Modelling the Power Cost of Application Software Running on Servers

by

Omar Alghamdi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Computer Engineering

Waterloo, Ontario, Canada, 2015

Abstract

One of the most important aspects of managing data centres is controlling the power consumption of applications running on servers. Developers, in particular, should evaluate each of their applications from a power consumption point of view. One can conduct an evaluation by creating models that predict power usage while running applications on servers. For this purpose, this study creates a non-exclusive test bench that can collect data on subsystem utilization by using a performance counter tool. Based on the selected subsystem performance, various models have been created to estimate the power consumption of applications running on servers.

The author's models are created based on collecting the performance on four subsystems (i.e. the CPU, Memory, Disk and Interface) by*Collectd* tool, and the actual power consumption of a machine using a TED5000 power meter. These subsystems have been chosen because they are the components of the server that consume the most power. In addition, as the experiments in this study demonstrate, using these subsystems as the model's input is the most efficient selection across different hardware platforms. The accuracy of the models is affected by the model inputs selection. Creating the model requires several steps: (i) connect the power meter to the server and install all the required packages such as *Collectd*; (ii) perform workloads on the selected subsystems; (iii) collect and simplify the data (subsystems counters and actual power) that has been stored during performing the workloads; and (iv) train the data by a modelling technique in order to create the model.

This work has seven dimensions; (i) collection of the performance counters and the actual power consumption of a system, and simplification of the collected data; (ii) introduction of a simple test bench for modelling and estimation of the power consumption of an application; (iii) introduction of two modelling techniques: Neural Network and Linear Regression; (iv) design of two types of workloads; (v) use of three real servers with different configurations; (vi) use of four scenarios to validate the models; (vii) proof of the importance of the subsystems selection; and (viii) automation of the test bench. With these models, power meter devices will no longer be necessary in measuring power consumption. Instead, the models can be used to predict power consumption. Generally, Neural Network models have fewer errors than Linear Regression models, and all the models (Neural Network or Linear Regression) perform better with long time workload design.

Acknowledgements

I would like to thank my supervisor Professor Sagar Naik for all his support, King Abdullah Foreign Scholarship Program, and all others who made this possible.

Dedication

To my beloved family; my parents, Ali Alghamdi, Rahmah Saeed, my wife Fatima Aloryni and my son Azzam.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Electrical energy is fundamental to all computer systems [1],[2]. However, it is not ubiquitous, and it is expensive. Reducing the power consumption in data centres is an important issue that researchers are now focusing on [3]. Information and Communication Technology as of 2009 consumed about 8% of the worlds electricity costs [4]. Between 2000 and 2005, data centres (servers, communication, cooling infrastructure and storage) power consumption increased by 115% [3]. Those in the business of operating high-volume data centres, servers, and bitcoin mining farms all collectively have an interest in understanding how their systems' resources are utilized.

Many strategies have been devised to save power in data centres, and have been surveyed in [5] and [6]. However, the main challenge in power-saving is the tradeoff between network performance and power-saving effects [4]. Often, however, fitting vast numbers of components with thermal detection instrumentation is not economically feasible. In addition, research has been done on reducing the power cost in data centres. The authors In reference [7], identify two categories of strategies for reducing the power cost in data centres. The first category is dependent on reducing the power cost through hardware design and engineering, and includes designing efficient chips, cooling systems, and power supplies. The second category can further be divided into three levels at existing data centres: first, at the server level, the idea is to control the CPU speed of a single server [8]; second, at the data centre level, the idea is to control dynamically the number of servers in the data centre [9]; and third, at the interdata centre level, the workload in the distributed data center should be routed to places that have a lower cost [10].

Reducing power consumption in computational processes is also important to software developers [11], [12], [13]. Ideally, a tremendous amount of software design goes into considerations that are critical to the power efficiencies of computer systems. Sometimes, software is designed by a high-level developer unaware of underlying physical components of the system architecture which can be exploited. Furthermore, even if a developer is aware, they design software geared towards mass end-user adoption and thus go for cross-compatibility. The challenge for the software designer is to utilize dynamic hardware adaptations. Dynamic hardware adaptations make it possible to reduce power consumption and overall chip temperature by reducing the level of available performance. However, these adaptations generally rely on inputs from temperature sensors, and because of the thermal inertia in microprocessor packaging [14], [15], [16], [17], the detection of temperature changes lag behind the power events that cause them.

A work-around to dynamically gauge a system's performance is to use performance counters that have been demonstrated to effectively proxy power consumption [18], [19]. Performance counters count the instances of local processor events and calls, and thus eliminate the need for sensors distributed about various parts of the system. However, considerable modelling challenges exist in relating the statistical count information to what is truly happening internally with respect to power consumption. Consequently, there is considerable financial interest in developing accurate models that use the performance counts to cost-effectively provide up-to-date power consumption information. With customization, engineers can target and customize specific aspects of system feedback in response to stress tests.

In this work, we use a non-exclusive test bench that has been introduced in our previous work [20] for modelling the power consumption of a server. Four main subsystems are used in this experiment: the CPU, Memory, Disk and Interface. Two modelling techniques (Neural Network and Linear Regression) are used for creating models. Most of the models available in the literature on power estimation employ Linear Regression to fit model parameters [3]. Two workload designs are also introduced, to compare the accuracy and the reflection of the workloads on the models. The models are applied to three real servers to show the accuracy across different hardware platforms. Four scenarios of real load have been used to validate the models.

The four subsystems (CPU, Memory, Disk and Interface) have been chosen because they are the most important components in a machine. Our experiments show that the CPU is the main component responsible for increasing the total power consumption in a machine. Thus, the CPU and the power consumption are directly related. On the other hand, the memory has a direct affect on the CPU. In some experiments, when the memory is stressed to only about 30%, the CPU will be loaded to 100%. There is also a relationship between the disk and the memory. While loading the disk, the component most affected is the memory. In fact, in our experiments we loaded each component to reach 100%, except the disk, which was not loaded until the memory reached 100%. Loading the memory over 100% is likely to cause a system crash.

The importance of the disk in data centres is the fact that most are used for data storage. Moreover, the interface in data centres is very important because of the use of the Internet. The interface load also has a direct effect on the memory, but this effect is not as strong as the effect on the disk. An experiment was conducted to discover the importance of using the four subsystems for modelling. A Neural Network model was created that is based only on the CPU counter and validated by the same load that validates a model based on the four subsystems. It was found that the model that is based on only the CPU has 24.14% error (Mean Absolute Error of energy), and the model based on the four subsystems has 0.85% MAEe. Hence, using the four subsystem counters for modelling is very important in achieving accuracy.

1.1 Problem Statement and Solution

Problem Statment:

Data centres are one of the fastest growing consumers of electricity. For example, in the United States in 2013 data centres consumed an estimated 91 billion kilowatt-hours [21]. The growth amount of data and applications also contribution to the growing power consumption. All online activities including email, social media, and conducting business, are delivered through an estimated nearly three million data centres across America [22].

The design of applications running on devices has an impact on power consumption [23]. Hence, the problem is the power consumption of data centers that is expanding with the growth of data around the world. With this growth of data there is also a growth of applications, which also plays an important role in power consumption.

Solution:

Developers should evaluate their applications from a power consumption point of view. One can conduct an evaluation by creating models that predict power usage while running applications on servers. For this purpose, this study have created a non-exclusive test bench that can collect subsystem utilization by using a performance counter tool and a power meter to collect the actual power consumption. Developers are thus given the ability to create models that can predict the power consumption of an application running on a server, without the need of a power meter.

1.2 Main Contributions

This work makes the following contributions:

1. Designs a workload for creating accurate models that estimate the power profile of an application software during its developmental stage. As the experiments suggest, the main component of creating the model is the workload. For each platform, a separate estimation model should be created. A workload designed for creating a model for a platform may provide a good estimation model, although this does not mean the same workload design would provide the same accuracy on a different platform.

2. Use two modelling techniques–Neural Network and Linear Regression–to prove accuracy comparatively.

3. Validates the importance of using the four subsystems (the CPU, Memory, Disk and Interface) counters for creating the models. Experiments have been conducted that show the need for these four subsystems to be used in models in order to obtain better accuracy.

4. Automates our test bench for measuring the actual power, loading the selected subsystems, simplifying the data, creating models and estimating their power. Automating our test bench is very important for developers, for many reasons, such as saving time, ease of use and reducing complexity.

1.3 Summary of the Results

After creating the models by using Neural Network and Linear Regression techniques for three real servers, the models were evaluated with four scenarios. The models have been created based on two workload designs on each server: models that are based on Workload A (short time workload) and models that are based on Workload B (long time workload). Generally, it was found that Neural Network technique is better than the Linear Regression technique. Also, both modelling techniques generate better results with workload B. However, after performing some experiments on the selection of the subsystems for modelling, CPU, Memory, Disk and interface are the better model inputs. Even though selecting different inputs may give better results in some cases, these four subsystems are more accurate across different platforms and across various applications.

1.4 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 presents a comprehensive literature review of power consumption modelling and estimation. Chapter 3 explains all the tools that have been used for collecting the system performance and the power meter configuration. It also shows how to simplify the data before creating the models. Chapter 4 introduces our test bench that has been used for modelling purposes. This chapter also shows the flowchart of the modelling process. Tow modelling techniques and workload design are explained in this chapter. Chapter 5 demonstrates the importance of selecting the subsystems counters (model inputs). Chapter 6 introduces the implementation of the test bench. Finally, Chapter 7 presents some concluding remarks, limitations of the thesis and future work.

Chapter 2

Literature Review

Research similar to the present study has been conducted. Some papers used the performance counter information to create models to estimate the power consumption [24], [25],[26]. In others, the focus was on the total power consumption of a system [17],[19],[27]. The authors of reference [17] use a simulator called SoftWatt that has been introduced in [28] to estimate the power consumption. The SoftWatt simulator is based on three components of the subsystem, namely: the CPU, memory, and disk. However, reference [19] focuses on the CPU; they measured the power for each core of the CPU to create their models. Also, in reference [27] the authors predicted the total power consumption of a machine depending on the CPU counter only. In reference [18] the authors use the microprocessor performance counters to measure the power consumption of a system by using two techniques of modelling: Linear Regression and polynomial regression. Furthermore, in reference [29] an automated test bench was introduced to measure the power consumption.

