UNIVERSITY OF CALIFORNIA, MERCED

Hybrid Power Models for Data Center Energy Efficiency

A thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Electrical Engineering and Computer Science

by

Nigel Bernard

Committee in charge:

Professor Hyrean Jeon, Chair
Professor Dong Li
Professor Alberto Cerpa

2021

The thesis of Nigel Bernard is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

(Professor Dong Li)

(Professor Alberto Cerpa)

(Professor Hyrean Jeon, Chair)

University of California, Merced

2021

TABLE OF CONTENTS

Signature Page iii

Table of Contents iv

List of Figures vi

List of Tables viii

Vita and Publications ix

Abstract x

Chapter 1 Introduction 1

Chapter 2 Background and Related Work 8
2.1 Background Technologies 8
2.1.1 Classifiers 8
2.1.2 Deep Neural Networks 9
2.2 System Power Modeling Solutions 9
2.2.1 Analytical Power Prediction 10
2.2.2 Microbenchmark Power Prediction 11
2.2.3 Machine Learning Power Prediction 12
2.3 Container Technology 13
2.3.1 Container Overhead Analysis 14
2.3.2 Container and Virtual Machine Comparative Analysis 15
2.3.3 Containers in HPC 17

Chapter 3 Methodology 19
3.1 Experimental Setup 19
3.2 Standard of Error - MAE 20
3.3 Load Migration Triggering Points 23
3.4 Datasets 23
3.4.1 SPEC dataset 25
3.4.2 Micro-benchmark stress tests 25
3.4.3 PARSEC benchmarks 26
3.4.4 SPLASH benchmarks 26

3.5 Load Migration Workloads 29
3.6 DNN Model Input Standardization 29

Chapter 4 Server-Wide Hybrid Power Model 32
4.1 Analytical Power Model 34
4.2 DNN Power Model 35
4.2.1 Cross-Validation 35
4.2.2 Model Performance 37
4.3 Random Forest Classifier 38

Chapter 5 Container DNN Power Model 41
5.1 Container DNN Model 42
5.1.1 Model Performance 43

Chapter 6 Evaluation 46
6.1 State-of-the-Art Implementation 47
6.1.1 Recursive Autoencoder Power Model 47
6.1.2 Design Details of Recursive Autoencoder Power Model 49
6.2 Analytical Power Models 51
6.3 Power Model Accuracy Evaluation 53
6.4 Hybrid Power Model Load Migration Decisions 53
6.5 Power Model Load Migration Evaluation 54

Chapter 7 Conclusion 57

Bibliography 59

LIST OF FIGURES

Figure 1.1: Proposed Hybrid Power Model Architecture : the power model selector chooses the best power model (between DNN model and analytical model) based on the current system condition. 4

Figure 3.1: MoCA Lab Server Setup: GPU Moderator polls worker nodes for system parameters and predicts instantaneous power consumption by utilizing the proposed hybrid power model. GPU triggers migration when threshold is reached on overloaded worker nodes to underutilized nodes. 22

Figure 3.2: Instantaneous Power Consumption in Watts: X-axis is CPU utilization (%). Two distinct platforms show different energy efficiency curves where Gimbap (left) and Kraken (right) have peak energy efficiency at 50% (135 Watts) and 60% (130 Watts) CPU utilization, respectively. 24

Figure 3.3: DNN Training/Validation Accuracy with Dynamic Range as Ground Truth 31

Figure 3.4: DNN Training/Validation Accuarcy with Absolute Power Value as Ground Truth 31

Figure 4.1: Server Power Prediction Errors of Analytical Models w.r.t. Application Group's Memory Intensity 34

Figure 4.2: Prediction Accuracy in MAE of Various DNN Architectures: Each comma separated number presents the number of neurons per layer in order. The right most DNN that has the lowest MAE is selected as the final design. The selected DNN has eight layers of 11, 16, 32, 64, 32, 16, and 8 neurons per layer and the last layer with one neuron. 36

Figure 4.3: Analytical Power Model vs DNN model MAE: DNN model performs best on TEST workload while Analytical Power Model performs best on Training workload. 37

Figure 4.4: Pearson's Correlation Coefficient between System Parameters and Server Power (closer to 1 has stronger relationship) 38

Figure 5.1: Prediction Accuracy in MAE of Various DNN Architectures: Each comma separated number presents the number of neurons per layer. The right most DNN that has the lowest MAE is selected as the final design. The selected DNN has six layers of 8, 16, 32, 16, and 8 neurons per layer and the last layer with one neuron. 42
Figure 5.2: Pearson Correlation graph for container parameters with respect to power. Parameters with >0.7 correlation were selected 44
Figure 5.3: Prediction Accuracy of Container DNN Model 44

Figure 6.1: RAE Architecture [39]: Single autoencoder node is depicted in frame. Successive nodes encode the output from previous node and input at time $i \in t - \tau, t$ 48
Figure 6.2: RAE loss over 3500 training samples 50
Figure 6.3: Equation 4.1, 4.2, 2.2 MAE on selected SPEC Platforms from SPEC-power2008 public dataset 52
Figure 6.4: Equation 4.1, 4.2, 2.2 MAE on MoCA lab platforms with Platforms/Dataset labelled. KM refers to Kraken and Medusa while KG refers to Kimchi and Gimbap 52
Figure 6.5: Power Model Performance on the CPU and MEM workloads 54
Figure 6.6: Model Selection Results: When cache miss ratio is high, DNN model is selected to trigger migration. 55
Figure 6.7: Data Center Power Consumption Comparison When Using Different Power Models: Blue graph is state-of-the-art RAE, yellow graph is Analytical model, and green graph is Hybrid Model 55

LIST OF TABLES

Table 3.1: MoCA Lab platform specifications 21
Table 3.2: Load Migration Triggering Point for Distinct Platforms . . . 23
Table 3.3: Containerized Workload Benchmark Suite 27
Table 3.4: Test Set and Training Set Dataset 28
Table 3.5: Memory Intensity of Individual Workloads 28
Table 3.6: Workloads Used In Load Migration Experiments 28
Table 3.7: SPEC Platforms used for Analytical Power Model evaluation 29
Table 3.8: Platforms and Their Dynamic Power Ranges: Minimum is found by finding the median idle power consumption and median power consumption at 100% CPU utilization across 500 datapoints, respectively 30

Table 4.1: Random Forest Classifier Performance on Compute- and Memory-Intensive Workloads 40

Table 5.1: Parameters selected for input in Container DNN 43

Table 6.1: Parameters used in Original Six-parameter RAE and Revised 11-parameter RAE 47
Table 6.2: Details of RAE Submodels 50

VITA

2018	B. S. in Computer Engineering, University of Pittsburgh
2021	M. S. in EECS, University of California, Merced

ABSTRACT OF THE DISSERTATION

Hybrid Power Models for Data Center Energy Efficiency

by

Nigel Bernard
Master of Science in Electrical Engineering and Computer Science
University of California Merced, 2021
Professor Hyrean Jeon, Chair

With the growing complexity of big data workloads that require abundant data and computation, data centers consume a tremendous amount of power daily. In an effort to minimize data center power consumption, works in the literature developed power models that can be used for load balancing algorithm decision making. There are several difficulties that make power modeling a nontrivial task. For one, the inter-dependencies between components must be captured in addition to the direct effect components have on energy consumption. Additionally, a power model should persist for a wide variety of workloads and generalize to different platforms. In the past, analytical power models mainly focused on single-variable CPU utilization functions to predict power consumption. However, with the increase of system memory and disk/cache accesses in emerging workloads like machine learning, these models face a significant accuracy reduction because they do not account for power hungry memory related events. As a result, machine-learning-based power models attempt to consider relevant parameters that capture these memory relevant events in emerging workloads. However, these machine-learning-based power models exhibit high latency and accuracy improvement mostly on non-compute-intensive workloads. Addition-

ally, machine learning power models are not easily generalized to different platforms because they are trained with profiling data of certain server nodes, given the increasing hardware heterogeneity in data centers. Additionally, as more datacenters are migrating to run containerized applications to leverage the isolation, security and scalability of containerized technology, there is a lack of study in the literature regarding container power modeling. Thus, load migration algorithms are unable to make accurate migrating decisions regarding the appropriate containers. This thesis tackles these issues by proposing several ideas. First, a hybrid power model is proposed that selects the best power model out of a lightweight analytical model and a more accurate DNN model by considering prediction accuracy and performance/power overhead. A workload classifier is incorporated in the hybrid power model that evaluates the common characteristics of currently working workloads and determines the better power model that captures the characteristics. Then the hybrid power model outputs a power prediction specific to the power model the workload is classified as. Secondly, a ground truth standardization method is proposed that enables one machine learning model to be used by various heterogeneous server nodes without a significant accuracy discrepancy due to server-specific features. Thirdly, a novel container power prediction is proposed to predict the power draw of individual container applications for accurate load migration decision making. We compare our hybrid power model against a state-of-the-art recursive autoencoder power model (RAE) and an analytical power model. Our experiments show that our hybrid power model provides up to 5-10% energy savings when integrated as a load migration trigger, compared to RAE and analytical power model.

Chapter 1

Introduction

As almost every computing device is connected to the internet these days, demands for data center services have increased significantly, which leads to concerns about data center energy consumption. Between 2010 and 2018, global network traffic (the quantity of data traversing the internet) increased more than ten-fold, while global data center storage capacity increased by a factor of 25 in parallel [16]. In 2020, data centers in the USA consumed 91 billion electricity units (kW/h) yielding $13 billion per year for electricity bills in the business sector [16]. In fact, some of the world's largest data centers can each contain many tens of thousands of computing devices and require more than 100 megawatts (MW) of power capacity, which is enough to power around 80,000 U.S. households. The significance of maximum energy efficiency in data centers cannot be overstated.

There have been efforts from both academia and industry to minimize data center energy consumption. Some studies used hardware actuators such as various sensors to monitor and control the power and thermal levels of individual hardware components [18, 29, 38]. Some other studies used software profiling tools to collect system parameters such as CPU utilization, cache miss

rate, network IO, and many more, and developed power consumption models [24, 27, 30, 35]. Software profiling-based power modeling is a more flexible and cost efficient alternative when hardware equipment is not available for monitoring system parameter modeling.

In this thesis, we explore limitations and advantages of various software profiling-based server power models and proposes a novel lightweight and highly scalable Hybrid Server Power model. We demonstrate the effectiveness of the new server power model in a small-scale data center by integrating the model into a data center workload scheduling algorithm. The proposed power model showed superior prediction accuracy and the lowest overall data center power consumption by assisting the workload scheduling algorithm, compared to state-of-the-art solutions [6, 43].

Server power modeling has been extensively studied. However, under increasing heterogeneity in both software and hardware [26], there is no single model that rules all possible server configurations. For example, unlike many conventional analytical models assumed, CPU is no longer the only significant power consumer in server systems. In the big data era, heavy memory traffic is an inevitable computing pattern shown in many server workloads. Various recent studies demonstrate notable impact of memory subsystems to the server power consumption. Therefore, the accuracy of CPU-utilization-based analytical models is questionable.

To accommodate the hardware heterogeneity, some studies included non-compute components such as memory, storage, and network for the power modeling [28, 47, 52]. However, including new parameters to existing analytical models and finding the optimal weight values have a scalability issue given the ever-increasing system heterogeneity. In domain-specific computing era, various new accelerators are being adopted in server systems that prove to be significant power drivers [25]. For a hassle-free model development, some recent studies

adopted machine learning algorithms [31, 39, 41, 49, 50]. To reflect non-linear impact of various system parameters towards server power consumption, while other researchers used deep learning algorithms [31, 39, 41]. These studies used hundreds to thousands of past system parameters and power measurements to train a power prediction model. These models show superior performance than many conventional models. However, given the inherent compute intensity of machine learning algorithms, machine learning-based power models themselves may cause performance and power overhead. Therefore, the usage of those compute-heavy power models should be carefully determined so that the models can be used only when necessary.

The hardware heterogeneity can be also sourced from CPU generations and vendors. Even in the same Intel CPUs, different CPU generations show different energy efficiency (CPU utilization per watt) patterns. According to our stress test results, Intel Cascade Lake CPUs and Intel Broadwell CPUs show peak energy efficiency point at 60% and 50% of CPU utilization, respectively. Also, the amount of on-chip resources such as caches cause different idle power consumption in different CPU models. However, many previous studies have not considered this low level heterogeneity, leading to issues with model's ability to generalize. For example, if absolute system power values are used as ground truth of machine learning algorithms, the prediction accuracy will vary depending on the underlying CPU model. Therefore, there should be a way to generalize the power prediction results across different CPU generations.

