goal B** (provide methods for using the results of the measurement methodology for data center provisioners and/or operators) by presenting power prediction mechanisms and models that utilize our measurement methodology and its results for power prediction. Specifically, the major contributions presented in this thesis are:

Contribution 1: Methodology for Server Efficiency Rating

This thesis presents a methodology for measuring and rating the energy efficiency of servers. The methodology describes experiment hardware and software setup, workload placement, workload measurement phases and intervals, measurement, and metric calculation. It is designed to be used by regulators and decision makers for rating the energy efficiency of servers. It measures a collection of grouped mini-workloads (worklets), covering a range of potential application scenarios in order to provide sufficient insight into the tested system's behavior in many different contexts of use.

We present a set of worklets and workload groupings that are implemented as an implementation of our methodology in the SPEC SERT 2.0. We use this implementation to evaluate our methodology regarding the properties of relevance, reproducibility, and fairness. In this evaluation, we show that the SERT's worklet selection is relevant and enables a thorough system analysis. In addition, we investigate reproducibility by analyzing inter and intra-run variations for our methodology and, finally, we evaluate our metric and its ability to ensure fairness using mathematical proof, correlations with system properties and its relationship with third-party workloads.

Contribution 2: Advanced Load Profiles for Energy Efficiency Measurement

We create models and extraction mechanisms for load profiles that vary over time and load distribution mechanisms and policies. The Descartes Load Intensity Model (DLIM) is designed to capture load profiles using a structured combination of piecewise mathematical functions. This way, models can be designed, extracted, and/or modified for tests using specialized workloads. We introduce three methods enabling the automated extraction of DLIM instances from existing arrival rate traces. The load distribution mechanisms place workloads on computing resources in a hierarchical manner (i.e., first servers, then CPU sockets, then CPU cores). We modify our measurement and rating methodology to allow for this kind of placement. For each of those levels, we select one of three basic strategies, creating a hierarchical strategy composition. The three basic strategies are: load balancing, load consolidation, and a new energy-efficient consolidation strategy.

The results of our evaluation show that the proposed load intensity model extraction methods are capable of extracting DLIM instances with good accuracy from nine different real-world load intensity traces. Our load intensity models can be extracted in less than 0.2 seconds and our resulting models feature a median modeling error of 12.7% on average. We also demonstrate the use of our load distribution method for evaluation. We evaluate the hierarchical load distribution policies and show that our new load distribution strategy can save up to 10.7% of power consumption on a single server node.

Contribution 3: Advanced Workloads for Energy Efficiency Measurement

We create additional workloads and workload-creation methods for specialized use-cases that are often relevant when discussing energy efficiency and power consumption. Firstly, we introduce PET, a framework for creation of synthetic workloads that emulate the power-consumption-relevant profile of other applications. PET is intended to be used to emulate workloads for which the setup of a power benchmarking infrastructure is either highly complex or infeasible.

Specifically, we show PET's use in the context of benchmarking virtual network functions, which usually require a complex network setup. In addition, we introduce TeaStore, a micro-service based, distributed test and reference application intended to serve as a benchmarking framework for researchers evaluating their work. TeaStore is designed to offer the degrees of freedom and performance characteristics required by software (power-)management, prediction, and analysis research.

We show that we are capable of emulating the power consumption behavior of realistic workloads with a mean deviation down to 0.19 W (1%). We also demonstrate that PET can emulate the power consumption of workloads with an average deviation of less than 10% even if the original workload used additional hardware, such as network interface cards. We demonstrate the use of TeaStore by extracting performance models, using it in an auto-scaling environment, and showing that it can be placed with different effects on power consumption of the system under consideration.

Contribution 4: Interpolating Power Consumption

We present a method for automated selection of interpolation strategies for performance and power characterization. We also introduce a new configuration method for piece-wise polynomial interpolation and two generic automated configuration and method selection approaches that improve prediction accuracy for system power consumption for a given system utilization.

We show that, in comparison to regression, our automated interpolation method configuration and selection approach improves modeling accuracy by 43.6% if additional reference data is available and by 31.4% if it is not.

Contribution 5: Offline Prediction of Power Consumption

We present two methods for offline prediction of power consumption of servers. Both methods use results provided by the SPEC SERT implementation of our power rating methodology to predict the power consumption of otherwise unavailable servers. Firstly, we introduce an approach for explicit modeling of the impact a virtualized environment has on power consumption. The approach predicts the SERT results when run within a virtualized system. Secondly, we describe a method to predict the power consumption of a software application. This approach predicts the power consumption of a third-party application for multiple load levels.

Our methods are able to predict power consumption reliably for multiple hypervisor configurations and for the target application workloads. Running on hypervisors, the power consumption of SERT's CPU and storage-bound workloads can be predicted with good accuracy, featuring a relative prediction error of less than 15% on all configurations. CPU workloads feature the greatest accuracy, with some workloads being predicted with errors of less than 1%. Application workload power prediction features a mean average absolute error of 9.5%.

Contribution 6: Online Prediction of Power Consumption

Finally, we propose an end-to-end modeling approach for predicting the power consumption of component placements at run-time. The approach uses run-time monitoring data to train a deployment and power model that allows the prediction of power consumption for different deployment options and load levels in heterogeneous environments. The model can also be used to predict the power consumption at load levels that were not yet observed on the running system. Optionally, we allow replacement of run-time power monitors with a pre-trained performance-counter based single-server power prediction model.

We show that we can predict the power consumption of two different distributed web applications with a mean absolute percentage error of 2.2%. In addition, we can predict the power consumption of a system at a previously unobserved load level and component distribution with an error of 1.2%. We also show that accurately predicting the power consumption of potential deployments and correctly choosing the least power consuming deployment can result in power savings of up to 29.4%.

16.2 Benefits

The work in this thesis benefits multiple groups of people. Among others, data center operators, regulatory govnerment agencies, and hardware developers benefit from the contributions introduced in this thesis. Specifically, we see the following major benefits of our work:

- Our power rating methodology is of benefit for regulators, industry practitioners and researchers. Regulators are the primary target group of the methodology and can use it as-is or with modifications targeted at their specific domain. They can use it to apply energy efficiency labels on servers with the goal of reducing the overall power consumption and carbon footprint. An example of this can be found in the U.S. EPA adopting the SPEC SERT implementation of this methodology for their ENERGY STAR program. Industry practitioners and researchers can use the methodology for the great amount of information that it provides on the systems under test. This information can be used for the development of more efficient devices, workloads, and management mechanisms. Many of the prediction methods in this thesis are an example of such an application.

- Our load intensity models, load distribution mechanism, and workloads allow researchers and system designers to perform more specialized tests. These methods are of special interest in the context of distributed and cloud computing, where varying load intensities and distributions are the order of the day. The TeaStore workload allows exploitation of many of the features that such environments offer. Our PET workload creation mechanism, on the other hand, can be used to reduce the effort needed for repeated setup. In general, our methods and workloads allow for more structured testing, enabling the development of more cost and energy efficient systems and management approaches.