2.1 Models for Estimating the Power Consumption of the Entire Server

The works that have been done for modelling the power consumption based on the performance counters can be divided into three categories. This section discusses the first category, which is models for estimating the power consumption of the entire server for examples the references [30], [31], [32], [33], [34], and [35]. All these models include the CPU utilization as part of the model inputs given that the CPU consumes the most power among all the components in a machine. Our experiments show that the CPU utilization alone is not enough for creating an accurate model. The authors of reference [33] and [32] use only the CPU utilization for the model inputs. Also, Vinicius et al. [35] used the CPU utilization and the CPU frequency for the inputs in their model. Moreover, the Linear Regression technique for training has been used in the references [30], [33], and [34].

Zahra et al. [33] aim to improve the selection of active servers using thermal awareness. In order to do that, first they created a model based on the CPU utilization to estimate the total power consumption of a server. Also, linear regression was used for training the data. Second, they estimated the dynamic workload by using two Kalman filters, which estimated the average arrival rate of HTTP requests. Then the authors proposed a two-tier approach of thermally aware workload placement. Their approach of saving energy is comprised of three factors: (i) turn off unnecessary servers, (ii) select the most thermal-efficient server, and (iii) direct the workload toward the thermally efficient server. Moreover, for the evaluation they used SPECweb2009 test bench.

After Xiaobo et al. [32] calculated the power consumption of some subsystem in a server, such as memory, disk, and motherboard, they selected the CPU as the main input of their model. The CPU utilizations of different servers are collected together to estimate their power. Three workloads have been selected, which are Websearch, Webmail and Mapreduce. The authors then introduced two techniques of saving energy in data centres, scaling the CPU voltage/frequency and improving the non-peak power efficiency.

Dimitris et al. [31] introduced a method for modelling full-system power consumption and provided real-time power predication. They considered the CPU, Memory, Disk I/O rate and interface I/O rate utilizations as the model inputs. For collecting the utilization, the SAR [36] command in Linux was used. While collecting the subsystem utilization they stressed each subsystem in order to derive the basic correlation between the subsystem and the power consumption. Then the creation of the model was performed by linear programming. Also, in the evaluation stage, several benchmarks were used, which are SPECcpu2000, SPECjbb2000, and SPECweb2005.

For model inputs Taliver et al. [30] selected CPU utilization, Network bandwidth, disk bandwidth, and number of open sockets. Linear regression was used for training the data and micro benchmark used for validation. Here, a cooperative Web server was designed for a heterogeneous cluster that uses optimization and modelling in order to minimize the energy consumption per request.

Suzanne et al. [34], used Mantis [31] to create their model for a system. Mantis stresses each of the major subsystems before the system model is created. This work introduced five different types of models. For example, they created models based on the constant power, CPU utilization, and CPU with Disk utilization. The linear regression modelling technique was used for training the data. However, for evaluation of their models, four benchmarks were used: SPEC CPU suites, SPEC JBB, Stream and I/O-intensive benchmarks.

2.2 Models Created in a Virtual Machine

The second category of modelling the power consumption based on the performance counters includes models that are created in a virtual machine [37], [38], [39], [40], [41]. These references use the CPU as part of their models inputs. The authors of references [38], [40], and [37] used the Linear Regression technique for training. Christoph et al. [3] provide two advantages of using virtual machine(VMs). Multiple VMs can be run on the same machine in parallel. VMs can also be moved from one machine to another without stopping the execution.

Gaurav et al. [39] created a virtual system in a physical machine which can have multiple VMs, each VM with its own OS. The power was measured when the machine was idle and when it was running different workloads (active power). The workloads were chosen from the CPU2000 suite and the VM was run at different CPU utilization levels. To ensure that the active power would not be affected by the workloads, they made the fan speed of the CPU constant. The model inputs in this work were Instruction Per Cycle, Memory Access Per cycle, Cache Transactions Per Cycle, and the CPU utilizations. For training the data, the Gaussian mixture model was used. After training the data the authors connected the VMs to the server where the model is located. After that, they can run any application on a VM and perform online predication for the power consumption.

Bohra et al. [38] used the model inputs CPU, Cache, Disk, and DRAM. This model can predict the power consumption of either a VM or of the full system. They created two linear regression models: the first model is based on CPU and Cache; the second model is based on DRAM and Disk. Each model can predict the power using highly correlated system events. The total power consumption can be predicted by the sum of the two models. Some benchmark programs such as NAS-NPB, Iozone, Bonnie++, and Cachebech have validated their work.

Bhavani et al. [42] created models based on CPU and Memory. The CPU model includes the processor power and the cache power. To measure the actual power, they used WattsUp power meter. Also, for the evaluation they used PEC2006 benchmarks. For the training they used the correlation analysis technique.

Anton and Rajkumar [37] created model based on the CPU since it has a dynamic voltage and frequency scaling relationship with the power consumption. In the workload stage they used data provided as a part of the CoMon project. The data were collected every five minutes for nine days. This work used linear interpolation for the modelling purpose.

Aman et al. [40] proposed a mechanism for VM power metering named Joulemeter. The main inputs of the model were the CPU, Memory, and Disk. These subsystems were selected because they are the components that consume the most power in a machine. They showed that the CPU consumes about 58%, Memory consumes about 28%, and Disk consumes about 14%. Linear interpolation was used as the training technique. For the evaluation, the SPEC CPU2006 suite and IOmeter benchmarks have been used.

The model inputs for Qian et al. [41] was automatically selected to consider the CPU, Memory, Disk, and the Network counters. Kernelized canonical correlation analysis was used for modelling. They also used the SPEC2006 test bench for evaluation.

2.3 Models for Estimating the Power Consumption of the CPU

The third category of modelling the power consumption based on the performance counters which consists of models that are only created for the CPU to estimate power consumption as in references [16], [15], [43], [44], [45], [46]. These papers all use the Linear Regression technique for training the data.

Daniel et al. [45] used a power meter called ZES LMG450 for measuring the actual power and a collector for storing the data along with the timestamp called Dataheap Server. Unlike the test bench SPECPower introduced in [47], the authors claim that their approach does not require a special network setup for communication with other hosts. In addition, as the power consumption of the system's fan depends on the temperature [48], their work does not allow it to inhibit these effects. They perform the workload on the CPU unit by unit where they stressed some units and kept the other units constant. They measured the idle power consumption of two CPUs in different statuses such as low and high frequency. The authors claim that their results can be used, for example, to measure the cost of data transfer from different locations in the memory into the processor registers.

Ramon et al. [46] presented models based on performance monitoring counters (PMC). They used perfmon2[49] as their power meter to measure the power consumption where they modified the perfmon2 to access the PMC readings. Also, they used the test bench SPECPower [47] and LM-BENCH [50] for evaluation, and Linear Regression technique for training the data. The main issue in this work is that the model is restricted to the processor and memory that they used.

Karan et al. [19] estimated the power consumption of a processor using the performance counter data on real hardware. They had written their own benchmarks so that their model is independent of their test bench. Their test bench was designed to stress the four counters selected, which are: floating point (FP), Memory, Stalls and Instructions retired. Also, for the modelling purpose they used multiple linear regression. For evaluating their work they used the following benchmarks SPEC2006[51], OMP[52], and NAS [53]. The challenge is when estimating the power consumption per-core since some resources are shared across cores such as cashes.

Xi et al. [44] introduced an online estimation of the power consumption in a multi-programmed and multi-core environment of interacting processes. Their approach does not need any modifying of the hardware or the applications. Their models used the reuse distance histograms, cache access frequencies and the relationship between the output and cache miss rate of each process for predicting the output. Also, the linear regression technique has been used for their work and the SPEC CPU2000 used for evaluation.

Canturk et al. [16] measured the total power of the CPU. They used an ammeter connected to the main line that feeds the CPU (12v) by its clamp. Also, they used SPEC2000 for the evaluation phase and piecewise linear regression for modelling.

Van et al. [43] used instructions retired and cache access for the model inputs. Piecewise linear regression was used for training the data. For the test bench, they used GenIDLEST[54].

Lloyd et al. [15] presented linear regression models for power consumption in a superscalar microprocessor. The SPEC2000 workloads are used in this work. For the model input the authors considered and calculated the coefficients of twenty-three observed performance monitoring counters (PMCs). Their model can be used to address the critical issues of low energy efficiency or high power consumption. They found that the number of instructions retired per cycle metrics was the most representative and uops fetched per cycle was most accurate.

In our previous work [20], only one workload design was created, which is similar to Workload A in this work (explained in Section 4.2). Also, we used only one modelling technique which is the Neural Network.

Our work is different in five distinct ways: (i) the focus on the designing of the workload because it plays an important role in the accuracy of the models; (ii) the use of four counters for creating the models, which are the CPU, Memory, Disk, and Interface; (iii) the use of two techniques of modelling (Neural Network and Linear Regression) that give the developers the ability to choose between them depending on the accuracy they want and the design of the workload; (iv) the use of four scenarios as real load to validate each modelling technique in each server; (v) the implementation of the test bench in Java and Matlab, where we created two Graphical User Interface (GUI) that can load the system, collect utilizations, simplify the data, create models and estimate the power consumption of applications running on servers.

Chapter 3

Data Collection and Simplification

In this chapter the collection and simplification of the data is explained in details. The first section is explaining the *Collectd* tool that has been used to collect the system performance. Since Collected is collecting the data for each subsystem in details, simplification of the data is required. We are using TED500 as a power meter for our experiments. We are explaining how TED works and how to connect it.

3.1 Collectd

Collectd is software that collects system performance, and works as a daemon (program that works in the background without direct control from the user). There are different ways to store the data by using *Collectd*; one way is to store the data in RRD files (a system that provides for time series data logging and graphing) and the other is to store the data in CSV format. The last update for *Collectd* was in 2014 (Version 5.4.1).

3.1.1 Collectd's Architecture and Features

The main features of *Collectd* are:

1. Modularity/portability: the daemon works on Linux, Mac OS X, NetBSD...etc. Also, it is supported for Windows by SSC Serv.

2. Reasonable defaults: it is easy to use and there is no need to configure anything, but customizing the daemon is possible.

3. High-resolution statistics: when a new value is logged, there is no need to start up a heavy interpreter every time because the daemon stays in the memory.

4. There are other features in *Collectd* such as the custom extensions, it is built to scale, and supports SNMP.

The structure of Collected is as shown in (Figure 3.1).