To reduce data center power consumption, the power models can be used as a delimiter of controlling individual server loads such that all servers can maintain the loads at their energy efficiency level [56]. To migrate loads across servers, it is important to understand individual workloads' power contribution. Though a few studies assumed that workload characteristics are known apriori through offline profiling, there will be a scalability issue if the scheduling relies

on static information. Also, profiling itself will take up extra power consumption. Therefore, there should be a mechanism that dynamically capture power contribution of individual workloads. Given the increasing usage of containers in data centers for more secure computing and efficient resource management, the mechanism should predict power consumption of individual containers. There have been some studies that analyzed virtual machine-based workload execution [53]. However, to our best knowledge, there has not been a study that designed container-level power model.

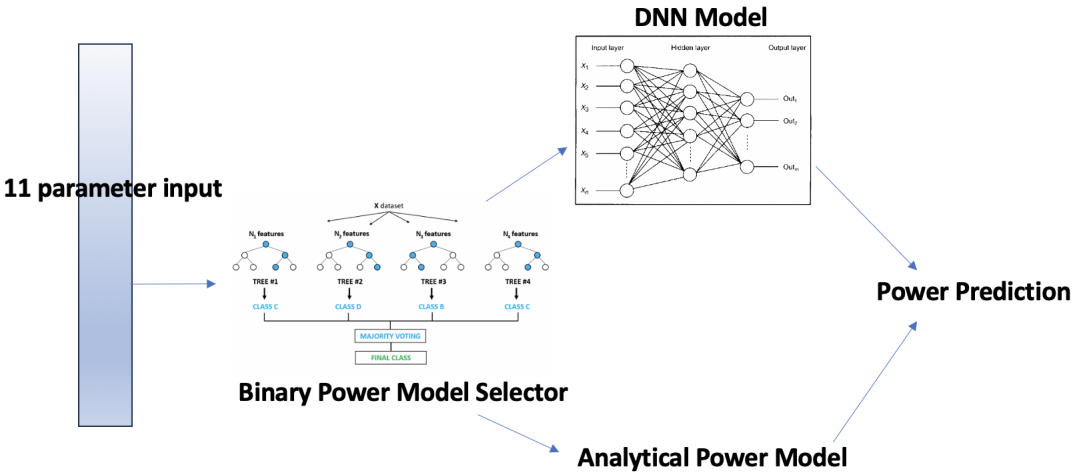

Figure 1.1: Proposed Hybrid Power Model Architecture : the power model selector chooses the best power model (between DNN model and analytical model) based on the current system condition.

This thesis tackles the aforementioned issues with the following approaches:

- A novel hybrid power model is designed to both tackle the accuracy deficiencies of conventional analytical models and the computational overhead of machine learning-based models. Figure 1.1 shows the overall archi-

tecture of the hybrid power model. Our model internally calls either a lightweight conventional CPU-utilization-based analytical power model or a DNN power model depending on the sampled workload characteristics at runtime. The DNN power model is trained with 11 system parameters that show over 90% correlation (when using Pearson's correlation) with server power consumption. Because the DNN power model associates server power consumption with both CPU and memory related events, our DNN model provides at least 13% greater prediction accuracy than conventional models for memory-intensive workloads. On the other hand, the conventional analytical model is orders of magnitude faster than DNN model prediction and shows a comparable accuracy for compute-intensive workloads. To take advantage of both models, our hybrid power model incorporates a random forest classifier that selects the best power model between DNN and analytical based on the classification results of workloads at run time. If the system statistics reveal that the currently executing workloads are mostly compute intensive, analytical model is selected. Otherwise, DNN model is selected. Our random forest classifier yields an average of 83% classification accuracy when classifying 1000 datapoints from both compute- and memory-intensive workloads.

- A new ground truth is designed to generalize the prediction results across different server types and CPU models. The new ground truth only takes care of dynamic portion of power consumption by excluding idle power. Also, to reflect different energy efficiency rate of individual servers, the new ground truth takes the relative power level within the maximum dynamic power range (maximum power at the peak utilization - idle power). Once a power model predicts the relative dynamic power value, the actual power value is calculated with a quick post processing. With the new ground truth, our models showed up to 50% increase in accuracy compared to

those that were trained/validated on absolute power values.

- We demonstrate the effectiveness of the proposed power model through exhaustive evaluations and thorough analysis. Our evaluations include experiments and analysis on the prediction accuracy of various power models for containerized applications as well as overall power savings in a small-scale data center when the power models are integrated to the first-fit decreasing (FFD) workload scheduling algorithm. Inspired by the state-of-the-art energy-proportional data center scheduler [56], we defined energy efficiency thresholds for individual server nodes and triggered migrations when any server exceeds the threshold. Our hybrid model is shown to yield up to 10% more energy savings when compared to a state-of-the-art power model.

- We compiled a containerized benchmark suite with workloads that represent various types and levels of computing loads of data centers. The benchmark suite workloads include micro-benchmarks that provide stress tests for important computing components such as CPU, memory, and storage etc., as well as assorted workloads of SPEC, PARSEC, and SPLASH benchmarks that provide single and parallel processing workloads. All our experiments used this benchmark suite to show the performance of the tested power models for various data center workloads, thereby providing unbiased conclusions.

The remainder of this thesis is organized as following. In Chapter 2, we discuss background information and related work. In Chapter 3, we describe the experimental setup, load migration trigger points and datasets. In Chapter 4, we explain the proposed hybrid power model. In Chapter 5, we evaluate the power impact of containerized application executions and propose a DNN container power model. In Chapter 6, we compare prediction accuracy and effectiveness

for data center power savings of our hybrid model, analytical models, and a state-of-the-art recursive autoencoder designs. In Chapter 7, we conclude the thesis and discuss potential future directions.

Chapter 2

Background and Related Work

2.1 Background Technologies

2.1.1 Classifiers

Classifiers are used for understanding the patterns of a given input data by extracting the common features and outliers among datapoints. K-NN Classifiers work to classify all training vectors in N dimensional space. Then, when classifying a novel input, K-NN chooses the class corresponding to the closest K nearest known datapoints in N-dimensional space. A Decision Tree builds classification models for its inputs in a tree-like structure. The rules for decisions along the tree's path are learned according to training inputs. A Random Forest Classifier consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest decides on a class prediction for an input. The Random Forest Classifier then aggregates the votes from different decision trees to decide the final class of the input sample. Random Forest Classifiers are highly efficent, and can handle thousands of input variables. Random Forest Classifiers are shown in the literature to have high

learning capabilities, as they infer accurately even when a large part of training data labels are missing from the dataset.

2.1.2 Deep Neural Networks

Deep Neural Networks are a kind of machine learning that uses multi-layers of neurons and back-propagation for machine training. Each layer in a deep learning model translates the representation of input data from one form to another by extracting partial information from raw input data. This approach is implemented from the first layer where raw data is fed, till the final layer where final representation is obtained. Each layer extracts a type of features from input and provides the abstracted data to the next layer. Deep learning consists of two phases: training and inference. At training phase, a deep learning model is developed by feeding the model with a large dataset. To make the machine learn the meaning of the input data, feed-forward and back-propagation paths are interleavingly processed. In back-propagation, the prediction and actual result are checked and the weight values of all layers are refined with respect to the prediction error. At inference phase, only feed-forward path is executed with the trained weight values.

2.2 System Power Modeling Solutions

There is substantial work regarding software solutions to power modeling. Previously, most works focused on linear and polynomial models utilizing CPU percent utilization for instantaneous power prediction. More recently however, research has evaluated a variety of deep learning models to address the diminishing accuracy of previous models when modeling emerging workloads with memory intensive operations. In this section we both introduce works that we will use as comparison and provide background and precedent for our approach.

2.2.1 Analytical Power Prediction

Farahnakian et al. [27] presented a linear regression based CPU usage prediction (LiRCUP), for VM migration. Specifically, the authors estimated the future CPU usage to predict overloaded and under-loaded hosts; then, some of VMs are migrated to other hosts before an SLA violation occurs. Consequently, such a solution relies on early migration of VMs even when the current resource usage of the considered hosts is still acceptable, thus resulting in unnecessary migrations.

To calculate power consumption of a server, Han et al. [30] use a CPU utilization linear model to account for the servers idle power consumption. An idle server consumes about 66-70% of its fully loaded configuration. This is due to the fact that servers must manage memory modules, disks, I/O resources, and other peripherals in an acceptable state. The rate of change of the linear model is given by the percent CPU utilization times the dynamic power consumption of the server.

$$Power = P_{MAX} \times (0.7 \times P_{IDLE} + 0.3 \times Util_{CPU}) \tag{2.1}$$

In their creation of a GreenCloud, a simulation environment for energy-aware cloud computing data, Kliazovich et al [35] used a single variable CPU Utilization model to account for the recent implementations of Voltage/Frequency Scaling (DVFS) [40] in server platforms. DVFS introduces a tradeoff between computing performance and the energy consumed by the server. The DVFS is based on the fact that switching power in a chip decreases proportionally to V2*f. Moreover, voltage reduction requires frequency downshift. This implies a cubic relationship of CPU frequency with Power.

$$P = P_{Idle} + Pf * f^3 \tag{2.2}$$

2.2.2 Microbenchmark Power Prediction

D. Economou et al. [24] proposed to develop an online power model that can be deployed in data center schedulers without adding up to the hardware cost to the end user. Their study is based on understanding the server power consumption across different systems, breaking down of different components and temporal variation. Power characterization was done based on the average dc power of the components by running different benchmark applications on system under test. The power prediction model takes in performance metrics at each point in time and produces an output that estimates AC power at each point in time. This power model gives a reasonable approximation of power when high level metrics are given. However, SPEC benchmark suites do not container containerized applications as well as

RAPL (Running Average Power Limit) technology present in Intel architecture provides power limiting features and accurate power readings for CPUs and DRAM which are easily accessible through different interfaces on large distributed computing systems. Evaluations on RAPL show promising results, as readings are highly correlated with overall system power and have negligible performance overhead. [33] shows the utility and gives a use case of microbenchmarks within software power modeling, but RAPL is only included on Intel processors. Our deep learning power model utilizes a containerized application suite to quickly learn how the components per system relate to power consumption and may be utilized across a variety of different processors.

Researchers in [37] investigate the quality of microbenchmark suites with a focus on suitability to deliver quick performance feedback and CI integration. The authors studied ten open-source libraries written in Java and Go with benchmark suite sizes ranging from 16 to 983 tests. The authors then provided a rating system to evaluate these benchmarks in terms of stability and slowdown. However, these benchmark suites are not containerized applications and do not

stress the system on a granular basis much like our approach does. Rather the benchmarks in this study work to stress test a system to a particular components threshold. We do not believe that training a machine learning model on data-points sampled from systems stressing at full capacity would give the model the capacity to recognize the power consumption of a system at lesser-intermediate loads.

2.2.3 Machine Learning Power Prediction

Shen et al. [49, 50] used LSTM-based multi-granularity power prediction. With the predicted power of immediate (next second) and further future (next 30 second), they aim to support efficient data center resource assessment. Hsu et al. [31] developed DNN-based power prediction model and used it to task allocation problem in a data center. The model predicts the total amount of power to complete a given task for the current server workload distribution and data center air conditioner temperature. A central controller decides workload allocation to minimize the total energy consumption for the series of workloads. Li et al. [39] used a recursive auto-encoder to predict server power consumption by using server power history data, per-server system counters, and total power draw of a data center. They predicted the current power consumption with server performance counters and the future power consumption with the historical power data. This work is the closest to ours. However, they only considered six system parameters without detailed correlation analysis. Also, they assumed a bare workload execution while we consider containerized execution, which is a necessary consideration as containers are becoming more prevalent within cloud environments and IoT systems [23, 46]. Also, we evaluated our model in a workload balancing problem. Lin et al. [41] evaluated four artificial neural networks (ANNs) for power prediction. They collected 16 system param-

eters and developed four different neural networks including LSTM and ENN. Unlike this study, we provide a clear correlation between system parameters and the server power consumption and design a hybrid model that considers both accuracy and overhead.

Machine learning has been also adopted in load forecasting to understand the power consumption pattern of server systems. Increasing energy demands are met efficiently by the use of smart grids. It is necessary to forecast short term load accurately and quickly so that power needs can be met by smart grids. Tinghui Ouyang et al [1] use a Gumbel-Hougaard Copula model to identify dependency between temperature, electricity price, and power load. Using the relation analysis, a deep belief network (DBN) to determine the day ahead and week ahead power load forecasting. It presents a LSTM machine learning model to forecast short term load. On comparing with the ARMA and SARIMA models, LSTM promises higher accuracy. Jatin Bedi et al [19] also use machine learning as they are very effective in handling non-linear problems. They use a D-FED model that they created which outperforms various standard machine learning approaches such as recurrent neural networks, support vector machines and artificial neural networks. These works present a basis for our novel work involving gradient containerized benchmarks for training.