- The offline power prediction methods, which employ our interpolation method, can be used by anyone considering to buy a server in order to make more informed decisions. Servers can be selected based on their power consumption for the target software environment (hypervisor), which is especially useful for cloud environments, where the data center operator knows this information, but does not know the application to be run on top of the hypervisor. In contrast, someone purchasing a server with an application in mind can purchase a server with low power consumption for this specific application, reducing operating costs and environmental impact.

- The online power prediction approach in this thesis can be used in multiple contexts: It can be used at run-time to find the least power consuming deployment option for the current throughput of a distributed application, but it could also be used for energy efficiency prediction when applied in conjunction with run-time performance prediction models, such as DML (Huber et al., 2017). In general, the models and the prediction approach can be leveraged to improve run-time energy management by providing more accurate predictions of the effect of management actions when it comes to re-deploying application components.

16.3 Future Work

The contributions in this thesis can be used as a basis for future work. We see several potential avenues to follow and challenges to address in the future:

Benchmarking of Further Server Hardware Components

Specialized co-processors are gaining popularity in server and cloud environments. Many cloud providers already offer services that provide general purpose GPU (GPGPU) accelerators (Leng et al., 2013). In addition, specialized devices are being introduced for blockchain computations (Magaki et al., 2016) and deep learning (Wang et al., 2016). These device types use different, sometimes vendor-specific, programming languages making comparison of performance and power consumption difficult. Therefore, we see a need for research to address the question of how to design workloads to benchmark such devices and servers using them. Such workloads should enable comparability between equivalent vendor solutions, yet still utilize the different specialized hardware features that such devices support.

Combining Power and Performance Models

In this thesis, we identified a need for power prediction methods and created such methods for offline and online contexts. In contrast, many prediction models already exist for software and server performance (e.g., Huber et al., 2017, Becker et al., 2009 and many more). Considering that energy efficiency is the ratio of performance and power, these models may be combined with our power prediction models for efficiency prediction.

Combining performance and power models is challenging as these models are not based on the same model inputs. For example, our power models use the power per load level data points, provided by the SPEC SERT implementation of our methodology for the offline scenario or a deployment model with runtime power measurements for the online scenario. Models with a focus on performance do not require power measurements and may use deployment models with different model elements designed for their specific purpose. Bridging these differences will require significant future work.

Power Management based on Prediction Models

Our power prediction models, especially our online power prediction model, can be used to predict the effect of power-related management decisions. They can be used by power management, or resource management in general, to make power aware decisions. We see the integration of our power models into resource management as future work. This integration is challenging in so far as it has to consider the trade-offs between many potential aspects, including, but not limited to service level agreements, performance or efficiency optimization, reliability, etc.

In addition, resource management using our power prediction must consider the many actions that could be performed in modern management infrastructures. With containerization further on the rise, these include resource nesting, additional resource control on the operating system level, and new monitoring options.

Static Power Models

In this thesis, we present offline power models that predict the power consumption of hypervisors or software stacks on unavailable machines. Both of these models require the target workload to be executed on an available reference machine. In contrast, we introduce a performance counter based power model as part of our online prediction model. Similarly, we also use performance counters to recreate power profiles. Based on this, we see a potential for static performance-counter-based power models that would not need a reference system for calibration. Such models would probably come with a loss of accuracy, but increased ease of use and might be useful in several contexts. For example, a static performance-counter-based power model could be used in development for quick estimation of a test case's power consumption, allowing for power-aware code optimization.

Energy Efficiency of Virtualized Network Functions

This thesis focuses on the energy efficiency of common commodity servers, which are generally used as regular compute servers. However, such commodity devices are now also being applied in the networking domain for hosting of virtualized network functions (VNFs) (Botero et al., 2012). This introduces a new workload type for servers to be measured and managed with regard to power consumption and energy efficiency. Benchmarking setups and measurement methodologies for such use-cases are a challenging future topic. We addressed this partially in our performance-counter based power emulation approach. However, measuring the power consumption and efficiency of non-emulated virtualized network functions remains challenging. A measurement methodology for such use-cases must define the hardware setup, which may be complex and which may differ depending on the specific VNF. It must also account for network latency and packet loss. This is especially challenging when trying to define load levels and when validating the functionality of the VNF under test. In addition, we expect a number of yet unforeseen challenges to arise when adapting our power rating methodology for this domain.

List of Figures

List of Tables

Acronyms

API Application Programming Interface. 189

APM Application Performance Monitoring. 89

BFAST Breaks for Additive Season and Trends. 15, 60, 63, 65, 158–161, 163

DLIM Descartes Load Intensity Model. 58–61, 63–65, 70, 75, 157–164, 180, 241, 254, 261, 262

DML Descartes Modeling Language. 200, 257

DVFS Dynamic Voltage and Frequency Scaling. 27, 71, 97

hl-DLIM high-level Descartes Load Intensity Model. 59, 60, 63, 65, 261

HTML Hypertext Markup Language. 90

HTTP Hypertext Transfer Protocol. 30, 31, 90, 118, 159, 160, 198, 199, 205, 231, 240, 263

Intel PCM Intel Performance Counter Monitor. 142, 154, 242, 249

JIT Java Just-In-Time. 49

JSP Java Server Page. 89, 91

JVM Java Virtual Machine. 21, 46, 108

MOF Meta-Object Facility. 59

OS Operating System. 80, 84, 173

OSG Open Systems Group. 47

PCM Palladio Component Model. 31, 32, 199

PET Performance Event Trigger Framework. 7, 77–84, 93, 94, 181, 184–189, 191, 195–197, 214, 254, 255, 257

REST Representational State Transfer. 89

s-DLIM Simple DLIM Extraction Method. 60, 64–66, 68–70, 75, 157, 160–164, 180, 261, 262

SERT Server Efficiency Rating Tool. 6, 8, 10, 16, 43–45, 47, 58, 71, 73, 99, 103, 105, 107–110, 113–116, 118, 119, 121, 132, 143, 150, 153–155, 165, 188, 215, 220, 225, 230, 231, 233, 236, 237, 239, 240, 248, 253–256, 258, 261, 266, 267

SLO Service Level Objective. 207

SPEC Standard Performance Evaluation Corporation. 6, 10, 16, 19–22, 27, 28, 31, 43–45, 47, 58, 103, 105, 107, 108, 113, 121, 150, 154, 220, 230, 231, 253, 255, 256, 258

SPEC RG Standard Performance Evaluation Corporation, Research Group. 7, 75, 207

SQL Structured Query Language. 92

SUT System under Test. 44–47, 49, 50, 53, 55, 74, 79, 80, 108, 109, 115, 118, 165, 178, 188, 198, 210, 211, 231, 239–242, 248–250

TDP Thermal Design Power. 142

TPC Transaction Processing Performance Council. 16, 20, 21, 23, 28, 31, 32

U.S. EPA U.S. Environmental Protection Agency. 1, 2, 6, 44, 107, 108, 132, 256

UML Unified Modeling Language. 35, 199

VM Virtual Machine. 26, 39, 108–110, 205, 206, 227–229, 261, 263

XML Extensible Markup Language. 19, 48, 189

Bibliography

Aderaldo, C. M., N. C. Mendonça, C. Pahl, and P. Jamshidi (2017). "Benchmark requirements for microservices architecture research". In: *Proceedings of the 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering*. IEEE Press, pp. 8–13 (see pages 29, 30).