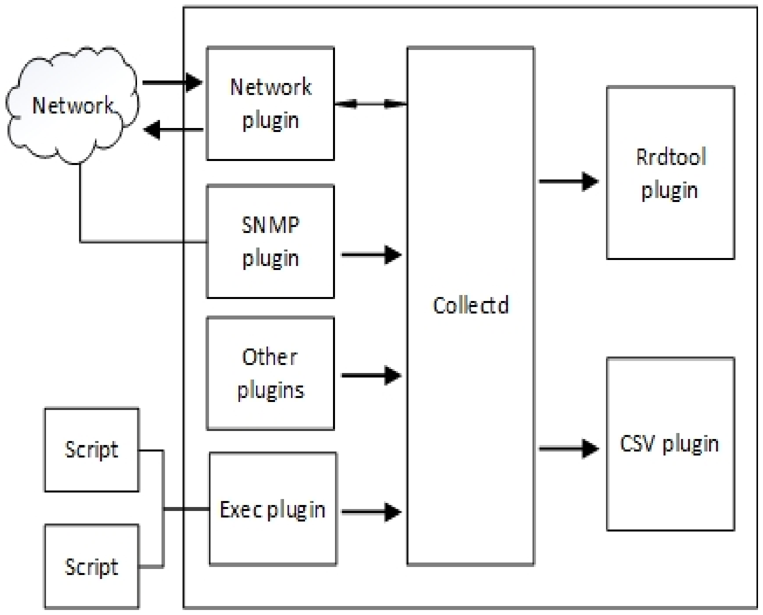


Figure 3.1: Collectd's architecture

1. SNMP: Simple Network Management Protocol that supports routers, switches, servers...etc

2. RRDtool:(round-robin database tool) a tool that handles time series data like temperatures or CPU load. The data are stored in a round-robin database (circular buffer). Moreover, it contains a tool that extracts RRD data into graphical format.

3.1.2 Using Collectd to Display a Machine's Performance

Requirements tools:

There are many ways to use the *Collectd* tool to display the performance of a machine. One of these ways is explained in this section. In Figure 3.2 the main tools used are shown.

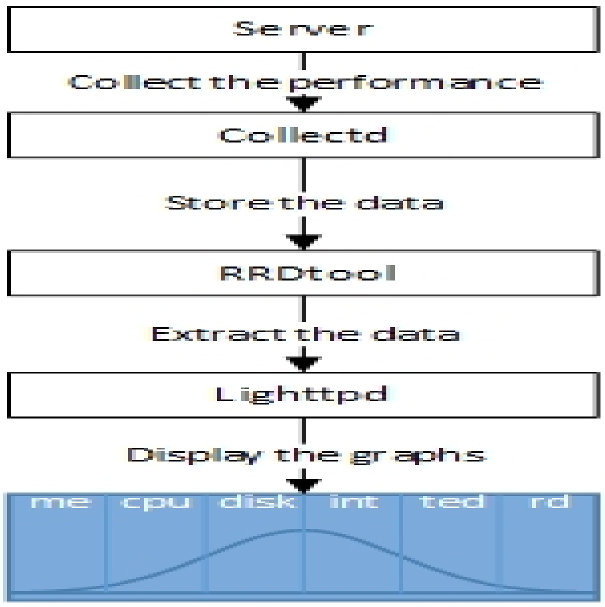
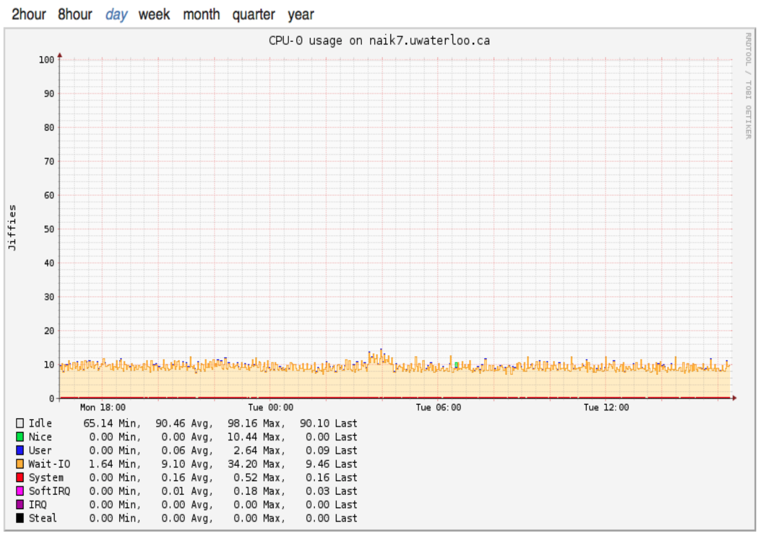


Figure 3.2: Using Collectd to display the performance

As mentioned, *Collectd* is a software tool that collects the performance data from the system. There are many plugins that can be enabled depending on what is needed to be displayed, such as CPU, Load, Memory...etc. Enabling or disabling the plugins easily can change via collectd.conf. collectd.conf contains all the plugins that can be used to enable one of the plugins; simply remove the # sign from the beginning of the plugin name.
RRdtool is where the data are stored. This tool is ideal for this kind of data because it is time series data. However, RRDtool is not the only choice.
Now one must use the data that is located in RRD. There are many front-end tools that provide the displaying of the data as graphs such as "Collectd Graph Z", "EcoStats", "Collection3"...etc. Collectd Graph Panel (CGP) was chosen as the front end tool. Some packages are required along with CGP tool such as "php5" and "php5-cgi".
Another tool needed to display the data is a web server tool. Here, "lighttpd" is used as the web server tool.
Now the graphs can be browsed by typing in a web browser http://IPaddress/cgp/.



```
□ Idle      65.14 Min,  90.46 Avg,  98.16 Max,  90.10 Last
🟩 Nice       0.00 Min,   0.00 Avg,  10.44 Max,   0.00 Last
🟦 User       0.00 Min,   0.06 Avg,   2.64 Max,   0.09 Last
🟧 Wait-IO    1.64 Min,   9.10 Avg,  34.20 Max,   9.46 Last
🟥 System     0.00 Min,   0.16 Avg,   0.52 Max,   0.16 Last
🟪 SoftIRQ    0.00 Min,   0.01 Avg,   0.18 Max,   0.03 Last
🟪 IRQ        0.00 Min,   0.00 Avg,   0.00 Max,   0.00 Last
■ Steal      0.00 Min,   0.00 Avg,   0.00 Max,   0.00 Last
```

Figure 3.3: CPU plugin graph

CPU plugin

The first plugin is CPU, which shows the amount of time spent by the cpu in various states [55]. The graph for this plugin is shown in Figure 3.3. The required time can be selected from 2 hour, 8 hour, day, week, month, quarter, or year. The figure also shows that for cpu-0; that is because there are 8 CPUs. This system has 8 cores which concedered as 8 CPUs from Collectd point of view.
In the graph, the X axis is the time that can be controled, and the Y axis is the jiffies/Hz (number of ticks). Moreover, in the bottom of the graph there is an explanation for each colour on the graph; for example, blue colour for user. Also, it shows the minimum, maximum and average.

Memory plugin

The graph of the memory plugin is shown in Figure 3.4. The X axis is the time, and the Y axis is the number of bytes of the physical memory. Since there is 4 Gb of RAM for this machine, the 4 Gb will be divided into free, buffered, cached, and used. For every part of the memory, there are different colours. For example, the green colour is used for the free memory.

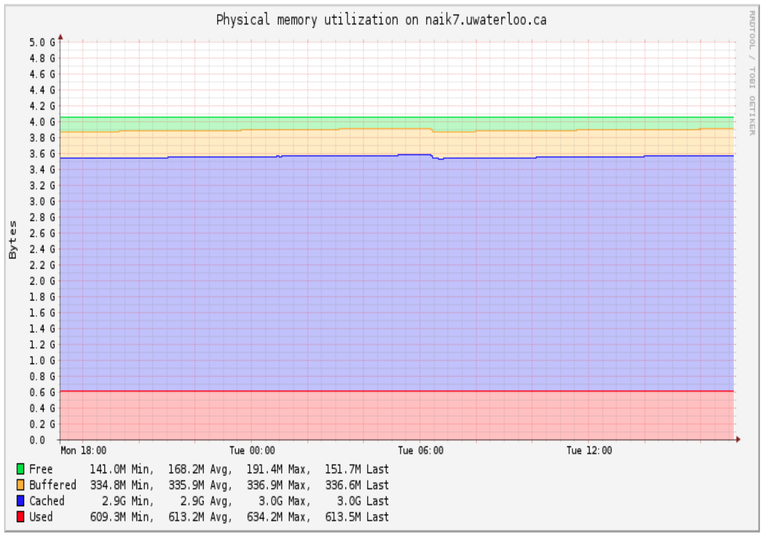


Figure 3.4: Memory plugin graph

Disk plugin

Disk plugin collects the performance of the statistics of hard-disks. As shown in Figure 3.5, there are four graphs. The first one shows the disk merged operations/second, which is the number of operations that can be merged into the other. In this graph, the higher the number the better. The next one shows the disk traffic (bytes/second). The third one covers the disk operations (ops/second). Last but not least is the disk time per operation (average time/op). This is the average time an I/O took to complete read or write.

Interface plugin

The Interface plugin collects the information about the traffic octets per second (byte/s). The colour meaning is: blue for receiving traffic and the green for transmiting traffic as in Figure 3.6. The X axis is the time and the Y axis is the number of bytes per second.