2.3 Container Technology

Since containers have been introduced as a lightweight virtualization system, a variety of domains are looking to evaluate and implement containerized applications to take advantage of the hardware independence, isolation and sustainability of the technology. The related works in this section investigate the viability, advantages, and disadvantages of containers in different domains. In general, these works lend credibility to the utility of our container power model

design in future works. Reproducibility in experiments on both virtualized and real world hardware is a challenging task for a wide variety of reasons. For instance, because researchers conduct experiments on different hardware platforms and software environments, the effects of these various environments' must be considered when attempting to reproduce experiments from existing work. Jimenez et al. [32] assessed the distinct characteristics of containerized technology that can reduce or eliminate the hardware and software complexities involved in reproducing experiments. A hardware mapping methodology is proposed to maintain an execution and hardware profile for generalizability. The authors discussed the benefits and limitations of using containers in tandem with this methodology as well as demonstrate a use case of container technology applications in distributed storage systems. The researchers showed that their system derived similar results when running a scalability experiment from Ceph [55] after replacing older hard drives with newer models. This work demonstrates the utility for containerized applications within the system performance analysis domain. With the growing use of containerized applications across domains, workload balancing with respect to containerized applications must be considered for data center energy efficiency.

2.3.1 Container Overhead Analysis

Containerized technology's versatility makes it more likely to be viable than a hypervisor alternative when constrained environments desire virtualized technology. For instance, domains such as Internet of Things (IOT) and Software Defined Networking (SDN) require a lightweight virtualization software to be at all viable. In an effort to evaluate the costs of container overhead on IOT devices and therefore lend credibility to its use, [45] compares the performance of native execution of benchmarks to containerized execution. To properly evaluate the

overhead consumption of containers, [45] uses synthetic benchmarks to generate different types of workloads to challenge a specific sub-system of hardware under test. CPU, Network I/O, Disk I/O, and Memory were all evaluated with the sysbench [11] testing suite. Researchers then evaluated performance metrics according to real-world workflows with an Apache server servicing requests and database instances. When running the benchmarks on top of a Raspberry Pi 2, the authors found that there is an almost negligible impact (under 5%) of the container virtualization layer compared to native execution. In fact, there is an increased efficiency of up to 7% on Dokcer UDP Client instances. The authors research lends credibility to the more widespread use of virtualized technology within constrained environments. This may lead to the need for workload balancing within these domains as well.

2.3.2 Container and Virtual Machine Comparative Analysis

Because a container only includes the executables and dependencies required to run specific applications, they are much more lightweight than hypervisor alternatives. The authors of [60] compare the performance and overhead of VM and container technology on a five server testbed. The authors perform a gradient analysis of the performance of multiples of instances of containers and virtual machines running simultaneously. This gradient approach perhaps provides a more accurate depiction of workloads than previous comparative studies in this domain. Their testing workload consists of Kmeans, Logistic Regression, Pagerank, and SQL Join. The authors found several notable results from their experiments. Most notably, they found that containerized applications are more convenient in startup and deployment for system administrators, display greater capacity for scalability in big data workloads, and achieve higher CPU

and memory utilization with similar workloads. This study provides evidence for the transition from hypervisor technology to containers for applications that require virtualization.

In [22], the authors compare Docker containerized applications to Hypervisor virtualization. The authors present a model for distributing Docker containers for increased efficiency by exploiting the capacity for Docker containers to share common files because their images are constructed from layered filesystems. Additionally, the authors evaluate Docker's performance in a HPC datacenter as well as evaluate their performance on prevailing HPC workloads. Chung et al finds that Docker containers are more efficient in terms of execution with data intensive workloads as well as overhead reduction compared to Virtual Machines. The authors work adds to an vast field of study that evaluates the two prominent virtualization technologies. When work evaluates container technology to perform more efficiently than hypervisor technology, it lends credence for applications to take advantage of virtualized technology when hypervisor overhead would render it impossible.

Before concluding the lightweight flexible nature of containerized applications outweigh the security and stability of hypervisor visualization, researchers compare the performance between Docker and virtual machines with respect to makespan, execution time, and CPU/Memory utilization [20]. In particular, the authors assess the performance differences in a big data use case, implementing the Apache Spark framework with both Docker and Virtual Machines to evaluate each with regards to makespan, execution time, and CPU/Memory utilization. Researchers organized up to N instances of both VM and Docker containers for worker nodes in their respective Spark instances. Then, they tested a variety of different Machine Learning, Graph Computation, and SQL Query applications. From their results, researchers advise to use Docker for map and calculation intensive applications because Docker provides lightweight op-

eration, copy-on-write (COW) and intermediate storage drivers for calculation instances. These results lend further credibility to a transition from hypervisor to container virtualization in the near future.

2.3.3 Containers in HPC

In high performance computing systems, virtualization was rarely utilized. Hypervisor technology's overhead outweighs the benefits of hardware independence, availability, isolation and security. However, with the rise of lightweight containerized technology, virtualization within HPC becomes more viable. Xavier et al. [57] evaluate the feasibility of containerized applications within emerging High Performance Computing workloads and environments. Linux VServer, OpenVZ and Linux Containers (LXC) instances were ran on various HPC workloads for isolation and security evaluations. To test a container technology's capacity to isolate, researchers ran different containers running different stress tests simultaneously and measured the performance degradation. Because containers share the same kernel, a container that greedily consumes resources will impact the execution of other containers. Researchers used the Isolation Benchmark Suite [33] and compared the baseline execution time of a single container and compared it to instances where several containers were executing simultaneously. Researchers found 10% performance degradation with network intensive stress tests, and found no significant performance loss with CPU intensive stress tests. To test performance, researchers evaluated containers on various microbenchmarks testing memory, CPU, and network. The researchers found that the containerized applications had near native performance but incurred faults due to security issues related to namespace designation with cgroups. This study further lends credibility to the emerging use of container virtualization in different domains. However, this study also notes security issues with container

technology. If these security issues are resolved it perhaps lends further credibility for the need of container power modeling for appropriate energy efficient scheduling.

Containerization has emerged as a new paradigm for software management within distributed systems. Containerization provides a new methodology for software development, management, and operations for online services. Containers enable developers to specify the exact environment of a software solution. Beyond this, there are several notable advantages to using containers within HPC environments. They include composability - allowing developers to explicitly define the modular design of their environment/project, portability- capacity to share containerized product across different computing systems and platforms, and version control integration- users are able to access public images defined on Docker Hub for painless updates and integration. However, this study also notes potential drawbacks of containerized applications in HPC [59]. Most notable are security- containers often allow root-level access to users which is undesirable in industry and networking- unnecessary in HPC workflows as often times HPC clusters utilize custom interconnects that bypass TCP/UDP protocols. Younge et al. propose that Singularity [36], custom Docker images created by Lawrence Berkeley National Laboratory address these issues. Singularity improves upon Docker security concerns by mounting volumes without granting root access. Singularity also addresses the would be unnecessary network sharing in containers by image wrapping for cross user access to shared resources. Researchers found that Singularity deployment on a Scray system had no significant overhead with performance. This work considers the possible drawbacks of containerized applications and provides a use case of containers to address these issues. Ultimately, it presents a notable nuanced approach towards using containerized applications within constrained supercomputing resources.

Chapter 3

Methodology

3.1 Experimental Setup

We use four worker servers (two distinct platforms) and one GPU moderator server from our MoCa Lab located at the University of California, Merced to gather various datasets while running emerging data center workloads. The GPU server, Galbi, polls each worker server every 5 seconds for relevant parameters used within our DNN model. To gather these parameters, each server uses psutil [5] to gather system parameters and contacts docker API for container statistics. When prompted by the GPU moderator server, the worker nodes reply with the relevant statistics. Using these parameters, Galbi classifies the workload with a random forest classifier, then uses either the trained DNN output or the conventional power model to determine the instantaneous power consumption of each worker node. If any worker node has eclipsed their efficient power threshold determined through research [56] the GPU server will predict the power consumption of each container on that specific node. The GPU server will then migrate the container with the lowest power consumption load to an underutilized server until when the migration source server falls below the effi-

cient power threshold. We choose the lowest power consuming application for migration because it can prevent migration bouncing issues and reduce migration overhead. However, different algorithms such as migrating the most power hungry application may be a faster power unbalancing problem solution. We will explore the pros and cons of the application selection algorithms as future work. Thanks to GPU acceleration, the DNN power model runs 78% faster on the GPU moderator server than on the other CPU-based worker nodes.

Two worker servers (namely Kraken and Medusa) utilize the Intel(R) Xeon(R) Silver 4214 CPU @ 2.20GHz with 3.2 Hyperboost technology. Another pair of worker servers (namely Kimchi and Gimbap) utilize Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz. Intel® Turbo Boost Technology accelerates processor and graphics performance for peak loads, automatically allowing processor cores to run faster than the rated operating frequency if they're operating below power, current, and temperature specification limits [4]. Including this technology within our dataset will most likely increase the intricacies our model must learn, and therefore lower its effectiveness. We turn off this functionality while gathering our dataset.

These heterogeneous server types (e.g., using different CPU model, cache size, and core count) enable us to evaluate the ability for the deep learning model to generalize its learning across different platforms. Given that heterogeneity is a norm of modern data centers due to legacy servers, our evaluation environment provides a realistic setup.

3.2 Standard of Error - MAE

We evaluate our power models by mean absolute error (MAE) shown in equation 3.1. Conventionally, models are evaluated using root square mean error (RSME). However, RSME penalizes larger predictions compared to MAE in its

Platform	**Processor Specs**	**CPU/GPU core count**
Kraken and Medusa	Intel Xeon Silver 4214 CPU @ 2.20GHz (up to 3.2 GHz Turbo Boost)	32 cores
Kimchi and Gimbap	Intel Xeon CPU E5-2620 v4 @ 2.10GHz	24 cores
Galbi	Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz, NVIDIA Volta GPU @ 1.38 GHz	80 CPU cores, 5120 GPU Cores

Table 3.1: MoCA Lab platform specifications

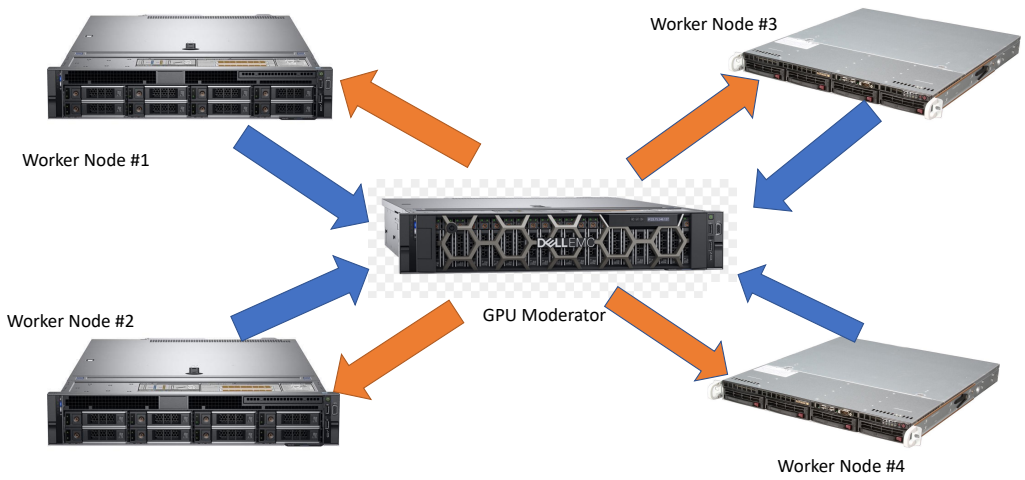

Figure 3.1: MoCA Lab Server Setup: GPU Moderator polls worker nodes for system parameters and predicts instantaneous power consumption by utilizing the proposed hybrid power model. GPU triggers migration when threshold is reached on overloaded worker nodes to underutilized nodes.

loss because it squares the errors before taking the mean error across samples. This can lead to confounding errors that don't necessarily represent the accuracy of particular a power model when comparing results across substantially different sample sizes. Outliers are normally penalized heavily in machine learning, but because we need to make assessments about analytical models with respect to SPEC datasets and our experiments (and therefore different sample sizes), MAE provides a better error standard for comparison for both our purposes and across future works. Additionally, MAE provides a clear expectation of error in Watts.

$$MAE = (\frac{1}{n}) \sum_{i=1}^{n} |y_i - x_i| \tag{3.1}$$

Platforms	Power Threshold (W)
Kimchi and Gimbap	135
Kraken and Medusa	130

Table 3.2: Load Migration Triggering Point for Distinct Platforms

3.3 Load Migration Triggering Points

We take inspiration from the state-of-the-art energy proportional data center scheduling study [56] for our designation of optimal power consumption efficiency points for MoCA Lab's platforms. From Figure 3.2, the kraken/medusa platforms (right) show disproportionate power consumption past the 135-Watt threshold for proportional increases in CPU utilization. For kimchi/gimbap (left) the wide distribution of plotted power consumption points at both 40% and 50% CPU show there is a disproportionate jump at around 130 Watts. We proceed with a 135-Watt and 130-Watt load migration threshold for kraken/medusa and kimchi/gimbap respectively.