AMD64 Architecture Programmer's Manual Volume 2: System Programming (2016). Advanced Micro Devices Inc. (see page 24).

Apte, V. and B. Doshi (2014). "PowerPerfCenter: A Power and Performance Prediction Tool for Multi-tier Applications". In: *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*. ICPE '14. Dublin, Ireland: ACM, pp. 281–284 (see page 98).

Arlitt, M. and T. Jin (2000). "A Workload Characterization Study of the 1998 World Cup Web Site". In: *IEEE Network* 14.3, pp. 30–37 (see page 205).

Arnold, J. (2013). "Chauffeur: A framework for measuring Energy Efficiency of Servers". Master Thesis. University of Minnesota (see pages 154, 230, 231).

Babcock, C. (2012). "NY Times data center indictment misses the big picture". In: New York, USA (see page 1).

Bao, K., I. Mauser, S. Kochanneck, H. Xu, and H. Schmeck (2016). "A Microservice Architecture for the Intranet of Things and Energy in Smart Buildings: Research Paper". In: *Proceedings of the 1st International Workshop on Mashups of Things and APIs*. MOTA '16. Trento, Italy: ACM, 3:1–3:6 (see page 29).

Barford, P. and M. Crovella (1998). "Generating representative Web workloads for network and server performance evaluation". In: *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*. SIGMETRICS '98/PERFORMANCE '98. Madison, Wisconsin, USA: ACM, pp. 151–160 (see page 34).

Barroso, L. and U. Holzle (2007). "The Case for Energy-Proportional Computing". In: *Computer* 40.12, pp. 33–37 (see pages 23, 33, 44, 71, 142).

Basmadjian, R. and H. De Meer (2012). "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*, pp. 1–10 (see page 27).

Basmadjian, R., N. Ali, F. Niedermeier, H. de Meer, and G. Giuliani (2011). "A Methodology to Predict the Power Consumption of Servers in Data Centres". In: *Proceedings of the 2nd International Conference on Energy-Efficient Computing and Networking*. e-Energy '11. New York, New York: ACM, pp. 1–10 (see pages 4, 38, 39, 85, 209).

Bastani, K. (2015). *Spring Cloud Example Project*. `https://github.com/kbastani/spring-cloud-microservice-example`. Accessed: 19.10.2017 (see pages 29, 32).

Becker, S., G. Brataas, and S. Lehrig (2017a). *Engineering Scalable, Elastic, and Cost-Efficient Cloud Computing Applications: The CloudScale Method*. 1st. Springer Publishing Company, Incorporated (see pages 9, 70, 75).

Becker, S., H. Koziolek, and R. Reussner (2009). "The Palladio component model for model-driven performance prediction". In: *Journal of Systems and Software* 82, pp. 3–22 (see pages 39, 85, 124, 199, 258).

Becker, S. et al. (2017b). *Q-Impress Consortium*. `www.q-impress.eu/wordpress/wp-content/uploads/2009/05/d21-service_architecture_meta-model.pdf`. Accessed: 2017.03.16 (see page 199).

Beitch, A., B. Liu, T. Yung, R. Griffith, A. Fox, and D. A. Patterson (2010). *Rain: A Workload Generation Toolkit for Cloud Computing Applications*. Tech. rep. UCB/EECS-2010-14. EECS Department, University of California, Berkeley (see pages 34, 59).

Belady, C., A. Rawson, J. Pfleuger, and T. Cader (2008). *Green Grid Data Center Power Efficiency Metrics: PUE and DCIE*. Tech. rep. White Paper 6. The Green Grid (see page 32).

Bellosa, F. (2000). "The Benefits of Event: Driven Energy Accounting in Power-sensitive Systems". In: *Proceedings of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC: New Challenges for the Operating System*. EW 9. Kolding, Denmark: ACM, pp. 37–42 (see pages 27, 36).

Beloglazov, A., J. Abawajy, and R. Buyya (2012). "Energy-aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing". In: *Future Gener. Comput. Syst.* 28.5, pp. 755–768 (see pages 4, 39, 85, 209).

Benz, D., A. Hotho, R. Jäschke, B. Krause, F. Mitzlaff, C. Schmitz, and G. Stumme (2010). "The Social Bookmark and Publication Management System BibSonomy". In: *The VLDB Journal* 19.6, pp. 849–875 (see pages 159, 205).

Bianchi, G., A. Detti, A. Caponi, and N. Blefari Melazzi (2013). "Check Before Storing: What is the Performance Price of Content Integrity Verification in LRU Caching?" In: *SIGCOMM Comput. Commun. Rev.* 43.3, pp. 59–67 (see page 203).

Bircher, W. L. and L. K. John (2012). "Complete System Power Estimation Using Processor Performance Events". In: *IEEE Transactions on Computers* 61.4, pp. 563–577 (see page 36).

Blinchikoff, H. J. and A. I. Zverev (1986). *Filtering in the time and frequency domains*. Krieger Publishing Co., Inc. (see page 70).

Bohra, A. E. H. and V. Chaudhary (2010). "VMeter: Power modelling for virtualized clouds". In: *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pp. 1–8 (see page 37).

Bohrer, P., E. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony (2002). "The Case for Power Management in Web Servers". English. In: *Power Aware Computing*. Springer US, pp. 261–289 (see page 98).

Bolla, R., R. Bruschi, C. Lombardo, and S. Mangialardi (2014). "DROPv2: Energy-Efficiency through Network Function Virtualization". In: *IEEE Network* 28.2, pp. 26–32 (see page 36).

Bondarev, E., P. de With, M. Chaudron, and J. Muskens (2005). "Modelling of input-parameter dependency for performance predictions of component-based embedded systems". In: *31st EUROMICRO Conference on Software Engineering and Advanced Applications*. Vienna, Austria: EUROMICRO, pp. 36–43 (see page 199).

Botero, J. F., X. Hesselbach, M. Duelli, D. Schlosser, A. Fischer, and H. de Meer (2012). "Energy Efficient Virtual Network Embedding". In: *IEEE Communications Letters* 16.5, pp. 756–759 (see page 259).

Brataas, G., N. Herbst, S. Ivansek, and J. Polutnik (2017). "Scalability Analysis of Cloud Software Services". (Workshop Paper). In: *Companion Proceedings of the 14th IEEE International Conference on Autonomic Computing (ICAC 2017), Self Organizing Self Managing Clouds Workshop (SOSeMC 2017)*. Columbus, Ohio: IEEE (see page 31).

Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone (1984). *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks (see pages 119, 127).

Breiman, L. (2001). "Random forests". In: *Machine learning* 45.1, pp. 5–32 (see pages 119, 127).

Brooks, D., V. Tiwari, and M. Martonosi (2000). "Wattch: A Framework for Architectural-level Power Analysis and Optimizations". In: *SIGARCH Comput. Archit. News* 28.2, pp. 83–94 (see pages 4, 38, 114).

Burden, R. and J. Faires (1989). *Numerical Analysis*. PWS-Kent, pp. 126–131 (see page 100).