3.2 Power Meter TED5000

Ted is a device that detects the power. Ted is used to discover the actual power for any machine such as computers. The model we used is TED5000. This model has a set of current transformers (CRTs), Measuring Transmitting Unit (MTU), Gateway, and Ethernet

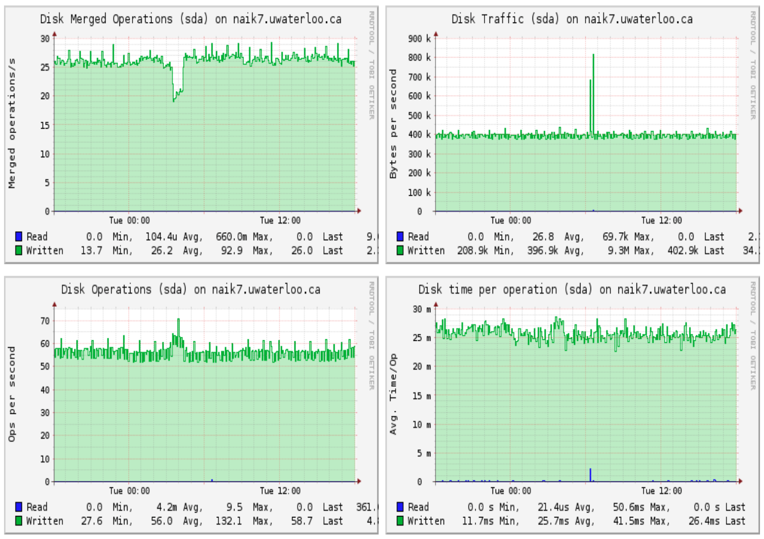


Figure 3.5: Disk plugin graph

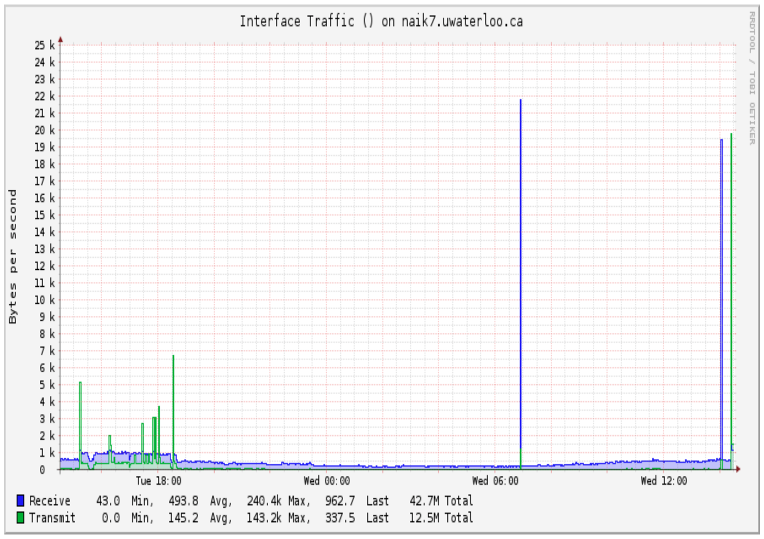


Figure 3.6: Interface plugin graph

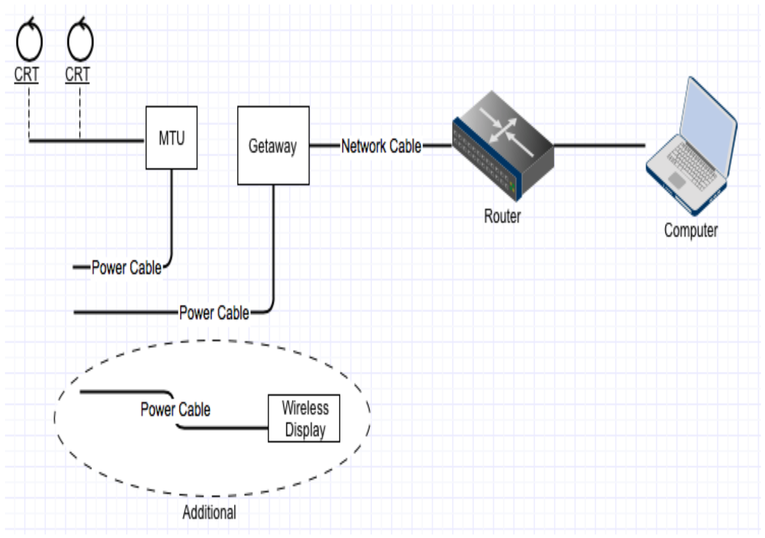


Figure 3.7: TED5000

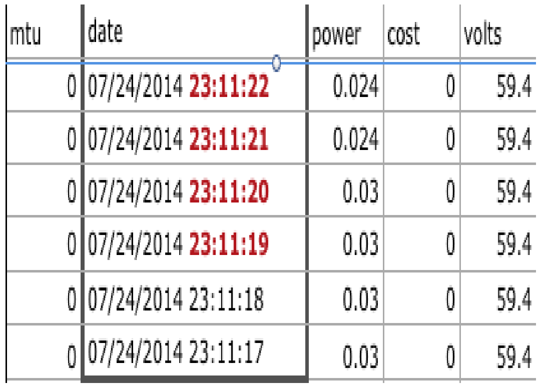
cable. Additionally it has Wireless Display.

As shown in (Figure 3.7) TED5000 works as following:

- The CRTs connected directly to the MTU. CRTs will read the current power and then the MTU will send these reads to the Gateway.
- The Gateway will take the reads of the current power from the power cable and transfer it wirelessly or wired.
- The additional display works wirelessly.
- When using the Ethernet, it can be connected directly to the computer to run its software. However, it is better to connect the Ethernet cable to a router and connect the computer to the same router.

Software Side of TED5000:

The software of TED5000 has some graphing, history, and a live dashboard. It also provides the history in CSV format as shown in Figure 3.8.



mtu	date	power	cost	volts
0	07/24/2014 23:11:22	0.024	0	59.4
0	07/24/2014 23:11:21	0.024	0	59.4
0	07/24/2014 23:11:20	0.03	0	59.4
0	07/24/2014 23:11:19	0.03	0	59.4
0	07/24/2014 23:11:18	0.03	0	59.4
0	07/24/2014 23:11:17	0.03	0	59.4

Figure 3.8: TED5000 output

3.3 Data Simplification

In this work, only four plugins are considered, which are: CPU, Memory, Disk and Interface. These utilizations were collected and simplified as the following sections.

3.3.1 CPU

Regarding the CPU (central processing unit), the measurement was CPU usage as a percentage. This was collected via the*CPU* plugin, which measures the fraction of the time spent by the CPU in several states: Idle, User, Nice, System, Wait, Softer, Interrupt, and Steal. Each state is explained in Table 3.1 [56]. However, *Collectd* gives the CPU utilization core by core. Since one of our system has 8 cores, *Collectd* will return the utility for each core separately as CPU1, CPU2,...CPU8.

By applying different loads to the server to observe the CPU states, it was found that System and User states have the highest performance count. Thus, the CPU usage was calculated as the addition of System and User parameters. As shown in Figure 3.9, CPU (CPU usage) and Idle CPU will be running from each core, then all the eight running CPUs will be added together.

It is important to note that *Collectd* collects statistics measured in *jiffies* (units of scheduling) instead of percentage. On most Linux kernels there are 100 *jiffies* to a second, but depending on both internal and external variables, this might not always be true. However, it is a reasonable approximation.

Table 3.1: CPU matrix

CPU Matrix	Description
Idle	Percentage of time that the CPUs were idle
Interrupt	Percentage of time spent by the CPU to service hardware interrupts
Nice	Percentage of CPU utilization that occurred while executing at the user level with nice priority
Softer	Percentage of time spent by the CPU to service software interrupts.
Steal	Percentage of time spent in involuntary wait by the virtual CPU while the hypervisor was servicing another virtual processor
System	Percentage of CPU utilization that occurred while executing at the system level (Kernel)
User	Percentage of CPU utilization that occurred while executing at the user level(application)
Wait	percentage of time that the CPU was idle during which the system had an outstanding disk I/O request

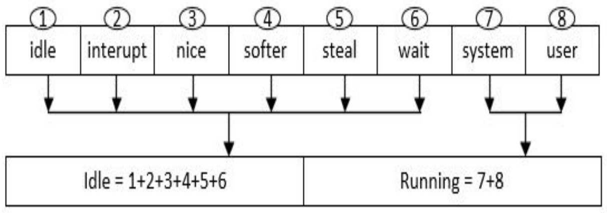


Figure 3.9: CPU simplification

3.3.2 Memory

For memory resources, we defined the percentage used as the core measurement. This was collected via the *Collectd memory* plugin, which has various measurements: Used, Buffered, Cached, and Free (explained in Table 3.2) [56]. The Linux kernel tends to minimize free memory by growing the cache as much as possible, but when an application demands extra memory, cached and buffered memory are released to give way to applications. Therefore, it was decided to consider only the memory used by applications for the percentage used.

Table 3.2: Memory matrix

Memory Matrix	Description
buffered	The amount of memory used as buffers
cached	The amount of memory used for caching
Free	The amount of idle memory
Used	The amount of memory used

3.3.3 Disk

For the hard-disk, the performance statistics were measured using the *Collectd disk* plugin. This plugin gives different measurements as shown in Table 3.3 [56].

Table 3.3: Disk matrix

Disk Matrix	Description
disk-merged read	The number of read operations
disk-merged write	The number of write operations
disk-octets read	bytes read from disk per second
disk-octets write	bytes written to disk per second
disk-ops read	read operations from disk per seconds
disk-ops write	write operations to disk per seconds
disk-time read	average time an I/O read operation took to complete
disk-time write	average time an I/O write operation took to complete

The bytes read/write from/to disk per second were combined together as the disk activities. If the machine has more than one disk, *Collectd* will return utilities for each disk.

3.3.4 Interface

For network interfaces, the *Collectd interface* plugin was used, and bytes per second was chosen as the unit. The *interface* plugin in *Collectd* collects different measurements as ”transfer” or ”receive”. It provides the rate of error, rate of bytes, and rate of packets transferred or received as in (Table 3.4) [56]. Here, the rate of bytes transfered/received is considered because this is the actual bytes received/transmitted by the network interface, and these are combined to obtain the interface activities. Moreover, utilization is provided by *Collectd* for each Ethernet port in the machine.

Table 3.4: Interface matrix

Interface Matrix	Description
if-errors rx	Rate of error in receiving data by network interface
if-errors tx	Rate of error in transmitting data by network interface
if-octets rx	Rate of bytes received by network interface
if-octets tx	Rate of bytes transmitted by network interface
if-packets rx	Rate of packets received by network interface
if-packets tx	Rate of packets transmitted by network interface

3.3.5 Actual Power

A measurement of the actual power is collected by using the energy detector (TED 5000). TED has been used because its reading are very accurate and it is simple to install [57]. Moreover, this device has a set of Current Transformers (CRT) that clips over the main power cable of a machine. In our test bench, it was connected to the server. The CRT will send the power reading to a Gateway tool, and from this tool, to the router over an Ethernet cable. The machine that has *Collectd* running on is connected to the same router. In this way, *Collectd* can take the power reading from TED because they are all on the same local area network (LAN). A plugin has been written and added to *Collectd* as a Python plugin to collect measurements from TED. The plugin code is as follows:

```
<Plugin python>
    Module path "/usr/lib/collectd"
    Import "ted5000"

            <Module ted5000>

                    Host "129.97.10.85"
                    DkipZeroes "true"
                    Verbose "false"

            </Module>
```

```
</Plugin>
```

The *Host IP address* (e.g, 129.97.10.85) is the TED's getaway address that collects the actual power read and is connected to the router.