3.4 Datasets

To gather datapoints from containers reflecting emerging workloads in datacenters, we ran a variety of applications on each respective platform and collected system parameters using two Linux commands, *psutil* and *perf*, during execution. When collecting datapoints we append the ground truth instantaneous power consumption via the a Wattsup Pro Power Meter [14]. We collected a total of 2000 data points for both the training and test suites respectively. The details about the workloads and the data collection settings are discussed below.

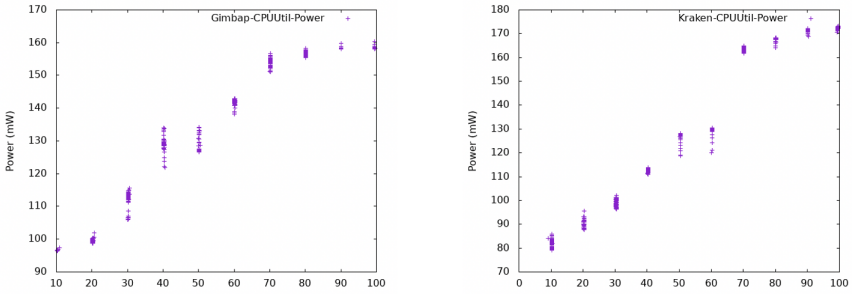

Figure 3.2: Instantaneous Power Consumption in Watts: X-axis is CPU utilization (%). Two distinct platforms show different energy efficiency curves where Gimbap (left) and Kraken (right) have peak energy efficiency at 50% (135 Watts) and 60% (130 Watts) CPU utilization, respectively.

3.4.1 SPEC dataset

SPEC, or Standard Performance Evaluation Corporation is a non-profit corporation formed to establish, maintain and endorse standardized benchmarks to evaluate aspects of emerging computing systems [8]. SPEC is also used in a multitude of prevailing works [34,54] due to its variety of benchmark suites for various evaluation tasks. SPEC provides public results for their SPECpower_ssj2008 benchmark [9]. For every year since its introduction, SPECpower_ssj2008 provides a variety of different platform's average load ranging from 0-100% (multiples of 10) and corresponding average active power. The benchmark workload includes typical server-side Java business applications. It exercises CPUs, caches, memory hierarchy, and the scalability of shared memory processors. Because only the CPU percent utilization is reported with these results, only our conventional power models will be evaluated upon this dataset. This will allow us to evaluate these conventional power models on emerging workloads executing on platforms significantly different from MoCA labs.

3.4.2 Micro-benchmark stress tests

For the DNN model to aptly learn the direct influence of events and parameters on power consumption, we applied a granular stress test approach in an effort to isolate components and active cores during test execution. We use the Linux/Unix tool Stress-ng [10] to generate loads on the CPU, rate of cache access, memory access, interrupts and number of processes. We gathered datapoints by running each specific test 6 different times with a gradient amount of process instances running simultaneously. For kimchi/kraken we ran tests with 1, 2, 4, 8, 16, and 32 active processes and for gimbap/kimchi we ran tests with 1, 2, 4, 8, 16, and 24 active processes. We informally verified that both the CPU load and relevant parameters increased with the increase of processes

via a visualization tool, *htop*. We collected about 10 datapoints per sample for about 10 datapoints per test and 360 datapoints overall for DNN model on each distinct platform.

To gather specific stress tests with respect to the CPU, we utilized a *system-load-generator* docker container [12]. We generated loads of 0-100% (multiples of .5%) on simultaneous core instances identical to the Stress-ng tests for each distinct platform. We collected about 5 datapoints from each multiple of CPU load for a total of 600 CPU utilization datapoints.

3.4.3 PARSEC benchmarks

The Princeton Application Repository for Shared-Memory Computer (PARSEC) [21] is a benchmark suite composed of multithreaded programs. PARSEC is a joint venture between Intel and Princeton University that seeks to provide researchers with an accessible suite of high-performing workloads. The suite focuses on emerging workloads and was designed to be representative of next-generation shared-memory programs for chip-multiprocessors. Because of their dynamic execution profile as is common with HPC applications, we collected about 75 datapoints from each of the 6 PARSEC applications used within our datasets for a total of 450 PARSEC datapoints.

3.4.4 SPLASH benchmarks

We also utilize several SPLASH benchmarks for datapoints included in our test set. SPLASH [48] is a benchmark suite that is popular with scientific studies of parallel machines with shared memory.Research shows that SPLASH-2 and PARSEC benchmark suites are significantly different in term of their workload/architectural characteristics such as instruction distribution, cache miss rate and working set size.

Application/Suite	**Components Stressed**	**Description**
Blackscholes/PARSEC	CPU	Prices a portfolio of options
Bodytrack/PARSEC	CPU/MEM	Tracks a human body through space
Streamclust/PARSEC	CPU/MEM/Network	Online clustering of an input stream
Canneal/PARSEC	CPU	Optimizes the routing of a chip design
Ferrit/PARSEC	CPU	Identify most similar images in a database
Raytrace/SPLASH	CPU	Real-time raytracing
$\text{OCEAN}_{NCP}/SPLASH$	CPU	Computes the cholesky factorization of a sparse matrix
Radix/SPLASH	CPU	iterative integer radix sort
Swaptions/PARSEC	CPU	Pricing of a portfolio of swaptions
Cache/Stress-NG	CPU/Cache Miss	Perform random wide spread memory read and writes
Sleep/Stress-NG	CPU/Interr/Proc	Stress-NG
iCache/Stress-NG	CPU/Instruction Cache Miss	Stress the instruction cache by forcing instruction cache reloads
STREAM/Stress-NG	Cache Miss/Network	perform multiple sleeps of ranges 1us to 0.1s
system-load-generator	CPU	Runs specific load 0-100% on select cores

Table 3.3: Containerized Workload Benchmark Suite

Training Sets	**Test Sets**
Stress-NG, System-Load-Generator	PARSEC, SPLASH benchmarks

Table 3.4: Test Set and Training Set Dataset

Application/Suite	**Mem Intensity**	MPKI
Blackscholes/PARSEC	LOW	1.44
Bodytrack/PARSEC	MEDIUM	8.98
Streamcluster/PARSEC	HIGH	17.38
Canneal/PARSEC	HIGH	21.16
Ferrit/PARSEC	LOW	0.58
Raytrace/SPLASH	LOW	1.24
Swaptions/PARSEC	MED	9.17
Cache/Stress-NG	HIGH	10.45
Sleep/Stress-NG	MED	3.09
iCache/Stress-NG	MED	6.47
STREAM/Stress-NG	HIGH	15.72
system-load-generator	LOW	0.32

Table 3.5: Memory Intensity of Individual Workloads

Container Applications	**Component Stressed**
Blacksholes, Ferrit, Raytrace, system-load-generator (0-100%)	CPU
Raytrace, Bodytrack, StreamCluster, Canneal, STREAM, Cache	MEMORY

Table 3.6: Workloads Used In Load Migration Experiments

Platform	Platform Specifications
SPEC Platform-1	Intel Xeon Platinum 8176 CPU 2.10 GHz (Intel Turbo Boost Technology)
SPEC Platform-2	Intel Xeon E5-2699 v4
SPEC Platform-3	AMD EPYC 7763 2.45GHz

Table 3.7: SPEC Platforms used for Analytical Power Model evaluation

3.5 Load Migration Workloads

For our load migration use case experiments we devise subsets of our applications to stress both the CPU and Memory in Table 3.6. From and overview of our results gathered within Table 3.5, we denote potential containers that can be assigned to servers based on workload type.

3.6 DNN Model Input Standardization

$$DR = P_{max} - P_{min} \tag{3.2}$$

$$GT = (P - P_{min})/DR \tag{3.3}$$

Because of the differences in idle and maximum power consumption between distinct platforms, we apply standardization on the absolute power measurements when gathered data-points. To this end we train a power model that associates our selected parameters with the proportion of dynamic power consumption for a specific platform sampled (Equation 3.3). We denote this adjusted dynamic

Platforms	Dynamic Range
kraken and medusa	66.2-173.1
kimchi and gimbap	94.9-160.4

Table 3.8: Platforms and Their Dynamic Power Ranges: Minimum is found by finding the median idle power consumption and median power consumption at 100% CPU utilization across 500 datapoints, respectively

range of power consumption as a standardized ground truth. We compared the prediction accuracy with DNN models that are trained with absolute power value and with the standardized ground truth for two distinct platforms, as can be seen in Figure 3.3 and Figure 3.4. Note that the DNN models will be explained in detail in Section 4. With the standardized ground truth, the models showed faster convergence in the training because similar value ranges are used as prediction results for different platforms by excluding platform-specific power values such as idle power. More specifically, the DNN model trained with new ground truth yielded a MAE of approximately 2%. To interpret these errors, we consider the dynamic range of each platform from Equation 3.3 and Table 3.8. This is because at runtime, the standardized output of a power model will be reconverted into power consumption in Watts for comparison with each platform's load migration trigger. Specifically, the GPU moderator will first use a power model to predict this standardized value for a platform. Then, the GPU moderator will evaluate Power $= DNN_{output} * RANGE + P_{IDLE}$. Therefore, for kraken and medusa, our model yields a $106.9 * .02 = 2.2W$ error, and for kimchi and gimbap, our model yields a $65.6 * .02 = 1.31W$ error. A DNN model yielded a MAE of approximately 14 watts when trained/validated on our training dataset without platform idle/max power standardization.

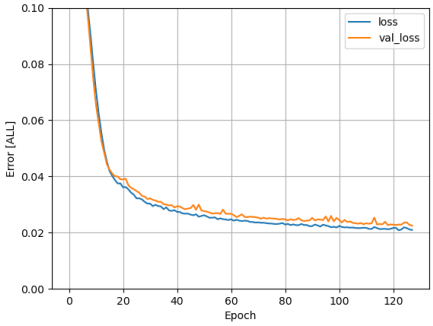

Figure 3.3: DNN Training/Validation Accuracy with Dynamic Range as Ground Truth

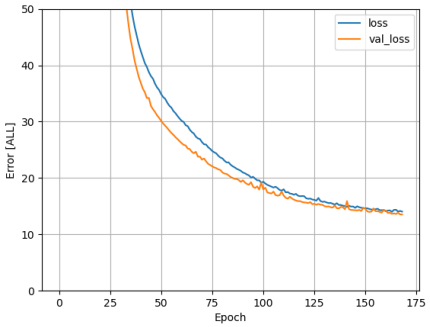

Figure 3.4: DNN Training/Validation Accuarcy with Absolute Power Value as Ground Truth

Chapter 4

Server-Wide Hybrid Power Model

Server power models have been extensively studied. To reflect the dominant contribution of CPU towards system power consumption, many models use linear or polynomial power prediction models that use CPU utilization as a primary parameter. CPU utilization and overall system power consumption is known to be highly correlated, especially for compute-intensive workloads. However, in the growing big data era, analytical power models are less effective due to accounting of power consumption inherent to system memory and disk accesses, and parallel computing communication. Figure 4.1 shows the root mean square error (RMSE) prediction accuracy of various power models against varying memory intensity of applications. Memory intensity is measured by using cache misses per kilo instructions (MPKI). We classified applications into low (less than 2), medium (greater than 2 but less than 10) to high (greater than 10) memory intensive groups according to MPKI, as shown in Table 3.5. We evaluated the prediction correctness of three different deviations of a analytical power model used in cloud systems [43] plane shown for reference simplicity as

Equations 4.1-4.3.

$$P = P_{min} + (P_{max} - P_{min}) \times Util_{CPU} \tag{4.1}$$

$$P = P_{min} + (P_{max} - P_{min}) \times Util_{CPU}^{2} \tag{4.2}$$

$$P = P_{min} + (P_{max} - P_{min}) \times Util_{CPU}^{1/2} \tag{4.3}$$

As can be seen in Figure 4.1, across all the models, the error rate exponentially increases when executing more memory intensive workloads. Therefore, server power models should accommodate various system parameters rather than relying only on the CPU utilization. Instead of explicitly microbenchmarking for each architecture to aptly define a polynomial model for power consumption while considering these different events, machine learning models can be trained to model different architecture power consumption with limited training data. The difficulty lies within sampling particular system parameters that both capture the interdependency between these different events as well as their isolated influence on power consumption. Including parameters that capture redundant events or contribute a negligible amount to power consumption as input to a deep learning power model will hinder its ability to generalize to other workloads.