Casale, G., A. Kalbasi, D. Krishnamurthy, and J. Rolia (2012). "BURN: Enabling Workload Burstiness in Customized Service Benchmarks". In: *IEEE Transactions on Software Engineering* 38.4, pp. 778–793 (see page 34).

Cavazos, J., G. Fursin, F. Agakov, E. Bonilla, M. F. O'Boyle, and O. Temam (2007). "Rapidly Selecting Good Compiler Optimizations using Performance Counters". In: *CGO '07 Proceedings of the International Symposium on Code Generation and Optimization*, pp. 185–197 (see page 36).

Cecchet, E., A. Chanda, S. Elnikety, J. Marguerite, and W. Zwaenepoel (2003). "Performance comparison of middleware architectures for generating dynamic web content". In: *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*. Springer-Verlag New York, Inc., pp. 242–261 (see page 30).

Cecchet, E., J. Marguerite, and W. Zwaenepoel (2002). "Performance and scalability of EJB applications". In: *ACM Sigplan Notices*. Vol. 37. 11. ACM, pp. 246–261 (see page 30).

Cecci, H. and J. E. Pultz (2016). "Five Steps to Maximize Data Center Efficiency and Get Effective Results, Including a Low PUE". In: (see page 1).

Chen, F., J. Grundy, Y. Yang, J.-G. Schneider, and Q. He (2013). "Experimental Analysis of Task-based Energy Consumption in Cloud Computing Systems". In: *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*. ICPE '13. Prague, Czech Republic: ACM, pp. 295–306 (see page 26).

Chen, J., B. Li, Y. Zhang, L. Peng, and J.-K. Peir (2011). "Statistical GPU power analysis using tree-based methods". In: *Green Computing Conference and Workshops (IGCC), 2011 International*, pp. 1–6 (see page 37).

Chen, Y., A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam (2005). "Managing Server Energy and Operational Costs in Hosting Centers". In: *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS '05. Banff, Alberta, Canada: ACM, pp. 303–314 (see pages 35, 71).

Chieu, T. C., A. Mohindra, A. A. Karve, and A. Segal (2009). "Dynamic scaling of web applications in a virtualized cloud computing environment". In: *E-Business Engineering, 2009. ICEBE'09. IEEE International Conference on*. IEEE, pp. 281–286 (see pages 29, 205).

Chisnall, D. (2008). *The definitive guide to the xen hypervisor*. Pearson Education (see page 108).

Choi, J., S. Govindan, B. Urgaonkar, and A. Sivasubramaniam (2008). "Profiling, Prediction, and Capping of Power Consumption in Consolidated Environments". In: *Modeling, Analysis and Simulation of Computers and Telecommunica-*

tion Systems, 2008. MASCOTS 2008. IEEE International Symposium on, pp. 1–10 (see page 37).

CloudScale Consortium (2016). *CloudStore*. `https://github.com/CloudScale-Project/CloudStore`. Accessed: 18.10.2017 (see page 31).

Contreras, G. and M. Martonosi (2005a). "Power prediction for Intel XScale reg; processors using performance monitoring unit events". In: *Low Power Electronics and Design, 2005. ISLPED '05. Proceedings of the 2005 International Symposium on*, pp. 221–226 (see page 27).

Contreras, G. and M. Martonosi (2005b). "Power Prediction for Intel XScale® Processors Using Performance Monitoring Unit Events". In: *Proceedings of the 2005 International Symposium on Low Power Electronics and Design*. ISLPED '05. San Diego, CA, USA: ACM, pp. 221–226 (see page 36).

Dell, Inc. (2011). *The DVD Store Version 2*. `http://en.community.dell.com/techcenter/extras/w/wiki/dvd-store`. Last accessed Jan. 2018 (see pages 31, 85, 153, 230, 240).

Dhiman, G., K. Mihic, and T. Rosing (2010). "A system for online power prediction in virtualized environments using gaussian mixture models". In: *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pp. 807–812 (see pages 37, 119).

Dorronsoro, B., S. Nesmachnow, J. Taheri, A. Y. Zomaya, E.-G. Talbi, and P. Bouvry (2014). "A hierarchical approach for energy-efficient scheduling of large workloads in multicore distributed systems ". In: *Sustainable Computing: Informatics and Systems* 4.4. Special Issue on Energy Aware Resource Management and Scheduling (EARMS), pp. 252–261 (see page 35).

Economou, D., S. Rivoire, and C. Kozyrakis (2006a). "Full-system power analysis and modeling for server environments". In: *In Workshop on Modeling Benchmarking and Simulation (MOBS* (see page 37).

Economou, D., S. Rivoire, C. Kozyrakis, and P. Ranganathan (2006b). "Full-system power analysis and modeling for server environments". In: *International Symposium on Computer Architecture*. IEEE (see page 29).

EPA (2013). *ENERGY STAR Program Requirements for Computer Servers*. `https://www.energystar.gov/ia/partners/prod_development/revisions/downloads/computer_servers/Program_Requirements_V2.0.pdf` (see pages 2, 107, 114).

Eyerman, S., L. Eeckhout, T. Karkhanis, and J. E. Smith (2006). "A Performance Counter Architecture for Computing Accurate CPI Components". In: *ASPLOS XII*, pp. 175–184 (see page 36).

Ezhilchelvan, P. and I. Mitrani (2016). "Optimal provision of multiple service types". In: *Modeling, Analysis and Simulation of Computer and Telecommunication*

Systems (MASCOTS), 2016 IEEE 24th International Symposium on. IEEE, pp. 21–29 (see page 29).

Fan, X., W.-D. Weber, and L. A. Barroso (2007). "Power Provisioning for a Warehouse-sized Computer". In: *The 34th ACM International Symposium on Computer Architecture* (see pages 37, 98, 106, 216).

Feitelson, D. (2002). "Workload Modeling for Performance Evaluation". English. In: *Performance Evaluation of Complex Systems: Techniques and Tools*. Ed. by M. Calzarossa and S. Tucci. Vol. 2459. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 114–141 (see page 34).

Floater, M. S. (2003). "Mean value coordinates". In: *Computer Aided Geometric Design* 20.1, pp. 19–27 (see page 154).

Förderer, K., M. Ahrens, K. Bao, I. Mauser, and H. Schmeck (2018). "Towards the Modeling of Flexibility Using Artificial Neural Networks in Energy Management and Smart Grids". In: *Proceedings of the Ninth International Conference on Future Energy Systems (e-Energy '18)*. Ed. by ACM. Ninth International Conference on Future Energy Systems (e-Energy '18), ACM. New York, NY, USA: ACM, pp. 85–90 (see page 39).

Friedman, J. H. (2001). "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics*, pp. 1189–1232 (see pages 119, 128).

– (1991). "Multivariate adaptive regression splines". In: *The annals of statistics*, pp. 1–67 (see page 34).

García-Castro, R. and A. Gómez-Pérez (2006). "Benchmark Suites for Improving the RDF(S) Importers and Exporters of Ontology Development Tools". In: *The Semantic Web: Research and Applications*. Ed. by Y. Sure and J. Domingue. Vol. 4011. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 155–169 (see pages 18, 20).

Garetto, M., E. Leonardi, and V. Martina (2016). "A unified approach to the performance analysis of caching systems". In: *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* 1.3, p. 12 (see page 203).