Chapter 4

System Model of Test Bench and modelling Process

In this chapter, the test bench that was used to create the models is explained. A description is included of the modelling process and two different techniques for modelling which are Neural Network and Linear Regression. This chapter will also explain two different workload designs that have been used in three machines. Moreover, validation of the models is provided in this chapter.

4.0.1 Test Bench

This experiment is dependent on a non-exclusive test bench that was used in our previous work[20] for modelling the power. There are several tools that have been used in this experiment. Figure(4.1) illustrates the test bench framework.

The figure has the following terms:

1. Performance Counters (*Collectd*): a software tool that collects the machine's performance such as the CPU usage, Memory, etc. This tool is running in the background of the system to gather the system's performance data (explained in detail in Section 3.1) . It has many plug-ins that a user can enable or disable. In the current modelling, the author uses five plug-ins (the CPU, Memory, Disk, Interface and TED plug-in). Each plug-in returns many counters. For example, the CPU plug-in returns eight counters for each core in the CPU. On the other hand, the TED plug-in -written in Python to extract the power- reads from the power meter (TED5000).

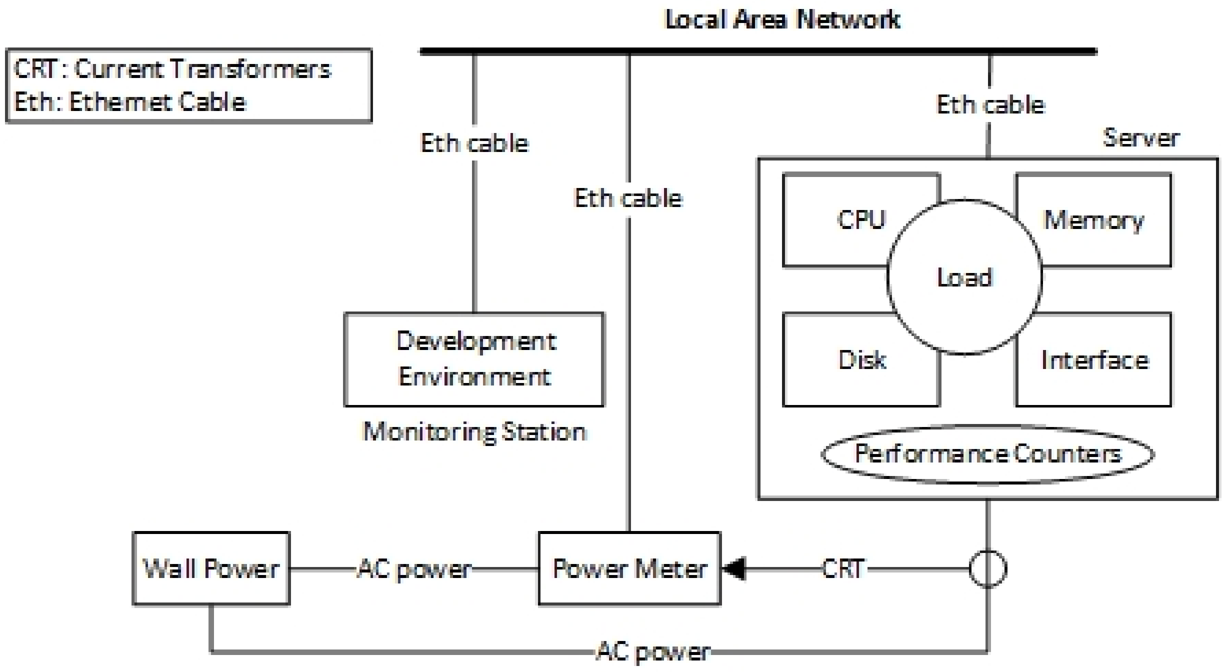


Figure 4.1: Test bench settings

2. Power Meter (TED5000): An external device that collects the actual power usage of a machine. It returns the power in Voltage for each second (explained in detail in Section 3.2).
3. Servers: In this work, three machines are used that have different hardware features. The server's parameters are shown in Section 4.3.
4. Development Environment: A computer used to store the collected data. The first tool in the development environment (computer) is Matlab. Matlab is a software that has been used for extracting the data from different files, simplifying the data, and creating the models.
5. Load: There are two meanings for the load in this figure. The first is the workload (Workload A and B) that has been used to create the models. The second is the applications that have been used to validate the model or want to estimate its power consumption.

The modelling process has two phases as shown in Figure(4.2). In the first phase

(Extracting the Data) there is the machine (server) that for which a model will be created. The machine should be connected to the power wall, to a Local Area Network (LAN) and running the *Collectd* tool. The second device in this phase is the power meter (TED5000). The power meter is also connected to the power wall and to the LAN. TED and the server should be connected to the same LAN so they can communicate. Additionally, TED detects the power through a Current Transformer that is connected to the power cable between the server and the wall power. From there, *Collectd* collect the power readings from TED via the LAN. Now, the server is ready to run the workloads, which will be used in the modelling Process phase. While running the workload in the server, *Collectd* and TED collects the data and stores it as a Comma-Separated Values (CSV) format. The data will be stored in files in the server and these files are shared with the development environment (Computer).

The second phase starts with collecting the resources utilization (data) via the *Matlab*. Next, is the process of simplifying the data, which has been done by a code written in *Matlab*. Briefly, the simplification of the data is as follows (explained in detail in Section 3.3):

- CPU: For each core, *Collectd* will return eight counters (i.e. idle, interrupt, nice, softer, steal, wait, system and user). The counters' description can be found in Section 3.3. The CPU counters are simplified to running and idle CPU.
- Memory: From *Collectd* four counters were obtained which are: used, buffered, cached and free. These counters are simplified as used memory and free memory.
- Disk: This plug-in will return the usage for each disk in the server and for each partition. We are considering only the bytes read/write operations (disk-octets). All the disk-octets from all disks are added together to obtain the disk activities.
- Interface: This plug-in is returning the counters for each Ethernet port. From this plug-in, the rates of bytes transfeard/received by the network interface is taken and added together to obtain the interface activities.

The next step in the modelling Process phase is the modelling. Two different techniques are used to model Neural Network and Linear Regression.

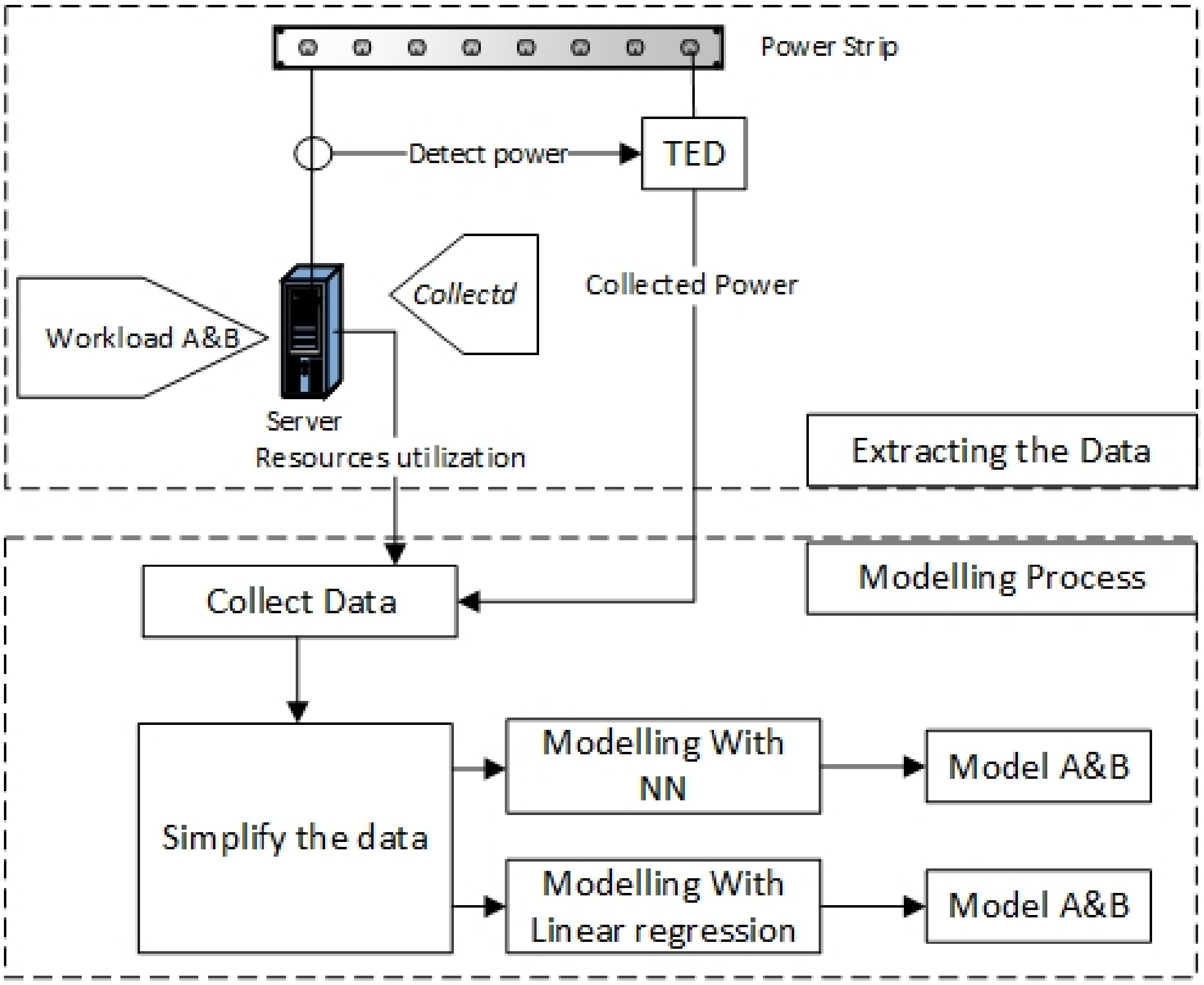


Figure 4.2: Modelling process

4.0.2 Phases of the modelling Process

The initialization and operation phases of the modelling process is as shown in (Figure 4.3). There are two phases in this model: the initialization phase set up to create the models and the operating phase where the models will be applied to estimate the power consumption while running various applications.
The explanation of the Initialization phase in this flow chart is as follows:

1. As a first step, all the hardware is connected that is needed for the experiment. In this work, the power meter TED5000 is connected to the main power cable that is connected to the server. It is also connected to *Collectd* through the network.