It is challenging to design an accurate analytical model that accounts for multiple parameters for power prediction. Even when one model is designed, the model is not easy to be generalized or extended to other servers due to wildly varying platforms and architectures configurations, such as cache size, CPU min/max frequency, number of CPUs, and so on. Instead, to account for analytical power model inaccuracies on these workloads, many studies have shown the effectiveness of machine-learning algorithms. However, these ap-

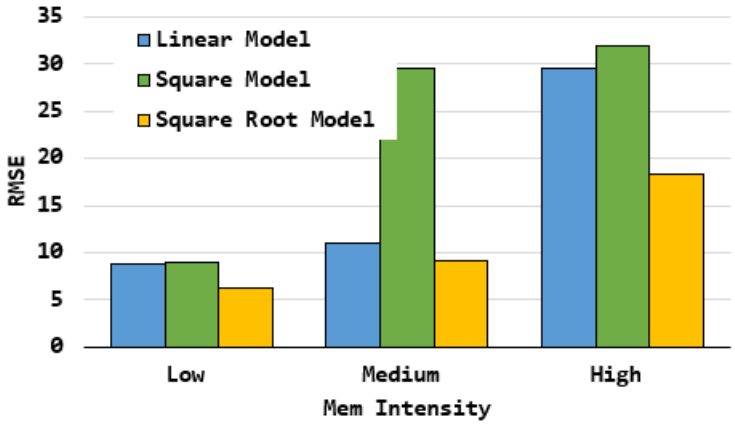

Figure 4.1: Server Power Prediction Errors of Analytical Models w.r.t. Application Group's Memory Intensity

proaches often employ compute-intensive algorithms such as support vector machine (SVM) [42], or tend to overfit with a Lasso regression approach [44]. Machine learning power models also do not account for the quality of service (QoS) constraints necessary for workload scheduling algorithms such as load balancing and migration [17]. In the following sections we describe the analytical power model, DNN and random forest classifier utilized within our hybrid power model.

4.1 Analytical Power Model

We compared the analytical power models depicted in Equations 4.1, 4.2, and 2.2 for use in tandem with our DNN power model. A fully detailed listing of results between the three analytical power models are noted in Chapter 6.2. Equation 4.1 was selected for inclusion within the hybrid power model as its MAE accuracy was up to 50% higher across both SPEC datasets on SPEC platforms and our train/test dataset on MoCA Lab platforms.

4.2 DNN Power Model

To capture complex correlations among multiple to many parameters, deep neural network (DNN) is an effective solution. In order to train a DNN model, the outputs of each layer within the model are fed as inputs into the next layer. Through back-propagation, the DNN weight values are updated/optimized in an attempt to reflect the underlying function describing the parameter/power consumption relationship.

Our DNN model uses 11 system parameters to predict the instantaneous system power. The input parameters were chosen by collecting 2000 datapoints of parameters from our training set and evaluating their Pearson correlation (Figure 4.4) with power consumption on the platforms in MoCA lab (Table 3.1). After evaluating 22 different parameters in Figure 4.4, we selected parameters with a correlation > 0.9 to be included as a set within each sampled data point.

The parameters chosen for our model are CPU frequency, user time, CPU utilization, software interrupts, number of processes, cache miss ratio, virtual memory usage, instructions, and shared memory usage. To justify the inclusion of solely these 11 parameters, we trained/validated 11-parameter and 22-parameter models on the training dataset. Before comparing performance, we first cross-validated each respective model to their optimal DNN architecture. 11-parameter models yield a minimum MAE of 2.3, while models trained with all 22 parameters yield an minimum MAE of >6. We therefore make the assumption that the 11 most highly correlated parameters make the best candidates for inclusion within our DNN.

4.2.1 Cross-Validation

In order to determine the best architecture for the DNN model, we utilized *Talos* [13] which fully automates hyperparameter tuning and model evaluation.

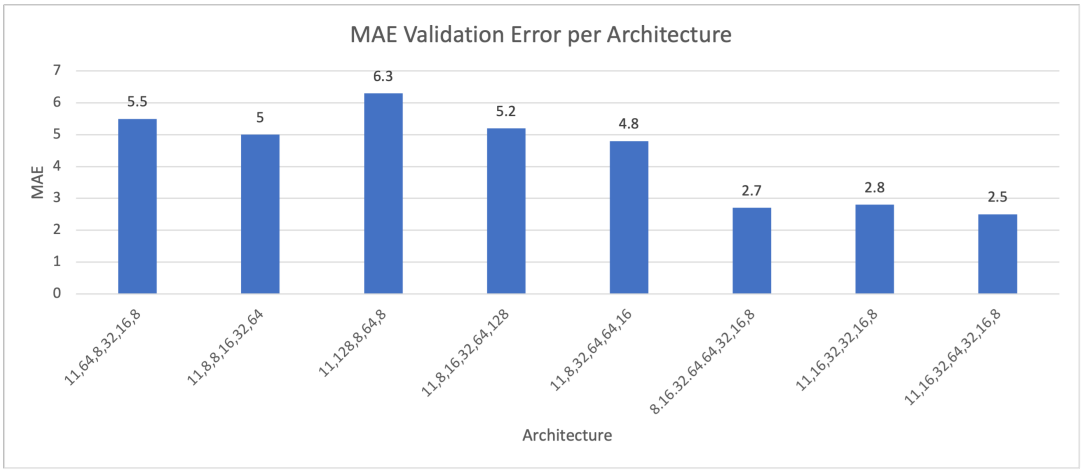

Figure 4.2: Prediction Accuracy in MAE of Various DNN Architectures: Each comma separated number presents the number of neurons per layer in order. The right most DNN that has the lowest MAE is selected as the final design. The selected DNN has eight layers of 11, 16, 32, 64, 32, 16, and 8 neurons per layer and the last layer with one neuron.

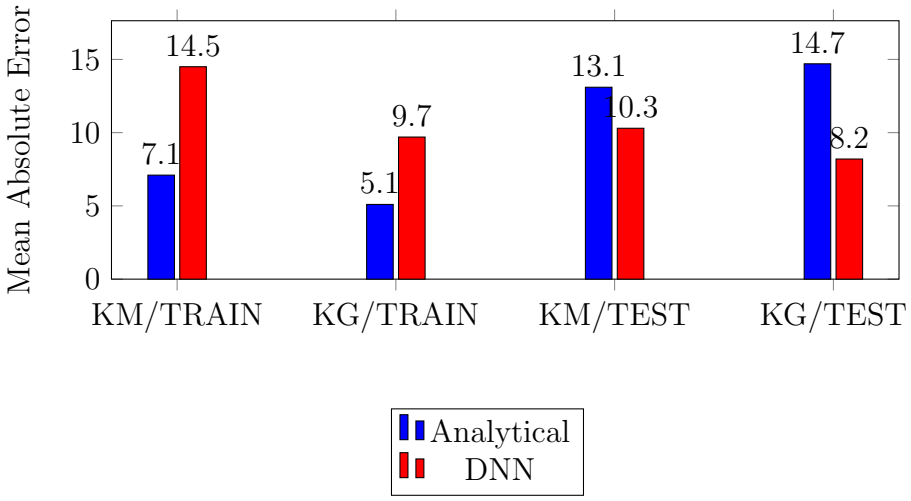

Figure 4.3: Analytical Power Model vs DNN model MAE: DNN model performs best on TEST workload while Analytical Power Model performs best on Training workload.

We select the model with the lowest MAE as the final architecture. Specifically we use an eight-layer fully-connected DNN model with 11, 16, 32, 64, 32, 16, 8 neurons for each hidden layer, respectively. The last layer has one neuron to predict one power value. As seen in Figure 4.2, models with higher amounts of neurons and therefore parameters fail to converge well on our selected dataset. We used a uniform random initializer, Adam optimizer, and RelU and TanH activation functions within our model for training.

4.2.2 Model Performance

On compute-intensive tasks, the conventional analytical power model showed comparable accuracy with our DNN model on both platforms. However on the memory-intensive workloads, our DNN model showed superior accuracy. This trend persisted for the DNN model on all four worker servers in MoCA Lab as

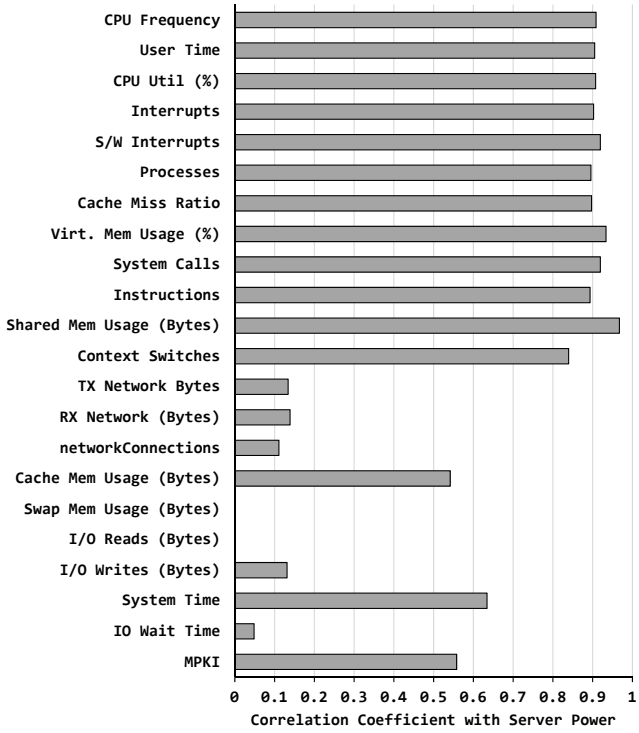

Figure 4.4: Pearson's Correlation Coefficient between System Parameters and Server Power (closer to 1 has stronger relationship)

seen in Figure 4.3.

4.3 Random Forest Classifier

We classify a workload using the sampled 11 parameters of a server according to which model– DNN or analytical– performs best. Our hybrid power model ultimately outputs the power prediction of the predicted better performing model by the random forest classifier (RF). To train this classifier we designed a training workflow. The difficulty within training lies within creating an encompassing enough dataset sampled from simultaneously executing containers that gener-

alizes to other possible workloads within our containerized use case. To this end we design a workflow that executes simultaneous containers of a given application set. For instance, one instance of a testing workload might be three containers executing Streamcluster simultaneously. Another workload might execute four containers of Swaptions and two containers of Bodytrack applications simultaneously. Our workflow then samples the 11 parameters used within our DNN power model, as well as Wattsup power readings. If the analytical power model (Equation 4.1) yields a lower MAE for power prediction than the DNN power model on the specific datapoint, the workflow enumerates the ground truth of that datapoint as 1, and if otherwise as 2. The RF is then trained on this dataset and its associated labeling.

We use a RF from the python package sklearn [7] to make the classifications. If the classifier has a confidence > 0.6 on its prediction of the better performing power model, the selected power model is used for power prediction. This confidence threshold was decided after empirically testing model accuracy on confidences and noting the trends of accuracy on values ranging from 0.2 to 1. If a prediction lies below the confidence threshold, the classifier will default to our DNN model. We decide the DNN to be the default predictor for our hybrid model due to the majority (around 65%) of datapoints classified through a workflow as our DNN model as the better predictor. Our classifier was shown to have an average around 82% accuracy when classifying across our CPU- and memory-intensive workloads.

Workload	**Accuracy**
CPU	77%
MEMORY	87%

Table 4.1: Random Forest Classifier Performance on Compute- and Memory-Intensive Workloads

Chapter 5

Container DNN Power Model

Recently, data-centers have been integrating containerized technology to leverage increased privacy and security capabilities of virtualized technologies. In the past, hypervisors, or virtual machines, were mainly used when privacy, security, and isolation within applications was desired. However, hypervisor technology requires extensive overhead to simulate a separate operating system (OS), including their respective libraries. Containers deviate by designating processes their own namespace, separate resources and shared memory within the same Host OS. Because container applications are now a much more feasible alternative to virtualization, there is a great transitioning of many datacenter workloads to containerized technology. To account for this transition, existing scheduling/load balancing algorithms must require per container power models in order to choose the most optimal container to migrate. We specifically utilize Docker containerized technology [2] in this study as it is the most widely used containerized technology. This chapter demonstrates our approach to building a DNN container power model.