George, B., G. Yeap, M. Wloka, S. Tyler, and D. Gossain (1994). "Power analysis for semi-custom design". In: *Custom Integrated Circuits Conference, 1994., Proceedings of the IEEE 1994*, pp. 249–252 (see page 27).

Gérard, S. and B. Selic (2008). "The UML–MARTE Standardized Profile". In: *IFAC Proceedings Volumes* 41.2, pp. 6909–6913 (see page 199).

Gomaa, M., M. D. Powell, and T. N. Vijaykumar (2004). "Heat-and-run: Leveraging SMT and CMP to Manage Power Density Through the Operating System". In: *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS XI. Boston, MA, USA: ACM, pp. 260–270 (see pages 27, 35).

Gong, Z., X. Gu, and J. Wilkes (2010). "Press: Predictive elastic resource scaling for cloud systems". In: *Network and Service Management (CNSM), 2010 International Conference on*. IEEE, pp. 9–16 (see pages 29, 31).

Groenda, H. and C. Stier (2015). "Improving IaaS Cloud Analyses by Black-Box Resource Demand Modeling". In: *Softwaretechnik-Trends* 35.3. in print (see pages 9, 70, 75).

Groenda, H. et al. (2017). *CACTOS toolkit version 2: accompanying document for prototype deliverable D5. 2.2* (see page 199).

Gurumurthi, S., A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, and M. Kandemir (2002). "Using complete machine simulation for software power estimation: the SoftWatt approach". In: *Proceedings Eighth International Symposium on High Performance Computer Architecture*, pp. 141–150 (see pages 37, 38).

Gustafson, J. and Q. Snell (1995). "HINT: A new way to measure computer performance". In: *System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on*. Vol. 2, 392–401 vol.2 (see page 18).

Halili, E. H. (2008). *Apache JMeter: A Practical Beginner's Guide to Automated Testing and performance measurement for your websites*. Packt Publishing Ltd (see page 59).

Happe, J., H. Koziolek, and R. Reussner (2011). "Facilitating performance predictions using software components". In: *IEEE Software* 28.3, pp. 27–33 (see pages 31, 85).

Happe, L., B. Buhnova, and R. Reussner (2014). "Stateful Component-based Performance Models". In: *Softw. Syst. Model.* 13.4, pp. 1319–1343 (see page 203).

Henderson, D., S. H. Jacobson, and A. W. Johnson (2003). *The Theory and Practice of Simulated Annealing*. Ed. by F. Glover and G. A. Kochenberger. Boston, MA: Springer US, pp. 287–319 (see page 81).

Henning, J. L. (2000). "SPEC CPU2000: Measuring CPU Performance in the New Millennium". In: *Computer* 33.7, pp. 28–35 (see pages 18, 21, 27, 28).

Heo, S., K. Barr, and K. Asanovic (2003). "Reducing power density through activity migration". In: *Low Power Electronics and Design, 2003. ISLPED '03. Proceedings of the 2003 International Symposium on*, pp. 217–222 (see pages 27, 35).

Herbst, N. (2018). "Methods and Benchmarks for Auto-Scaling Mechanisms in Elastic Cloud Environments". PhD thesis. University of Würzburg, Germany (see pages 60, 61).

Herbst, N. R., S. Kounev, A. Weber, and H. Groenda (2015). "BUNGEE: An Elasticity Benchmark for Self-adaptive IaaS Cloud Environments". In: *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and*

Self-Managing Systems. SEAMS '15. Florence, Italy: IEEE Press, pp. 46–56 (see pages 9, 70, 75).

Herbst, N. and more (2016). "Ready for Rain? A View from SPEC Research on the Future of Cloud Metrics". In: *CoRR* abs/1604.03470 (see page 207).

Hoorn, A. van, M. Rohr, and W. Hasselbring (2008). "Generating Probabilistic and Intensity-Varying Workload for Web-Based Software Systems". In: *Proceedings of the SPEC international workshop on Performance Evaluation: Metrics, Models and Benchmarks*. SIPEW '08. Darmstadt, Germany: Springer-Verlag, pp. 124–143 (see page 34).

Hoorn, A. van, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey, and D. Kieselhorst (2009). *Continuous Monitoring of Software Services: Design and Application of the Kieker Framework*. Forschungsbericht. Kiel University (see pages 89, 93).

Hoorn, A. van, J. Waller, and W. Hasselbring (2012). "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)*. ACM, pp. 247–248 (see pages 89, 93).

Hoschek, J. and D. Lasser (1993). *Fundamentals of Computer Aided Geometric Design*. Natick, MA, USA: A. K. Peters, Ltd. (see page 99).

Hsu, C.-H. and S. W. Poole (2015). "Measuring Server Energy Proportionality". In: *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. ICPE '15. Austin, Texas, USA: ACM, pp. 235–240 (see pages 4, 33).

Huang, C. X., B. Zhang, A.-C. Deng, and B. Swirski (1995). "The Design and Implementation of PowerMill". In: *Proceedings of the 1995 International Symposium on Low Power Design*. ISLPED '95. Dana Point, California, USA: ACM, pp. 105–110 (see page 27).

Huber, N., F. Brosig, S. Spinner, S. Kounev, and M. Bähr (2017). "Model-Based Self-Aware Performance and Resource Management Using the Descartes Modeling Language". In: *IEEE Transactions on Software Engineering (TSE)* 43.5 (see pages 39, 124, 200, 204, 257, 258).

Huppler, K. (2009). "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. Springer Berlin Heidelberg, pp. 18–30 (see pages 18, 19, 23).

Huppler, K. and D. Johnson (2014). "TPC Express - A New Path for TPC Benchmarks". English. In: *Performance Characterization and Benchmarking*. Ed. by R. Nambiar and M. Poess. Vol. 8391. Lecture Notes in Computer Science. Springer International Publishing, pp. 48–60 (see page 17).

IBM (2015). *ACME Air*. `https://github.com/acmeair/acmeair`. Accessed: 19.10.2017 (see pages 29, 32).

Ilyushkin, A., A. Ali-Eldin, N. Herbst, A. Bauer, A. V. Papadopoulos, D. Epema, and A. Iosup (2018). "An Experimental Performance Evaluation of Autoscalers for Complex Workflows". In: *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (ToMPECS)* 3.2, 8:1–8:32 (see page 206).

Ilyushkin, A., A. Ali-Eldin, N. Herbst, A. V. Papadopoulos, B. Ghit, D. Epema, and A. Iosup (2017). "An Experimental Performance Evaluation of Autoscaling Policies for Complex Workflows". In: *Proceedings of the 8th ACM/SPEC International Conference on Performance Engineering (ICPE 2017)*. Best Paper Candidate (1/4). l'Aquila, Italy: ACM (see page 85).

Intel® 64 and IA-32 Architectures Software Developer's Manual (2016). Intel Corporation (see page 24).

Isci, C. and M. Martonosi (2003). "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, pp. 93– (see page 36).

Jin, Y., Y. Wen, and Q. Chen (2012). "Energy Efficiency and Server Virtualization in Data Centers: An Empirical Investigation". In: *2012 IEEE Conference on Computer Communications Workshops*, pp. 133–138 (see page 26).