2. After connecting the power meter, the power meter reading will begin. TED5000 has its own software that shows the power reading in seconds and the power in Watts with the time stamp.

3. On the server, *Collectd* is run to gather the system performance. In this work, four counters have been considerd: (the CPU, Memory, Disk and Interface). Also, the fifth counter is for the TED5000 to gather the power reading.

4. In this step, the data has been collected from many files in CSV format. Steps 4-10 will occur in the development machine.

5. Here the data from many counters is simplified and reduced to only four counters.

6. The data are presented in two tables The first is Resource Utilization (RU) which has the four subsystems' performance (the CPU, Memory, Disk and Interface). The second table is the actual power reading table (Pa).

7. At this point, we are left with two choices, which are to model with Neural Network or Linear Regression. Here, the Neural Network requires two inputs, RU table as input and Pa table as the target.

8. The Linear Regression requires the same two inputs which are RU and Pa.

9. After training the data with Neural Network, the model (Net File) will be obtained, which has all the model information.

10. After training the data with Linear Regression, the model (File) which has all the model information will be obtained.

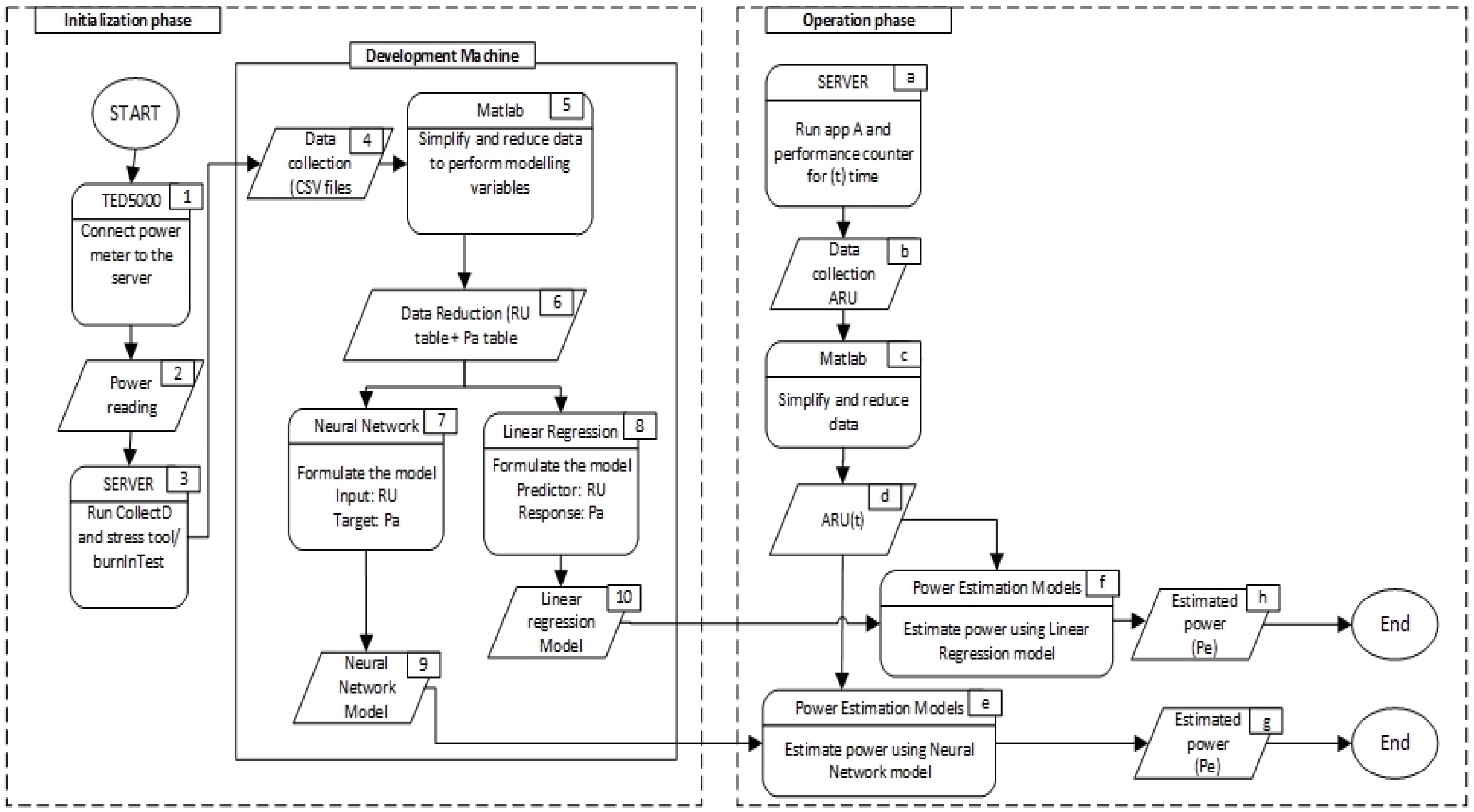

Initialization phase
START
TED5000 1
Connect power meter to the server
Power reading 2
SERVER 3
Run CollectD and stress tool/ burnInTest
Development Machine
Data collection (CSV files 4
Matlab 5
Simplify and reduce data to perform modelling variables
Data Reduction (RU table + Pa table 6
Neural Network 7
Formulate the model Input: RU Target: Pa
Linear Regression 8
Formulate the model Predictor: RU Response: Pa
Neural Network Model 9
Linear regression Model 10
Operation phase
SERVER a
Run app A and performance counter for (t) time
Data collection ARU b
Matlab c
Simplify and reduce data
ARU(t) d
Power Estimation Models f
Estimate power using Linear Regression model
Power Estimation Models e
Estimate power using Neural Network model
Estimated power (Pe) h
Estimated power (Pe) g
End
End

Figure 4.3: Phases of the modelling process

The explanation of operation phase in this flow chart is as follows:

a) The first step in the operation phase is to run an application (A) and collect the system performance while running this app. Since the models are already known, there will be no further need for the power meter.

b) Then the performance of the subsystems will be collected that was obtained by *Collectd*.

c) After that, the data will be simplified and reduced in order to have only the four matrices of the subsystems (the CPU, Memory, Disk and Interface).

d) The RU is ready for estimating the application's power by this study's models (Neural Network model or Linear Regression model).

e) The power consumption of software (A) with Neural Network (NN) model obtained from the Initialization phase is estimated.

f) The power consumption of software (A) with Linear Regression model obtained from the Initialization phase is estimated.

g) This is the step which estimates power by the NN model.

h) This is the step which estimates power by the Linear Regression model.

4.1 Modelling Techniques

This section provides an explanation of the two modelling techniques that have been used in this work, which are Neural Network and Linear Regression.

4.1.1 Neural Network Model

Neural Network (NN) is a nonlinear information processing technique [58]. The function of the NN is to map an input into a desired output [59]. Figure 4.4 shows the artificial of a simple NN, where the input can be 1-n (in this work, there are four inputs). In the figure, there is one hidden layer where the number of neurons in this layer can be selected by the user.

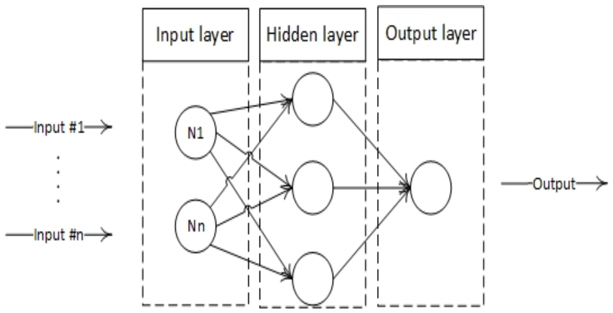


Figure 4.4: Artificial Neural Network

The Neural Network (NN) Fitting Tool (nftool) in Matlab has been used to create the models. In Neural Network, two matrices should be provided to create the model. The first matrix (Input) will have the four subsystem utilities (i.e. Running CPU, Used Memory, Disk Activities and Interface Activities). The second matrix (Target) will have the actual power utility. The data that has been collected while running the workload will be divided into three samples (default): 70% for training, 15% for validation, and 15% for testing. Also, the number of the hidden neurons is 10. These default settings have been chosen because changing them will not effect the accuracy of the model that much. After training the data, the NET file (Model) will contain all the information about the data and the NN configurations. By using the NN, two models were created for each machine in this work; Model A which is based on Workload A, and Model B which is based on Workload B. However, to get the Neural Network model we perform the following equation:

$$RU^L = [U_c^L, U_m^L, U_d^L, U_i^L], \tag{4.1}$$

where RU^L is the resources utilization while performing the workload, U_c^L is the CPU utilization, U_m^L is the memory utilization, U_d^L is the disk utilization, U_i^L is the Interface utilization. Then for creating the model in Matlab, the following equation is performed:

$$\{C\} = f(RU^L, P_a^L), \tag{4.2}$$

where $\{C\}$ is the model, f is the Neural Network function, RU^L is the resources utilization of loading the four subsystems (Workload A or Workload B), and P_a^L is the actual power.

After obtaining the Model, an application is run and the resources utilization of the four subsystems is collected. Let RU^A be the resources utilization of application A. Then the following equation is applied to estimate the power of application A as follows:

$$RU^A = [U_c^A, U_m^A, U_d^A, U_i^A], \tag{4.3}$$

$$P^E = f(RU^A, \{C\}) \tag{4.4}$$

where P^E is the estimated power, f is the function that used to estimate the power, and $\{C\}$ is the model that we got from equation 4.2.

4.1.2 Linear Regression Model

The second modelling technique used in this work is Linear Regression. This technique has been used in several researches as discussed in the related work (Chapter 2). In Linear Regression the input data are the predictor variables and the response variables. The predictor variables in this work are the four subsystems matrix and the response variables represent the actual power. The equation of the Linear Regression in this case will be written as follows (the same equation has been mentioned in [60]):

$$P_a^L = C_0 + C_1U_c^L + C_2U_m^L + C_3U_d^L + C_4U_i^L, \tag{4.5}$$

where P_a^L is the actual power while performing all the workload design , C_0 is the idle power, $[C_1, C_2, C_3, C_4]$ are the set of the fitting parameters, U_c^L: the CPU utilization, U_m^L : the Memory utilization, U_d^L : the Disk utilization, and U_i^L : the Interface utilization.
Moreover, The fitting data and the actual power will be the data that collected during Workload A or Workload B (explained in Section 4.2).