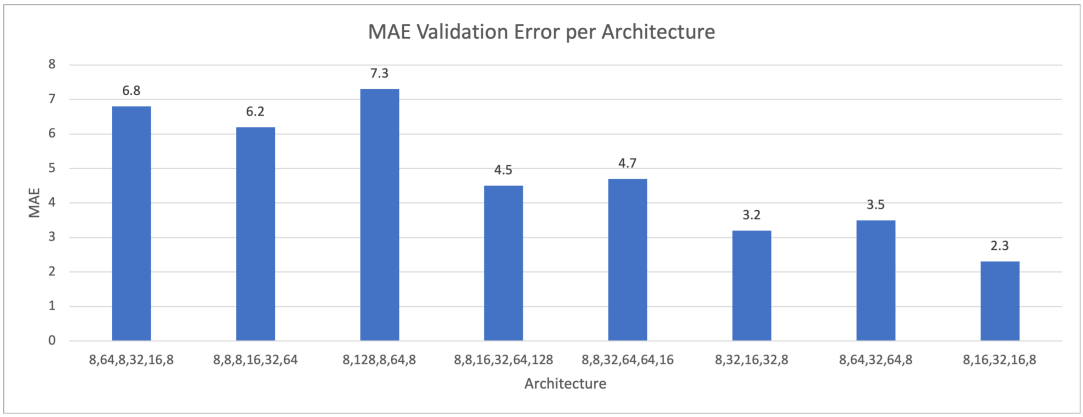

Figure 5.1: Prediction Accuracy in MAE of Various DNN Architectures: Each comma separated number presents the number of neurons per layer. The right most DNN that has the lowest MAE is selected as the final design. The selected DNN has six layers of 8, 16, 32, 16, and 8 neurons per layer and the last layer with one neuron.

5.1 Container DNN Model

Because container applications execute on the same host OS, we evaluate the Pearson correlation of similar parameters to our server-wide DNN power model. However, because hardware- and software-interrupts are kernel wide events, we excluded them from container parameters. Inclusion of these interrupt parameters would confound the DNN model as multiple containers executing simultaneously would yield higher interrupts on the host OS, but would not change individual container contributions to the system's power consumption. To collect datapoints for our container DNN, we first ran each of the training containers separately and sampled 2000 total datapoints containing the listed parameters in Figure 5.1 for every time step.

To ensure we only capture statistics related to a container, we use psutil to first query the system for a process matching a container's name. We then

Container DNN Parameters
CPU Frequency, userTime, idleTime, CPU Utilization, context switches, number of processes, Instructions, cache miss ratio

Table 5.1: Parameters selected for input in Container DNN

perform perf sampling on the relevant processes specific to a container. Note that we collected system parameters through perf and psutil tools without giving specific process ids. By collecting system parameters of container-specific process id, we can model per-container power model.

Again, to determine which parameters to include in our container DNN Power Model, we collected stats via Docker API, perf, and psutil and selected parameters that had greater than 70% correlation in Figure 5.2. Using Talos, we trained/validated across this set to yield the optimal hyper parameters and architecture according to MAE. Because the container DNN power model uses fewer input parameters, it performs better with fewer layers and neurons than the server-wide DNN power model. Figure 5.1 shows the MAE from trial architectures. We selected a model that uses five hidden layers with 8, 16, 32, 16, and 8 neurons per layer. The last layer has one neuron for predicting one power value. We used the standardized ground truth as described in Section 3.6 to account for platform differences.

5.1.1 Model Performance

On the training/validation set containing datapoints from each distinct platform, the model converges to a MAE of around 2.2% as seen in Figure 5.3. With the standardized power consumption ground truth depicted in Equation 3.3, this translates to a MAE error of 3 Watts for kraken and medusa and 2 Watts for

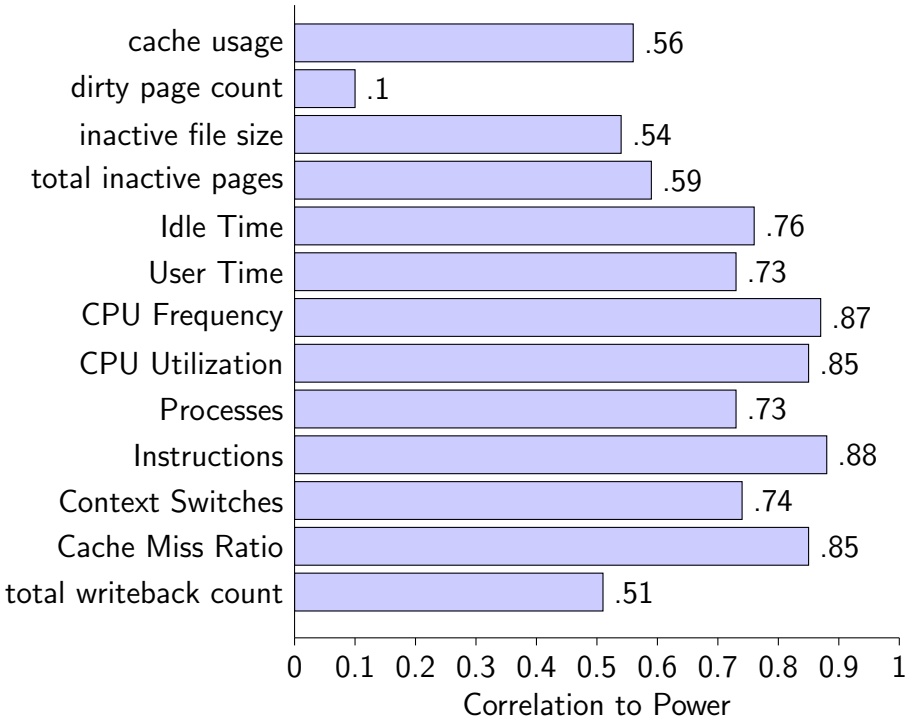

Figure 5.2: Pearson Correlation graph for container parameters with respect to power. Parameters with >0.7 correlation were selected

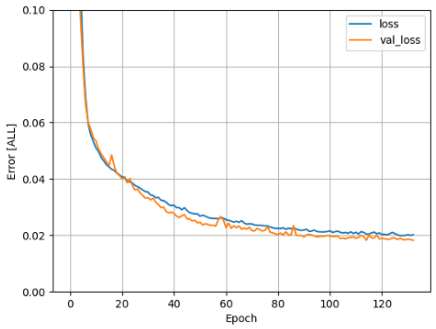

Figure 5.3: Prediction Accuracy of Container DNN Model

kimchi and gimbap.

Chapter 6

Evaluation

In order to evaluate the accuracy and effectiveness of both our hybrid and container power models, we designed a realistic use case scenario that uses either our hybrid power model, a conventional analytical power model, or a state-of-the-art recursive autoencoder (RAE) [39] as a load migration trigger when a server's power consumption threshold is predicted to be breached. We then profile all containers executing on the culprit server using a combination of psutil and perf. We then input the resulting parameters into our container power model to migrate the container with the lowest predicted power to the lowest utilized server operating within its peak energy efficiency threshold by using First-Fit Decreasing Scheduling (FFD). Balancing these containerized workloads across all servers so that each operates within their peak energy efficiency saves more power over time than consolidating most workloads to fewer servers [56]. We compare the energy savings between the most accurate analytical model, our proposed hybrid model, and a state-of-the-art machine learning model [39], which uses RAE.

For the test case, we run a combination of compute- and memory-intensive workloads. The applications are categorized into compute and memory intensive

Parameters in Original RAE	Parameters in Revised RAE
CPU utilization, input/output packets per second, number of runnable tasks, memory usage, number of disk read/write operations, and file system usage	CPU frequency, user time, CPU utilization, software interrupts, number of processes, cache miss ratio, virtual memory usage, instructions, and shared memory usage

Table 6.1: Parameters used in Original Six-parameter RAE and Revised 11-parameter RAE

as listed in Table 3.6. As described earlier, memory-intensive workloads have high MPKI as shown in Table 3.5.

6.1 State-of-the-Art Implementation

6.1.1 Recursive Autoencoder Power Model

Li et al. [39] implement both fine-grained and coarse-grained power models by utilizing a RAE. Their fine-grained design outputs an instantaneous power prediction for the current time step, t, while their coarse-grained implementation outputs an average power consumption over a time period, $T = 1$ hour. With regards to fine-grained prediction, the RAE accounts for the fast power fluctuation of workloads by encoding the system parameters starting at $t - \tau$ for a prediction at time t for every sample where τ = amount of encoded time steps for an training input X_t. Because our implemented scheduling algorithm requires instantaneous power prediction, we compare against this fine-grained design.

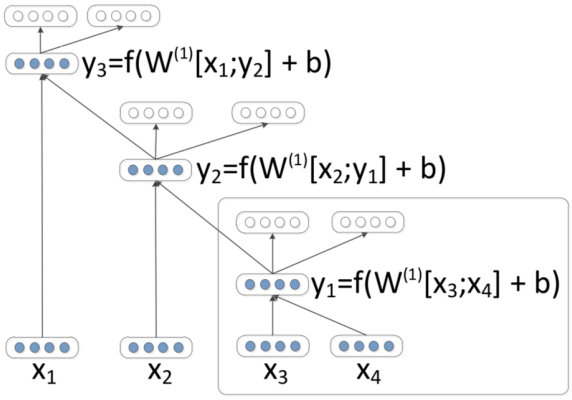

Figure 6.1: RAE Architecture [39]: Single autoencoder node is depicted in frame. Successive nodes encode the output from previous node and input at time $i \in t - \tau, t$

The original authors utilized six parameters in their RAE power model: CPU usage, input/output packets per second, number of runnable tasks, memory usage, number of disk read/write operations, and file system usage. We revised the RAE power model to use 11 parameters that are used in our hybrid power model to conduct a fair comparison. This revision doesn't impact prediction accuracy because 1) the paper did not note their motivation behind their listed parameters while we correlated our parameters with our training application's power consumption and 2) as can be seen in Table 6.1, most parameters in both cases qualitatively encapsulate the events that the original RAE power model's parameters capture. We trained a RAE power model for fine-grained prediction across 3500 samples from our training dataset for both MAE and energy savings as a load migration trigger comparisons.

6.1.2 Design Details of Recursive Autoencoder Power Model

For the RAE implementation, we used sample autoencoder code from [6] and the Tensorflow functional API to create three main submodels – an input encoder (ie), merge encoder (m), and decoder (d). More information about the functionality of these submodels are depicted in Table 6.2. We denote $\tau = 30$, batch size $(L_1) = 64$, validation size $(L_2) = 16$, and epochs = 100. We use the same optimization objective as the original RAE power model (Equation 6.1) ϵ_{RAE} as the loss function for our Adam optimizer.

$$\epsilon_{RAE} = \epsilon_{PRD} * 0.95 + \epsilon_{AE} * 0.05 \tag{6.1}$$

$$\epsilon_{PRD} = \frac{\sum^{N} |y(t) - y'(t)|^2}{N} + .0001 \cdot ||W||^2 \tag{6.2}$$

$$\epsilon_{AE} = \frac{\sum^{N} Err_{rec}(t)}{N} \tag{6.3}$$

Denote the length of the dataset as N, number of parameters as p, and τ as the number of preceding time steps we want to encode within a single training sample. We first convert our dataset into numpy arrays of size (τ, p) by concatenating datapoints $\in [\tau, N]$ with the previous τ datapoints. We use this resulting collection of 2D training samples for our RAE. We start each epoch with a randomly selected starting point, $x \in [\tau, N - (L1 + L2)]$. Each training sample has the size (τ, p), comprised of the parameters sampled at each previous τ time steps. First, we encode each of the τ inputs of size p for use in our model by using the 'ie' submodel and denote them as 'leaves' in our RAE tree. In Figure 6.2, the leaf nodes are X1, X2, X3 and X4. The output size of these leaf submodels are 5 $(p//2)$ as in the previous RAE power model and other works [51] show an encoder's output neurons are half the number of input neurons. Then,

Input Encoder (ie)	**Merge Encoder (m)**	**Decoder (d)**
Encodes the inital input neurons from 11→ 5 for use within RAE	Concatenates separate inputs of 5 neurons and encodes into 5 neuron output. Also evaluates the reconstruction error for ϵRAE calculation	Inputs the last RAE node's 5 neuron output for power prediction

Table 6.2: Details of RAE Submodels

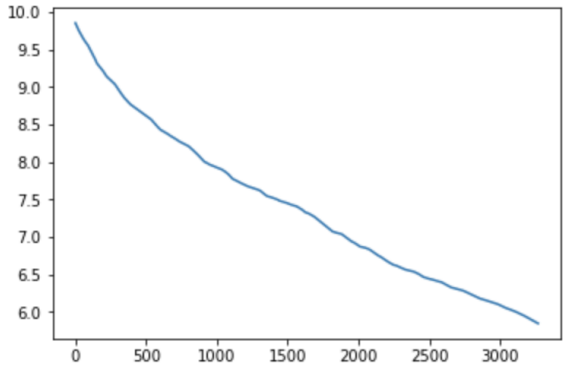

Figure 6.2: RAE loss over 3500 training samples

denoting the 0th index leaf or input at time step at $t-\tau$ as current, we iteratively combine using separate instances of the 'm' submodel. Specifically, the current node with its output are concatenated with the output of the leaf node at the next time step until no more remain as seen in Figure 6.2. The output of the last node is inputted/decoded into the 'd' submodel to yield the RAE power prediction. Through Tensorflow's gradient tape, we update the same 'ie', 'd', and $\tau - 1$ 'm' instances (there are $\tau - 1$ merges with τ inputs for every training sample) after determining the ϵ_{RAE} for every batch. To determine the ϵ_{RAE}, the reconstruction error, Err_{rec} is summed at every time step in a training sample using the output from merge encoder at t, $m(t)$. We only update the submodels parameters if there is improvement upon the validation error, summed across test samples from $[x + L_1, L_2]$. This follows the inspiration of the original RAE power model.