Jung, G., M. Hiltunen, K. Joshi, R. Schlichting, and C. Pu (2010). "Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures". In: *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, pp. 62–73 (see page 39).

Kadayif, I., T.Chinoda, M. Kandemir, N. Vijaykrishnan, M. Irwin, and A. Sivasubramaniam (2001). "vEC: Virtual Energy Counters". In: *PASTE'01*, pp. 28–31 (see page 36).

Kahng, A. B., B. Li, L. S. Peh, and K. Samadi (2009). "ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration". In: *2009 Design, Automation Test in Europe Conference Exhibition*, pp. 423–428 (see pages 4, 38, 114).

Kansal, A., F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya (2010). "Virtual Machine Power Metering and Provisioning". In: *Proceedings of the 1st ACM Symposium on Cloud Computing*. SoCC '10. Indianapolis, Indiana, USA: ACM, pp. 39–50 (see page 37).

Koziolek, A., D. Ardagna, and R. Mirandola (2013). "Hybrid Multi-Attribute QoS Optimization in Component Based Software Systems". In: *Journal of Systems and Software* 86.10, pp. 2542–2558 (see page 204).

Krogmann, K., M. Kuperberg, and R. Reussner (2010). “Using genetic search for reverse engineering of parametric behavior models for performance prediction”. In: *IEEE Transactions on Software Engineering* 36.6, pp. 865–877 (see page 205).

Kuhn, M., S. Weston, C. Keefer, and N. Coulter (2012). *Cubist Models For Regression*. http://cran.r-project.org/web/packages/Cubist/vignettes/cubist.pdf, Last accessed: Oct 2014 (see page 34).

Kusic, D., J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang (2008). “Power and Performance Management of Virtualized Computing Environments Via Lookahead Control”. In: *Autonomic Computing, 2008. ICAC ’08. International Conference on*, pp. 3–12 (see page 35).

Lange, K.-D. (2009). “Identifying Shades of Green: The SPECpower Benchmarks”. In: *Computer* 42.3, pp. 95–97 (see pages 3, 28, 32, 33, 44, 45, 50, 54).

Lange, K.-D., J. A. Arnold, H. Block, N. Totura, J. Beckett, and M. G. Tricker (2013). “Further Implementation Aspects of the Server Efficiency Rating Tool (SERT)”. In: *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*. ICPE ’13. Prague, Czech Republic: ACM, pp. 349–360 (see page 215).

Lange, K.-D. and M. G. Tricker (2011). “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, pp. 145–150 (see pages 1, 78).

Lange, K.-D., M. G. Tricker, J. A. Arnold, H. Block, and C. Koopmann (2012). “The Implementation of the Server Efficiency Rating Tool”. In: *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. ICPE ’12. Boston, Massachusetts, USA: ACM, pp. 133–144 (see page 215).

Lee, B. C. and D. M. Brooks (2006). “Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction”. In: *SIGPLAN Not.* 41.11, pp. 185–194 (see pages 37, 98).

Lee, I. and R. K. Iyer (1995). “Software dependability in the Tandem GUARDIAN system”. In: *IEEE Transactions on Software Engineering* 21.5, pp. 455–467 (see page 85).

Lefèvre, L. and A.-C. Orgerie (2010). “Designing and Evaluating an Energy Efficient Cloud”. In: *J. Supercomput.* 51.3, pp. 352–373 (see page 26).

Lehrig, S., R. Sanders, G. Brataas, M. Cecowski, S. Ivanšek, and J. Polutnik (2018). “CloudStore—towards scalability, elasticity, and efficiency benchmarking and analysis in Cloud computing”. In: *Future Generation Computer Systems* 78, pp. 115–126 (see page 31).

Lemire, D. and A. Maclachlan (2005). "Slope one predictors for online rating-based collaborative filtering". In: *Proceedings of the 2005 SIAM International Conference on Data Mining*. SIAM, pp. 471–475 (see page 92).

Leng, J., T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi (2013). "GPUWattch: Enabling Energy Optimizations in GPGPUs". In: *SIGARCH Comput. Archit. News* 41.3, pp. 487–498 (see page 258).

Lewis, A., S. Ghosh, and N.-F. Tzeng (2008). "Run-time Energy Consumption Estimation Based on Workload in Server Systems". In: *Proceedings of the 2008 Conference on Power Aware Computing and Systems*. HotPower'08. San Diego, California: USENIX Association (see pages 36, 37, 98).

Li, H. (2010). "Realistic Workload Modeling and Its Performance Impacts in Large-Scale eScience Grids". In: *Parallel and Distributed Systems, IEEE Transactions on* 21.4, pp. 480–493 (see page 34).

Littlewood, B. and L. Strigini (1995). "Validation of ultra-high dependability for software-based systems". In: *Predictably Dependable Computing Systems*. Springer, pp. 473–493 (see page 85).

Lochmann, A., F. Bruckner, and O. Spinczyk (2017). "Reproducible Load Tests for Android Systems with Trace-based Benchmarks". In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. ICPE '17 Companion. L'Aquila, Italy: ACM, pp. 73–76 (see page 28).

Magaki, I., M. Khazraee, L. V. Gutierrez, and M. B. Taylor (2016). "ASIC Clouds: Specializing the Datacenter". In: *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 178–190 (see page 258).

Mauser, I., M. Dorscheid, and H. Schmeck (2014). "Run-Time Parameter Selection and Tuning for Energy Optimization Algorithms". In: *Parallel Problem Solving from Nature – PPSN XIII*. Ed. by T. Bartz-Beielstein, J. Branke, B. Filipič, and J. Smith. Cham: Springer International Publishing, pp. 80–89 (see page 39).

Mauser, I., J. Müller, and H. Schmeck (2017). "Utilizing Flexibility of Hybrid Appliances in Local Multi-modal Energy Management". In: *Proceedings of the 9th International Conference EEDAL'2017 - Energy Efficiency in Domestic Appliances and Lighting*. JRC Conference and Workshop Report. Publications Office of the European Union, pp. 1282–1297 (see page 39).

Menascé, D. A., V. A. F. Almeida, R. Riedi, F. Ribeiro, R. Fonseca, and W. Meira Jr. (2003). "A hierarchical and multiscale approach to analyze E-business workloads". In: *Perform. Eval.* 54.1, pp. 33–57 (see page 34).

Metri, G., S. Srinivasaraghavan, W. Shi, and M. Brockmeyer (2012). "Experimental Analysis of Application Specific Energy Efficiency of Data Centers with Heterogeneous Servers". In: *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pp. 786–793 (see pages 3, 32, 54).

Nagasaka, H., N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka (2010). "Statistical power modeling of GPU kernels using performance counters". In: *Green Computing Conference, 2010 International*, pp. 115–122 (see page 98).

Nathuji, R. and K. Schwan (2007). "VirtualPower: Coordinated Power Management in Virtualized Enterprise Systems". In: *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*. SOSP '07. Stevenson, Washington, USA: ACM, pp. 265–278 (see page 26).

.NET Foundation (2017). *MusicStore (test application)*. `https://github.com/aspnet/MusicStore`. Accessed: 18.10.2017 (see pages 29, 32).