To obtain the linear regression model in Matlab, the following equation is performed:

$$RU^L = [U_c^L, U_m^L, U_d^L, U_i^L], \tag{4.6}$$

$$\{C\} = f(RU^L, P_a^L) \tag{4.7}$$

where $\{C\} = [C_0; C_1; C_2; C_3; C_4]$(The Model), f is the Linear Regression function, RU^L is the resources utilization of loading the four subsystems (Workload A or Workload B), and P_a^L is the actual power.

After obtaining the Model, an application is run and the resources utilization of the four subsystems is collected. Let RU^A be the resources utilization of application A. After obtaining the Model, the following equation is applied to estimate the power of application A as follows::

$$RU^A = [U_c^A, U_m^A, U_d^A, U_i^A], \tag{4.8}$$

$$P^E = f(RU^A, \{C\}) \tag{4.9}$$

where P^E is the estimated power, f is the function that is used to estimate the power, and $\{C\}$ is the model obtained from equation 4.7.

Linear Regression technique has been used to create two models for each machine: Model A, which is based on Workload A, and Model B, which is based on Workload B.

4.2 Workload Design

Loading the system to create the model is very important. In some experiments conducted, the load was not enough (short time) for creating an accurate model for a certain machine. However, in other experiments the system was overloaded, which created an overtrained model. Moreover, designing the load for the model is the key for an accurate model. The experiment shows how designing the load could cause very low or very high estimated results. In this work, two different loads were designed that were applied in three different machines.

The first workload (Workload A) has four steps: The first step is loading the CPU core by core where the first core is loaded to 100% for 1 minute, then two cores are loaded, etc. until all cores are loaded to 100%. The second step is loading the memory gradually until it is loaded to 100%. The third step is loading the hard disk as done with memory. The fourth step is loading the interface by downloading large files (approximately 7Gb). The first three steps took 24 minutes, while the interface load took approximately 40 minutes.The total time that Workload A took is 64 minutes.

The second workload (Workload B) has six steps. The first three steps are similar to Workload A except for the time taken to load each subsystem. Then, the interface took approximately 60 minutes. The fifth step was simulating the daily activity that occurred in the server for 60 minutes. The final step involved loading all the four subsystems (the CPU, Memory, Disk, and Interface) to the maximum limit for 30 minutes. This model took approximately 390 minutes. The matrix of the Workload A is organized as:

$$RU(CPU, Memory, Disk, Interface) \tag{4.10}$$

Also the matrix of Workload B is organized as:

$$RU(Idle, Daily, CPU, Memory, Disk, Interface, Loadall) \tag{4.11}$$

The differences between Workload A and Workload B are the load design and the length of the workload. The loads for each workload are as shown in Tables (4.1, 4.2).

For the CPU load, a tool called *Stress* was used. *Stress* is a workload generator for POSIX (Portable Operating System Interface for Unix) systems. In this tool, users can choose how many cores from the CPU they want to load to 100% and for how long. It works by directing the core to calculate the square root of a random number by calling the *rand* and *sqrt* functions in a loop until the timeout that has been chosen. We stressed each core to 100% for 1 minute in Workload A and 10 minutes in Workload B. The total time for loading all cores in a machine that has 8 cores in Workload A is 8 minutes and 80 minutes in Workload B.

Stressing the memory via *Stress* tool is done by thrashing the RAM (Random Access Memory) by forking processes to allocate and release memory and that is done by calling (*Looping malloc()*, and *free()*) functions. The memory was loaded gradually until it reach 100% for 8 minutes in Workload A and 80 minutes in Workload B. Stressing the memory should be done carefully because loading the memory more than 100% will cause the system to crash.

Also, the *Stress* tool was needed to load the disk. For the hard disk, the *Stress* tool will force the hard disk with writes in loop by calling *write* function. The hard disk was loaded gradually. Essentially, the hard disk was loaded until the memory reached 100% because loading the hard disk affects the memory directly. This process has a timeout of 8 minutes in Workload A and 80 minutes in Workload B.

However, for the interface, the load in Workload A was different than Workload B. In Workload A the load was downloading large files (approximately 7 GB), but in Workload B we used a tool called *BurnInTest*. In Workload A, we collected the data from start to finish of the downloading process, but in Workload B, the tool was run to stress the interface for 60 minutes.
The next three loads are only in Workload B. The daily activity load is simulating the daily activity in the server. This load was performed by browsing the server, browsing the web and running some applications for approximately 60 minutes.

However, "Load all" means loading all the four subsystems (the CPU, Memory, Disk, and Interface) together. That was done by using the *Stress* tool for Disk and *BurnInTest* tool for CPU, Memory and Interface simultaneously. This test ran for 30 minutes. Lastly, the server was run and kept idle from the user perspective for 120 minutes.
After all, the idea behind running the load for a longer duration in Workload B and adding three more loads is to gather more information about the machine for the model to be more aware of the changes in the system's performance and power consumption.

Table 4.1: Load design for Workload A

Utility	Workload A
CPU	Loading each core to 100% for 1 minute
Main Memory	Gradually loading memory to 100 for 8 minutes %
Hard Disk	Gradually loading disk to 100 % for 8 minutes
Interface	Loading the interface by downloading large files

Table 4.2: Load design for Workload B

Utility	Workload B
CPU	Loading each core to 100% for 10 minutes
Main Memory	Gradually loading memory to 100 % for 80 minutes
Hard Disk	Gradually loading disk to 100 % for 80 minutes
Interface	Loading interface for 60 minutes
Daily activity	Browsing the server as daily activity for 60 minutes
Load all	Load CPU,Memory,Disk and Interface for 30 minutes
Idle	Running Server without any load

4.3 Validation

Validating the models is an important stage of the modelling process because it shows the accuracy of the model. These experiments of modelling and validation have four diminutions. First, three real servers with different configurations as shown in (Table 4.3, Table 4.4, and Table 4.5) have been used. Second, two workloads designs have been explained in Section 4.2. Third, two techniques of modelling are used in this work which are Neural Network and Linear Regression. Fourth, four scenarios have been used to validate the models.

Table 4.3: Server 1 machines configuration

Parameter	Real Server (Dell PowerEdge 2950)
Processor	7x Intel Xeon, 3 GHz, 4 cores per processor
Hard Disk	1.7 Tera Bytes SAS
Main Memory	32 Giga Bytes
Interface	Ethernet cable, 100 Mb/s
Operating System	Linux (Ubuntu 13.10)

Table 4.4: Server 2 machines configuration

Parameter	Real Server (HP ProLiant DL385G5)
Processor	Quad-Core AMD Operon(tm) Processor 2356x4
Hard Disk	55.2 Giga Bytes
Main Memory	15.7Giga Bytes
Interface	Ethernet cable, 100 Mb/s
Operating System	Linux (Ubuntu 14.04 LTS)

For each machine, two workloads (Workload A and Workload B) have been used to create four models (two Neural Network models based on Workload A/Workload B, and two Linear Regression models based on Workload A/Workload B). To validate these models, we have demonstrated four scenarios and calculating their errors. We are counting two types of errors, Mean Absolute Error of power (MAEp), and the Mean Absolute Error of energy (MAEe). In the MAEp we are presenting the maximum point of estimated power reached during the load and the minimum point in percentage and comparing it with the actual power. The MAEp was calculated by a function in Matlab. MAEp min/max was calculated as (Equation 4.12, 4.13).

$$MaxError = \frac{MAE_p}{MaxPower} * 100 \tag{4.12}$$

Table 4.5: Server 3 machines configuration

Parameter	Real Server (Dell PowerEdge 2950)
Processor	Intel Xeon CPU 5160 @ 3.00 GHz x 4
Hard Disk	277.8 Giga Bytes
Main Memory	7.8 Giga Bytes
Interface	Ethernet cable, 100 Mb/s
Operating System	Linux (Ubuntu 14.04 LTS)

$$MinError = \frac{MAE_p}{MinPower} * 100 \tag{4.13}$$

Where the *MaxPower* is the highest point the power reach during the load, and *MinPower* is the lowest point. Moreover, the error was calculated as the difference between the actual power and the model output (the estimated power). Then we calculate the absolute value of MAEe as shown in (Equations 4.14 4.15).

$$Error = ActualPower - ModelPower \tag{4.14}$$

$$|MAE_e| = \frac{Error}{ActualPower} * 100 \tag{4.15}$$

We chose to conduct four scenarios that stress our four subsystems. Downloading (first scenario) is stressing most the interface. Gaming (second scenario) is stressing the memory and the CPU for processing. Also, local movie (third scenario) is stressing the disk and the memory. Finally, Youtube (fourth scenario) is stressing most the interface and the memory. Figures 4.5, 4.6, 4.7 show the differences in the MAEe between all models. Where in the graphs NN-A and NN-B means Neural Network Model A and Model B. LR-A and LR-B means Linear Regression Model A and Model B. The figures also show the error of using two techniques of modelling which are Neural Network and Linear Regression. However, the four scenarios explanation is as follow:

- Scenario 1 (Downloading): In this scenario we tried to load the system a real load. This was done by downloading large files from the Internet. We used Firefox browser to access a web page that has many movies in high quality. We downloaded four movies from the same web page at the same time, which has in total, approximately 7 GB. Then we applied our models to estimate the power consumption of this experiment after collecting the data during the downloading. We downloaded the same movies in all three servers.

- Scenario 2 (Gaming): We installed a game in the machine and start collecting the system performance from when we started plying the game. We played the game for about 30 minute. The same experiment was performed on all three servers. The game in Server 2 and 3 was too heavy for the system. For example, we could click a bottom in the game and the response for that click will take about 20 seconds.

- Scenario 3 (Local Movie): After downloading a high quality movie in the hard drive, we played the movie and collected the data for 30 minutes. The movie was viewed in full screen. The same movie was played in all the servers with the same player software.

- Scenario 4 (YouTube): One of the high quality videos on YouTube has been chosen for this experiment. The video was 20 minutes in length but we collected the data for 15 minutes. The same video was chosen in all servers.

Generally, the results of the validation were as follows:

- Server 1, Models A: There are no big differences between Neural Network (NN) and Linear Regression (LR) in this model.

- Server 1, Models B: NN has fewer errors than the LR model.

- Server 2, Models A: First scenario (downloading), NN perform better but the other scenarios LR is better.

- Server 2, Models B: The NN model has less errors than LR.

- Server 3, Models A: LR has fewer errors than the NN model in most scenarios.

- Server 3, Models B: NN has fewer errors than the LR model in most scenarios.