6.2 Analytical Power Models

To justify our analytical model selection both within our hybrid model and as a comparative approach, we evaluate three widely-used conventional power models on the prominent SPECpower dataset.

The MAE of the three conventional power models on the SPEC dataset are not negligible as errors range from 16 to 46 W. As expected, conventional power models do not perform well on workloads that utilize caches, memory hierarchy, and the scalability of shared memory processors. In particular, with respect to equation 2.2, its large error is most likely due to the assumption of the idle consumption ratio. This value often varies from platform to platform, and therefore introduces error when assumed.

When the analytical power models predict power consumption on the CPU load micro benchmark dataset, the error rate decreases substantially. Equation

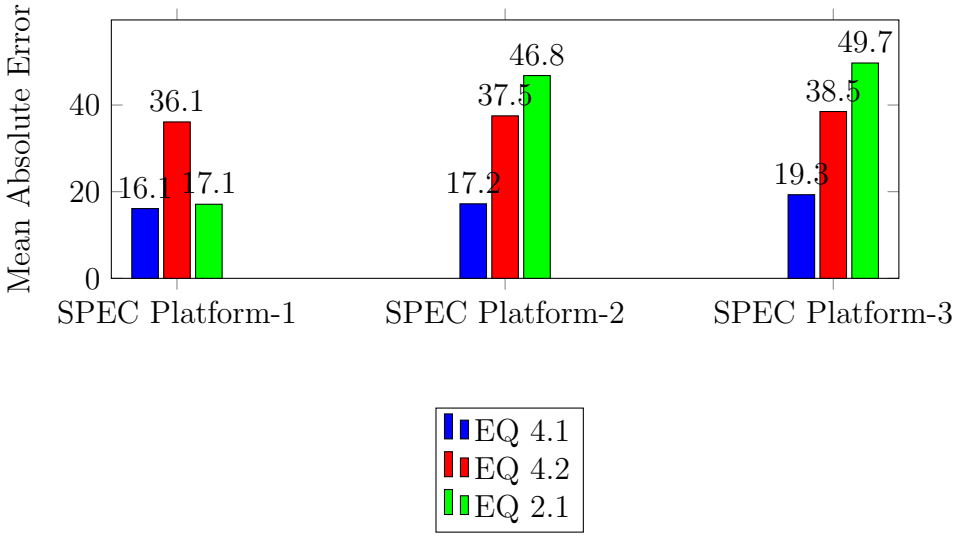

Figure 6.3: Equation 4.1, 4.2, 2.2 MAE on selected SPEC Platforms from SPECpower2008 public dataset

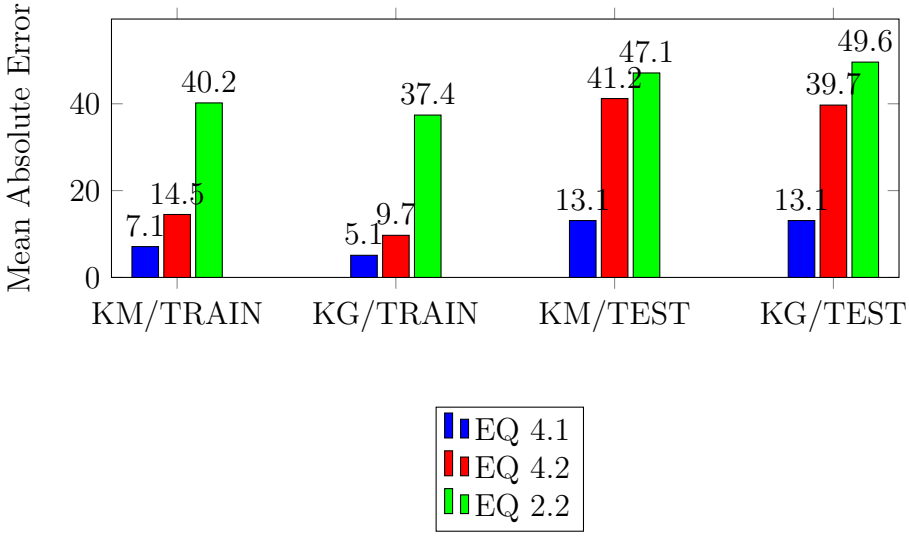

Figure 6.4: Equation 4.1, 4.2, 2.2 MAE on MoCA lab platforms with Platforms/Dataset labelled. KM refers to Kraken and Medusa while KG refers to Kimchi and Gimbap

4.1 seems to maintain its current trend of effectiveness, with only a MAE of 5 on kraken platforms. **We therefore utilize Equation 4.1 as the choice Analytical Model for comparison and integration in our hybrid power model.**

6.3 Power Model Accuracy Evaluation

We evaluate the accuracy of the different power models on our CPU and MEM datasets. We find that our hybrid power model outperforms the other two models by up to 11%. However, we must preface these results as the original RAE performed more optimally with a mean prediction error of .24. It is important to note that 1) our PARSEC and SPLASH workloads are high performance computing benchmarks, and as a result exhibit dynamic power consumption. It is possible that the dynamic WC98 webtrace the original RAE trained upon is less dynamic. And 2) the original RAE was tested on two homogeneous servers. Because we sample datapoints from 2 distinct platforms, the RAE does not standardize to account for platform differences.

6.4 Hybrid Power Model Load Migration Decisions

We assess the effectiveness and reliability of model selection decisions of our hybrid model. Figure 6.6 shows the datacenter total power consumption (power consumption of all four worker servers in MoCA Lab) drawn in blue color and the cache miss ratio at moments in time tracked with bar graphs. The purple dots show moments that the analytical model is selected to trigger migration and the red dots show those that the DNN power model is selected to trigger

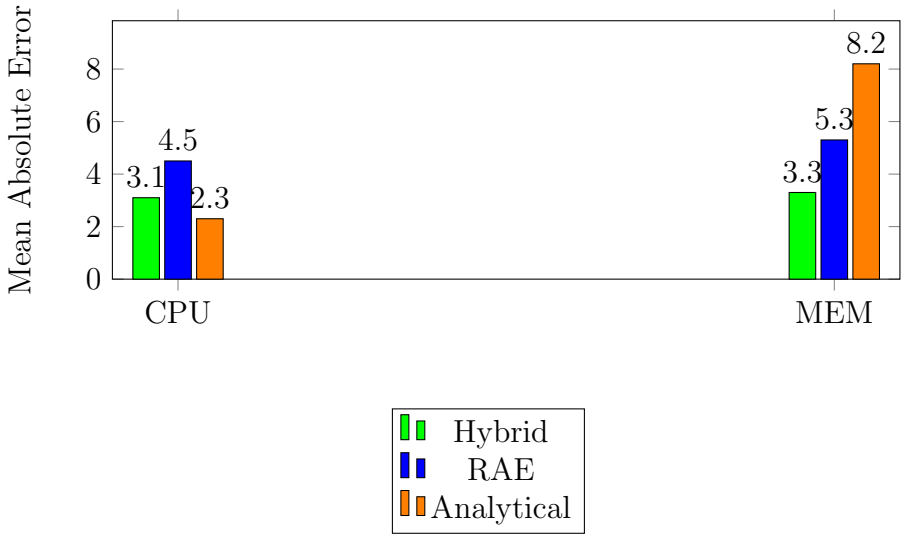

Figure 6.5: Power Model Performance on the CPU and MEM workloads

migration. Figure 6.6 shows that the classifier selects DNN model when cache miss ratio is higher than 40%, as shown in between 60 to 75 time steps that have tall miss ratio bar graphs. At the remaining time steps where cache miss ratios are mostly less than 10%, analytical model was selected. This result shows that the hybrid model correctly selects expected models (analytical model for compute-intensive workloads and DNN model for memory-intensive workloads). As far as the workloads show consistent computing pattern, the same model will be reliably selected.

6.5 Power Model Load Migration Evaluation

In order to evaluate the power models in an adequate use case, we devise an experimental workload where the GPU moderator first schedules a container application from our memory-intensive workload in Table 3.6 and migrates the lowest consuming containers whenever there is a power threshold violation. We

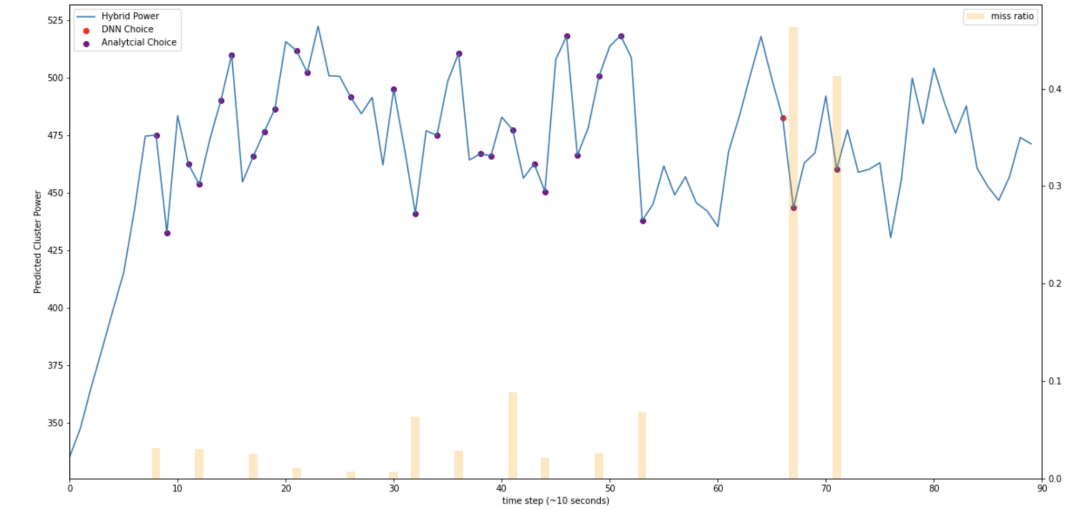

Figure 6.6: Model Selection Results: When cache miss ratio is high, DNN model is selected to trigger migration.

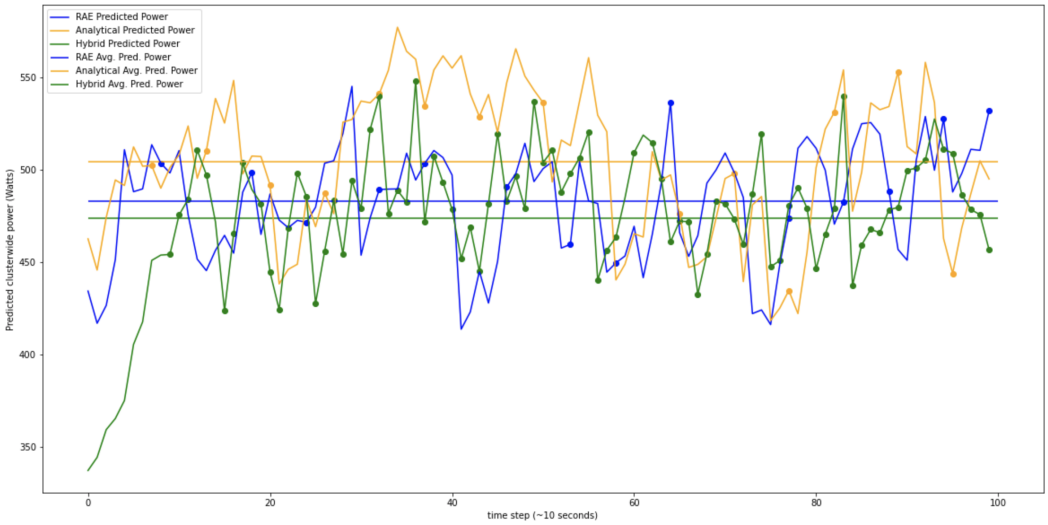

Figure 6.7: Data Center Power Consumption Comparison When Using Different Power Models: Blue graph is state-of-the-art RAE, yellow graph is Analytical model, and green graph is Hybrid Model

controlled workload execution time and injection time by following long tail distribution to mimic realistic data center execution [58]. We launched some applications concurrently to simulate realistic data center execution where container workloads generally contain up to tens of containers executing simultaneously [3, 15]. Migration points are denoted as plotted dots on the line graphs in Figure 6.7. From Figure 6.7, the proposed hybrid power model yields up to 10% and 5% overall power savings compared to the analytical and RAE power models, respectively.