Niu, D., Y. Wang, and D. D. Wu (2010). "Power load forecasting using support vector machine and ant colony optimization". In: *Expert Systems with Applications* 37.3, pp. 2531–2539 (see page 37).

Noorshams, Q., D. Bruhn, S. Kounev, and R. Reussner (2013). "Predictive Performance Modeling of Virtualized Storage Systems using Optimized Statistical Regression Techniques". In: *Proceedings of the ACM/SPEC International Conference on Performance Engineering (ICPE 2013)*. ICPE'13. Prague, Czech Republic: ACM, pp. 283–294 (see page 121).

Okanović, D., A. van Hoorn, C. Heger, A. Wert, and S. Siegl (2016). "Towards Performance Tooling Interoperability: An Open Format for Representing Execution Traces". In: *Proceedings of the 13th European Workshop on Performance Engineering (EPEW '16)*. Springer (see page 93).

Oracle and S. Microsystems (2005). *JPetStore 2.0*. `http://www.oracle.com/technetwork/java/index-136650.html`. Accessed: 17.10.2017 (see page 32).

Papadopoulos, A. V., A. Ali-Eldin, K.-E. en, J. Tordsson, and E. Elmroth (2016). "PEAS: A Performance Evaluation Framework for Auto-Scaling Strategies in Cloud Applications". In: *ACM Trans. Model. Perform. Eval. Comput. Syst.* 1.4, 15:1–15:31 (see page 204).

Pinheiro, E., R. Bianchini, E. V. Carrera, and T. Heath (2001). *Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems* (see pages 35, 71).

Podzimek, A., L. Bulej, L. Y. Chen, W. Binder, and P. Tůma (2015). "Analyzing the Impact of CPU Pinning and Partial CPU Loads on Performance and Energy Efficiency". In: *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. CCGRID 2015. Accepted for publication. Shenzhen, Guangdong, China (see page 27).

Poess, M., R. O. Nambiar, K. Vaid, J. M. Stephens Jr, K. Huppler, and E. Haines (2010). "Energy benchmarks: a detailed analysis". In: *Proceedings of the 1st*

International Conference on Energy-Efficient Computing and Networking. ACM, pp. 131–140 (see pages 3, 28, 32, 44, 54).

Quan, D. et al. (2012). “Energy Efficient Resource Allocation Strategy for Cloud Data Centres”. English. In: *Computer and Information Sciences II*. Ed. by E. Gelenbe, R. Lent, and G. Sakellari. Springer London, pp. 133–141 (see page 35).

Quinlan, J. R. et al. (1992). “Learning with continuous classes”. In: *Proceedings of the 5th Australian joint Conference on Artificial Intelligence*. Vol. 92. Singapore, pp. 343–348 (see page 34).

Raghavendra, R., P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu (2008). “No "Power" Struggles: Coordinated Multi-level Power Management for the Data Center”. In: *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS XIII. Seattle, WA, USA: ACM, pp. 48–59 (see pages 35, 71).

Rausch, A., R. Reussner, R. Mirandola, and F. Plasil (2008). “The Common Component Modeling Example”. In: *Lecture notes in computer science* 5153 (see page 31).

Reyes-Lecuona, A., E. González-Parada, E. Casilari, J. Casasola, and A. Diaz-Estrella (1999). “A page-oriented WWW traffic model for wireless system simulations”. In: *Proceedings ITC*. Vol. 16, pp. 1271–1280 (see page 34).

Rivoire, S., M. Shah, P. Ranganathan, C. Kozyrakis, and J. Meza (2007a). “Models and Metrics to Enable Energy-Efficiency Optimizations”. In: *Computer* 40.12, pp. 39–48 (see page 32).

Rivoire, S., P. Ranganathan, and C. Kozyrakis (2008). “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, pp. 3–3 (see pages 4, 37, 39, 124, 132, 216).

Rivoire, S., M. A. Shah, P. Ranganathan, and C. Kozyrakis (2007b). “JouleSort: A Balanced Energy-efficiency Benchmark”. In: *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’07. Beijing, China: ACM, pp. 365–376 (see pages 3, 28, 32, 44, 54).

Roy, S., T. Begin, and P. Goncalves (2013). “A complete framework for modelling and generating workload volatility of a VoD system”. In: *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*, pp. 1168–1174 (see page 34).

RUBiS User’s Manual (2008) (see pages 30, 31, 85, 240).

Russell, J. and M. Jacome (1998). “Software power estimation and optimization for high performance, 32-bit embedded processors”. In: *Computer Design: VLSI in Computers and Processors, 1998. ICCD ’98. Proceedings*. Pp. 328–333 (see page 27).

Rygielski, P., M. Seliuchenko, and S. Kounev (2016). “Modeling and Prediction of Software-Defined Networks Performance using Queueing Petri Nets”. In: *Proceedings of the Ninth International Conference on Simulation Tools and Techniques (SIMUTools 2016)*. Prague, Czech Republic, pp. 66–75 (see page 204).

Schall, D., V. Hoefner, and M. Kern (2012). “Towards an Enhanced Benchmark Advocating Energy-Efficient Systems”. In: *Topics in Performance Evaluation, Measurement and Characterization*. Ed. by R. Nambiar and M. Poess. Vol. 7144. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 31–45 (see pages 4, 33).

Schönherr, J. H., J. Richling, M. Werner, and G. Mühl (2010). “Event-driven Processor Power Management”. In: *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*. e-Energy ’10. Passau, Germany: ACM, pp. 61–70 (see page 27).

Schroeder, B., A. Wierman, and M. Harchol-Balter (2006). “Open versus closed: a cautionary tale”. In: *Proceedings of the 3rd conference on Networked Systems Design & Implementation - Volume 3*. NSDI’06. San Jose, CA: USENIX Association, pp. 18–18 (see pages 14, 198).

Shen, X., F. Mohd-Zaid, and R. Francis (2012). “Runge Phenomenon: A virtual artifact in image processing”. In: *Proceedings of the 2012 International Conference on Image Processing, Computer Vision, and Pattern Recognition*. Las Vegas, USA (see page 100).

Shepard, D. (1968). “A Two-dimensional Interpolation Function for Irregularly-spaced Data”. In: *Proceedings of the 1968 23rd ACM National Conference*. ACM ’68. New York, NY, USA: ACM, pp. 517–524 (see page 99).

Sim, S. E., S. Easterbrook, and R. C. Holt (2003). “Using Benchmarking to Advance Research: A Challenge to Software Engineering”. In: *Proceedings of the 25th International Conference on Software Engineering*. ICSE ’03. Portland, Oregon: IEEE Computer Society, pp. 74–83 (see page 18).

Singh, K., M. Bhadauria, and S. A. McKee (2009). “Real Time Power Estimation and Thread Scheduling via Performance Counters”. In: *ACM SIGARCH Computer Architecture News* 37, pp. 46–55 (see page 36).

Sitaraman, M., G. Kulczycki, J. Krone, W. F. Ogden, and A. L. N. Reddy (2001). “Performance Specification of Software Components”. In: *SIGSOFT Softw. Eng. Notes* 26.3, pp. 3–10 (see page 199).