Chapter 7

Conclusion

In this thesis, we propose a hybrid server power model to take advantage of low latency analytical power models and accuracy of machine learning approaches. Our hybrid server power model first classifies a set of parameters according to what it predicts will perform best: analytical model or DNN model. If the prediction eclipses the confidence threshold of 0.6, The hybrid power model outputs using the classified power model, defaulting to DNN power prediction otherwise. Our work also presents a per-container DNN power model that provides load balancing algorithms to make optimal decisions based on the predicted power cost for a container. This study additionally contributes DNN power model standardization to allow for DNN power models to better generalize across distinct platforms. Our results show a 50% increased accuracy after applying this standardization to our dataset with two distinct platforms. To evaluate our hybrid model performance, we compare against a state-of-the-art recursive auto encoder (RAE) power model and analytical power model in terms of accuracy, and as a load migration trigger. Our experiments show that our hybrid power model outperforms the analytical power model by up to 10% and the RAE power model up to 5% on a load migration use case.

In future works, we will compare our hybrid power model against other state-of-the-art techniques and workloads for continued evaluation, and determine if our standardization applied on our DNN output yields similar improvements when applied to other machine learning power models. Additionally, we will test the capacity of our standardized inputs on a greater amount of distinct platforms. We will also evaluate alternative load migration techniques for improved runtime decision making. Some possible considerations are: designing objectives to minimize that account for shared memory among containers, and deeper analysis of how to evaluate per container power consumption on fully saturated server loads.

Bibliography

[1] A deep learning framework for short-term power load forecasting. *International Conference on Advances in the Emerging Computing Technologies*.

[2] Docker, howpublished=`https://www.docker.com`, note = Accessed: 2021-01-21.

[3] docker-phronix, howpublished = `https://github.com/nicbet/docker-phoenix`, note = Accessed: 2021-02-29.

[4] Intel TurboBoost Technology 2.0, howpublished = `https://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html`, note = Accessed: 2021-01-17.

[5] Psutil, howpublished = `https://psutil.readthedocs.io/en/latest/`, note = Accessed: 2021-01-17.

[6] RAE, howpublished = `daniel-at-world.blogspot.com/2020/03/recursive-auto-encoders-in-tensorflow-20.html`, note = Accessed: 2021-05-5.

[7] RandomForestClassifier, howpublished = `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.randomforestclassifier.html`, note = Accessed: 2021-01-21.

[8] SPEC, howpublished = `http://spec.org`, note = Accessed: 2021-01-19.

[9] SPECPower2008, howpublished = `http://spec.org/power_ssj2008/results/`, note = Accessed: 2021-01-21.

[10] Stress-ng, howpublished = `https://manpages.ubuntu.com/manpages/artful/man1/stress-ng.1.html`, note = Accessed: 2021-01-21.

[11] Sysbench, howpublished = http://manpages.ubuntu.com/manpages/trusty/man1/sysbench.1.html, note = Accessed: 2021-01-21.

[12] System-load-generator, howpublished = https://github.com/pradykaushik/system-load-generator. Accessed: 2021-01-21.

[13] Talos Hyperparam serach, howpublished = https://pypi.org/project/talos/, note = Accessed: 2021-01-21.

[14] Vernier Wattsup Pro Power Meter, howpublished = https://www.vernier.com, note = Accessed: 2021-01-21.

[15] vm-docker-bench, howpublished = https://github.com/dockerdemos/vm-docker-bench, note = Accessed: 2021-02-29.

[16] Why Energy Is a Big and Rapidly Growing Problem for Data Centers.

[17] M. Abu-Tair, M. I. Biswas, P. Morrow, S. McClean, B. Scotney, and G. Parr. Quality of service scheme for intra/inter-data center communications. In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pages 850–856, 2017.

[18] K. M. U. Ahmed, J. Sutaria, M. H. J. Bollen, and S. K. Rönnberg. Electrical energy consumption model of internal components in data centers. In *2019 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe)*, pages 1–5, 2019.

[19] Jatin Bedi and Durga Toshniwal. Deep learning framework to forecast electricity demand. *Applied Energy*, 238:1312–13265, 2019.

[20] J. Bhimani, Z. Yang, M. Leeser, and N. Mi. Accelerating big data applications using lightweight virtualization framework on enterprise cloud. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, 2017.

[21] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[22] M. T. Chung, N. Quang-Hung, M. Nguyen, and N. Thoai. Using docker in high performance computing applications. In *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*, pages 52–57, 2016.

[23] Jürgen Cito, Philipp Leitner, Thomas Fritz, and Harald C. Gall. The making of cloud applications: An empirical study on software development for the cloud. ESEC/FSE 2015, page 393–403, New York, NY, USA, 2015. Association for Computing Machinery.

[24] C. Kozyrakis P. Ranganathan D. Economou, S. Rivoire. Full-system power analysis and modeling for server environments. *Workshop on Modeling, Benchmarking, and Simulation*.

[25] M. Dayarathna, Y. Wen, and R. Fan. Data center energy consumption modeling: A survey. *IEEE Communications Surveys Tutorials*, 18(1):732–794, 2016.

[26] Christina Delimitrou and Christos Kozyrakis. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, page 77–88, New York, NY, USA, 2013. Association for Computing Machinery.

[27] Fahimeh Farahnakian, Pasi Liljeberg, and Juha Plosila. Lircup: Linear regression based cpu usage prediction algorithm for live migration of virtual machines in data centers. In *Proceedings of the 2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, SEAA '13, page 357–364, USA, 2013. IEEE Computer Society.

[28] Rong Ge, Xizhou Feng, and Kirk W. Cameron. Modeling and Evaluating Energy-Performance Efficiency of Parallel Processing on Multicore based Power Aware Systems. *International Symposium on Parallel Distributed Processing*, 2009.

[29] S. Greenberg, E. Mills, Bill Tschudi, and Peter Rumsey. Best practices for data centers: Lessons learned from benchmarking 22 data centers, *2006 ACEEE Summer Study on Energy Efficiency in Buildings*. 2006.

[30] G. Han, Wenhui Que, G. Jia, and L. Shu. An efficient virtual machine consolidation scheme for multimedia cloud computing. *Sensors (Basel, Switzerland)*, 16, 2016.

[31] Ying-Feng Hsu, Hayato Kuwahara, Kazuhiro Matsuda, and Morito Matsuoka. Toward a workload allocation optimizer for power saving in data centers. In *IEEE International Conference on Cloud Engineering*.

[32] I. Jimenez, C. Maltzahn, A. Moody, K. Mohror, J. Lofstead, R. Arpaci-Dusseau, and A. Arpaci-Dusseau. The role of container technology in reproducible computer systems research. In *2015 IEEE International Conference on Cloud Engineering*, pages 379–385, 2015.

[33] Kashif Nizam Khan, Mikael Hirki, Tapio Niemi, Jukka K. Nurminen, and Zhonghong Ou. Rapl in action: Experiences in using rapl for power measurements. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 3(2), March 2018.

[34] A. J. KleinOsowski and D. J. Lilja. Minnespec: A new spec benchmark workload for simulation-based computer architecture research. *IEEE Computer Architecture Letters*, 1(1):7–7, 2002.

[35] D. Kliazovich, P. Bouvry, Y. Audzevich, and S. U. Khan. Greencloud: A packet-level simulator of energy-aware cloud computing data centers. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pages 1–5, 2010.

[36] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. Singularity: Scientific containers for mobility of compute. *PLoS ONE*, 12(5), 5 2017.

[37] C. Laaber and P. Leitner. An evaluation of open-source software microbenchmark suites for continuous performance assessment. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pages 119–130, 2018.

[38] Dong Li, Rong Ge, and Kirk Cameron. System-level, Unified In-band and Out-of-band Dynamic Thermal Control. *International Conference on Parallel Processing*, 2010.

[39] Yuanlong Li, Han Hu, Yonggang Wen, and Jun Zhang. Learning-based power prediction for data centre operations via deep neural networks. In *International Workshop on Energy EFficient Data Centers*.

[40] Li Shang, Li-Shiuan Peh, and N. K. Jha. Dynamic voltage scaling with links for power optimization of interconnection networks. In *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.*, pages 91–102, 2003.

[41] Weiwei Lin, Guangxin Wu, Xinyang Wang, and Keqin Li. An artificial neural network approach to power consumption model construction for servers in cloud data centers. In *IEEE Transactions on Sustainable Computing*.

[42] Liang Luo, Wenjun Wu, W.T. Tsai, Dichen Di, and Fei Zhang. Simulation of power consumption of cloud data centers. *Simulation Modelling Practice and Theory*, 39:152–171, 2013. S.I.Energy efficiency in grids and clouds.

[43] Antonios T. Makaratzis, Konstantinos M. Giannoutakis, and Dimitrios Tzovaras. Energy modeling in cloud simulation frameworks. *Journal of Future Generation Computer Systems*, 79:715–725, 2018.

[44] John C. McCullough, Yuvraj Agarwal, Jaideep Chandrashekar, Sathyanarayan Kuppuswamy, Alex C. Snoeren, and Rajesh K. Gupta. Evaluating the effectiveness of model-based power characterization. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'11, page 12, USA, 2011. USENIX Association.

[45] R. Morabito. A performance evaluation of container technologies on internet of things devices. In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 999–1000, 2016.

[46] René Peinl. and Florian Holzschuher. The docker ecosystem needs consolidation. In *Proceedings of the 5th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,*, pages 535–542. INSTICC, SciTePress, 2015.

[47] Swapnoneel Roy, Atri Rudra, and Akshat Verma. An Energy Complexity Model for Algorithms. *Conference on Innovations in Theoretical Computer Science*, 2013.

[48] C. Sakalis, C. Leonardsson, S. Kaxiras, and A. Ros. Splash-3: A properly synchronized benchmark suite for contemporary research. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 101–111, 2016.

[49] Ziyu Shen, Xusheng Zhang, Binghui Liu, Bin Xia, Zheng Liu, Yun Li, and Saiqin Long. Pcp-2lstm: Two stacked lstm-based prediction model for power consumption in data centers. In *Seventh International Conference on Advanced Cloud and Big Data*.

[50] Ziyu Shen, Xusheng Zhang, Bin Xia, Zheng Liu, and Yun Li. Multi-granularity power prediction for data center operations via long short-term memory network. In *IEEE International Conference on Parallel & Distributed Processing with Applications*.

[51] Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 151–161, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics.

[52] Bogdan Marius Tudor and Yong Meng Teo. On Understanding the Energy Consumption of ARM-based Multicore Servers. *International Conference on Measurement and Modeling of Computer Systems*, 2013.

[53] T. Veni and S. M. Bhanu. Prediction model for virtual machine power consumption in cloud environments. *Procedia Computer Science*, 87:122–127, 2016.

[54] Reinhold Weicker. On the use of spec benchmarks in computer architecture research. *SIGARCH Computer Architecture News*, 25(1):19–22, March 1997.

[55] Sage Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI '06)*, November 2006.

[56] Daniel Wong. Peak efficiency aware scheduling for highly energy proportional servers. In *International Symposium on Computer Architecture*, 2016.

[57] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. De Rose. Performance evaluation of container-based virtualization for high performance computing environments. In *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 233–240, 2013.

[58] Yunjing Xu, Zachary Musgrave, Brian Noble, and Michael Bailey. Bobtail: Avoiding long tails in the cloud. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, page 329–342, USA, 2013. USENIX Association.

[59] A. J. Younge, K. Pedretti, R. E. Grant, and R. Brightwell. A tale of two systems: Using containers to deploy hpc applications on supercomputers and clouds. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 74–81, 2017.

[60] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, and W. Zhou. A comparative study of containers and virtual machines in big data environment. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 178–185, 2018.