Skadron, K., M. Martonosi, D. I. August, M. D. Hill, D. J. Lilja, and V. S. Pai (2003). “Challenges in Computer Architecture Evaluation”. In: *Computer* 36.8, pp. 30–36 (see pages 18, 19).

Software, P. (2016). *Spring PetClinic*. `https://github.com/spring-projects/spring-petclinic`. Accessed: 19.10.2017 (see page 32).

Spinner, S., G. Casale, F. Brosig, and S. Kounev (2015). “Evaluating Approaches to Resource Demand Estimation”. In: *Performance Evaluation* 92, pp. 51–71 (see page 201).

Spinner, S., G. Casale, X. Zhu, and S. Kounev (2014). “LibReDE: A Library for Resource Demand Estimation”. (Demo Paper). In: *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*. Dublin, Ireland: ACM Press, pp. 227–228 (see page 93).

Srikantaiah, S., A. Kansal, and F. Zhao (2008). “Energy Aware Consolidation for Cloud Computing”. In: *Proceedings of the 2008 Conference on Power Aware Computing and Systems*. HotPower’08. San Diego, California: USENIX Association, pp. 10–10 (see page 26).

Standard Performance Evaluation Corporation (2018). *SPEC fair use rule*. `http://www.spec.org/fairuse.html`. Gainsville, VA, USA (see pages 10, 22).

Standard Performance Evaluation Corporation (SPEC) (2010). *SPEC jEnterprise 2010 Design Document*. `https://www.spec.org/jEnterprise2010/docs/DesignDocumentation.html`. Accessed: 16.10.2017 (see pages 29, 31).

Stefani, F., A. Moschitta, D. Macii, and D. Petri (2003). “FFT benchmarking for digital signal processing technologies”. In: *17th IMEKO World Congress* (see page 18).

Stier, C., A. Koziolek, H. Groenda, and R. Reussner (2015). “Model-Based Energy Efficiency Analysis of Software Architectures”. In: *Proceedings of the 9th European Conference on Software Architecture (ECSA ’15)*. Lecture Notes in Computer Science. Dubrovnik/Cavtat, Croatia: Springer (see pages 4, 38, 39).

Tian, Y., C. Lin, and M. Yao (2012). “Modeling and analyzing power management policies in server farms using Stochastic Petri Nets”. In: *Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy), 2012 Third International Conference on*, pp. 1–9 (see page 39).

Tu, J., L. Lu, M. Chen, and R. Sitaraman (2013). “Dynamic provisioning in next-generation data centers with on-site power production”. In: (see page 98).

Urgaonkar, R., U. Kozat, K. Igarashi, and M. Neely (2010). “Dynamic resource allocation and power management in virtualized data centers”. In: *Network Operations and Management Symposium (NOMS), 2010 IEEE*, pp. 479–486 (see pages 4, 39).

Vaughan, A. (2015). “How viral cat videos are warming the planet”. In: London, United Kingdom (see page 1).

Verbesselt, J., R. Hyndman, G. Newnham, and D. Culvenor (2010). “Detecting trend and seasonal changes in satellite image time series”. In: *Remote Sensing of Environment* 114.1, pp. 106–115 (see pages 15, 61, 63, 65, 158, 160).

Verma, A., P. Ahuja, and A. Neogi (2008a). "pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems". In: *Middleware 2008*. Ed. by V. Issarny and R. Schantz. Vol. 5346. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 243–264 (see pages 35, 38, 71, 98).

– (2008b). "Power-aware Dynamic Placement of HPC Applications". In: *Proceedings of the 22nd Annual International Conference on Supercomputing*. Island of Kos, Greece: ACM (see page 38).

Vieira, M., H. Madeira, K. Sachs, and S. Kounev (2012). "Resilience Benchmarking". In: *Resilience Assessment and Evaluation of Computing Systems*. Ed. by K. Wolter, A. Avritzer, M. Vieira, and A. van Moorsel. XVIII. ISBN: 978-3-642-29031-2. Berlin, Heidelberg: Springer-Verlag (see page 16).

Walter, J., A. D. Marco, S. Spinner, P. Inverardi, and S. Kounev (2017a). "Online Learning of Run-time Models for Performance and Resource Management in Data Centers". In: *Self-Aware Computing Systems*. Ed. by S. Kounev, J. O. Kephart, A. Milenkoski, and X. Zhu. Berlin Heidelberg, Germany: Springer Verlag (see page 205).

Walter, J., C. Stier, H. Koziolek, and S. Kounev (2017b). "An Expandable Extraction Framework for Architectural Performance Models". In: *Proceedings of the 3rd International Workshop on Quality-Aware DevOps (QUDOS'17)*. l'Aquila, Italy: ACM (see page 205).

Wang, C., Q. Yu, L. Gong, X. Li, Y. Xie, and X. Zhou (2016). "DLAU: A Scalable Deep Learning Accelerator Unit on FPGA". In: *CoRR* abs/1605.06894. arXiv: `1605.06894` (see page 258).

Weaver, V. M. (2015). "Self-monitoring Overhead of the Linux perf_event Performance Counter Interface". In: *ISPASS 2015*. IEEE, pp. 102–111 (see page 37).

Weaver, V. M., D. Terpstra, and S. Moore (2013). "Non-Determinism and Overcount on Modern Hardware Performance Counter Implementations". In: *Performance Analysis of Systems and Software, 2013. ISPASS 2013. IEEE International Symposium on* (see pages 24, 36).

Weaveworks Inc. (2017). *Sock Shop: A Microservice Demo Application*. `https://github.com/microservices-demo/microservices-demo`. Accessed: 19.10.2017 (see pages 29, 32, 85).

Welch, T. (1984). "A Technique for High-Performance Data Compression". In: *Computer* 17.6, pp. 8–19 (see page 47).

Willhalm, T., R. Dementiev, and P. Fay (2012). *Intel® Performance Counter Monitor - A better way to measure CPU utilization*. https://software.intel.com/en-us/articles/intel-performance-counter-monitor-a-better-way-to-measure-cpu-utilization (see pages 142, 154, 242, 249).

Willnecker, F., A. Brunnert, W. Gottesheim, and H. Krcmar (2015a). “Using dynatrace monitoring data for generating performance models of java ee applications”. In: *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. ACM, pp. 103–104 (see page 205).

Willnecker, F., M. Dlugi, A. Brunnert, S. Spinner, S. Kounev, and H. Krcmar (2015b). “Comparing the Accuracy of Resource Demand Measurement and Estimation Techniques”. In: *Computer Performance Engineering - Proceedings of the 12th European Workshop (EPEW 2015)*. Ed. by M. Beltrán, W. Knottenbelt, and J. Bradley. Vol. 9272. Lecture Notes in Computer Science. Madrid, Spain: Springer, pp. 115–129 (see pages 31, 85).

Zakay, N. and D. G. Feitelson (2013). “Workload resampling for performance evaluation of parallel job schedulers”. In: *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*. ICPE ’13. Prague, Czech Republic: ACM, pp. 149–160 (see page 34).

Zaparanuks, D., M. Jovic, and M. Hauswirth (2009). “Accuracy of performance counter measurements”. In: *ISPASS 2009*, pp. 23–32 (see page 36).