The following sections explain the errors in further detail for each modelling technique.

4.3.1 Neural Network Model Results

The first model (Model A) has been created based on Workload A in all the servers and (Model B) based on Workload B. When we validated Model A in Server 1 the MAEe was between 12.64% and 15.07% which is considered a high error in comparison to other servers. From here we designed Workload B to create Model B. The errors in Model B were impressive. The MAEe was between 0.07% and 2.39%. Figure 4.5 shows the errors for Models A and B in server 1 for the two techniques of modelling.

However, in Server 2 and 3, workload B has fewer errors compared to Workload A. The MAEe for the models that are based on Workload A are as follow: in Server 2 was between 0.62% and

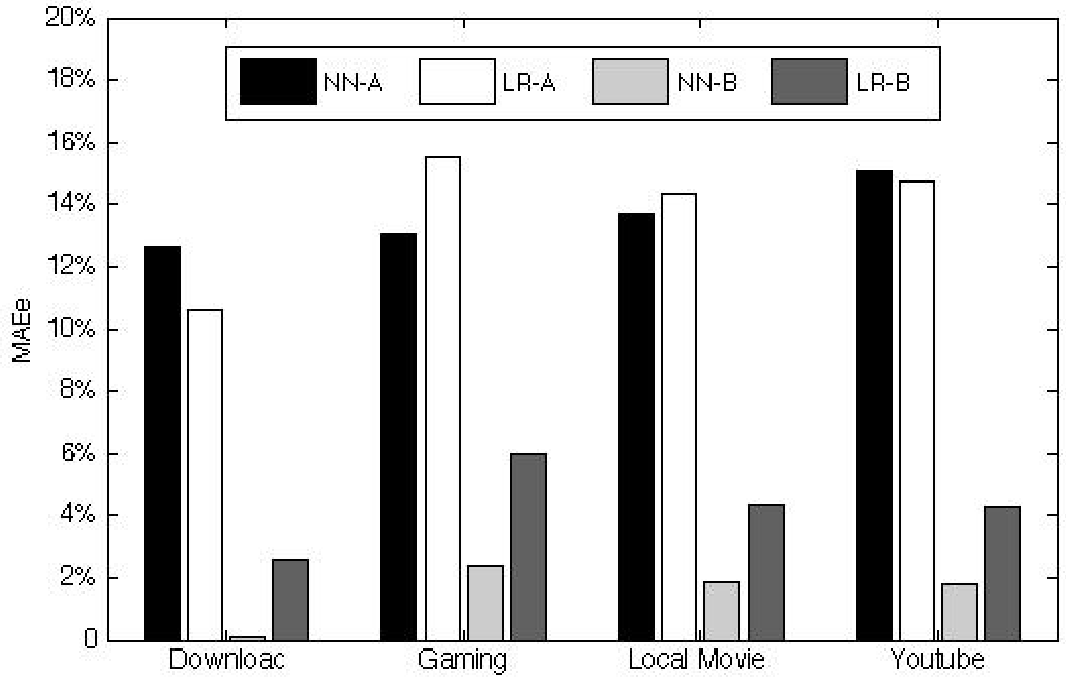


Figure 4.5: MAEe comparison between all models in Server 1

5.95%, and in Server 3 the MAEe was between 3.12% and 6.11%; for models that are based on Workload B the MAEe in Server 2 was between 0.14% and 3.33%, and in Server 3 the MAEe was between 0.40% and 3.67%. Figures (4.6, 4.7) show the errors in server 2 and server 3. Generally, in all the servers all the models that are based on Workload B perform better than models that are based in Workload A.

4.3.2 Linear Regression Model Results

Linear Regression models Model A is based on Workload A and Model B is based on Workload B. The MAEe for the models that are based on Workload A are as follow: between 10.60% and 15.51% in Server 1, between 3.67% and 10.92% in Server 2, between 0.23% and 3.60% in Server 3.

However, the MAEe for the models that are based on Workload B are as follow: between 2.56% and 6.01% in Server 1, between 0.01% and 10.99% in Server 2, between 0.46% and 7.98% in Server 3.

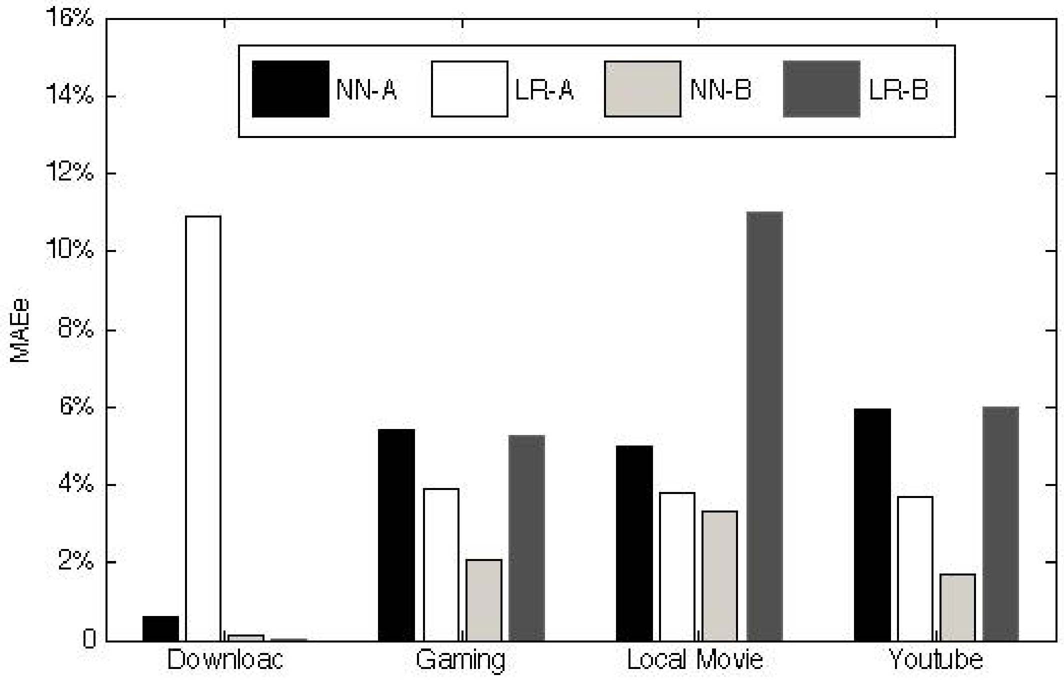


Figure 4.6: MAEe comparison between all models in Server 2

For Servers 1 and 3 Model B has fewer errors than Model A, but in Server 2 Model A has fewer errors than Model B as shown in Figures (4.5, 4.6, 4.7).

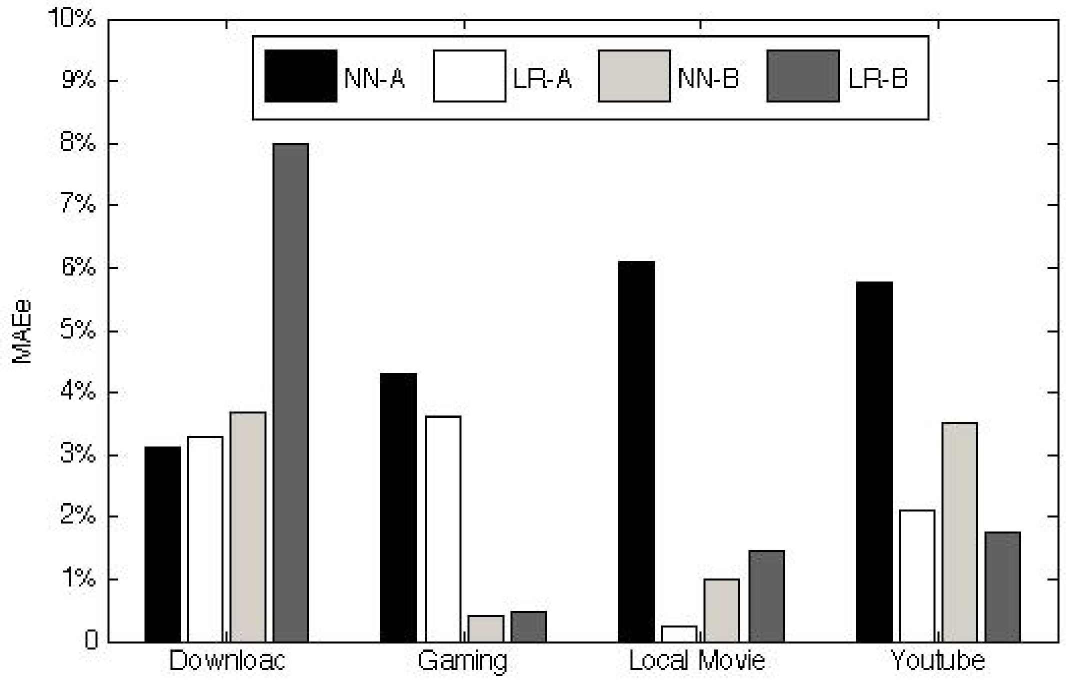


Figure 4.7: MAEe comparison between all models in Server 3

Chapter 5

Importance of Subsystem Selection for Creating the Models

Selecting the subsystem for a model as inputs is very important for the accuracy of the model. In the current work, various models have been created that use different inputs and evaluated the models by the same applications. The inputs are as follows: CPU, CPU & Memory, CPU & Disk, CPU & Interface, CPU & Memory & Disk, CPU & Memory & Interface, and CPU & Memory & Disk & Interface (Table 5.1 shows the model's inputs). Then the same inputs were used to create models using Neural Network and Linear Regression technique. Two servers and two workload designs have been used in this experiment. The following results were obtained:

1. Four subsystems (CPU, Memory, Disk and Interface) were used, as the model input is more accurate across different workload design and different hardware platforms.
2. Neural Network produced better results with Workload B, while Linear regression produced better results with Workload A.

Figures (5.1 and 5.2) show the errors of the models that are created on Server 1 using two modelling techniques: NN and LR. Figures (5.3 and 5.4) show the errors of the models that are created on Server 2 using two modelling techniques: NN and LR.

Table 5.1: Different models' inputs for Neural Network and Linear Regression

Model Input	Neural Network	Linear Regression
CPU	$RU^L = [U_c^L]$	$P_a^{L1} = U_c^L$
CPU,Memory	$RU^L = [U_c^L, U_m^L]$	$P_a^{L1} = U_c^L + U_m^{L1}$
CPU,Disk	$RU^L = [U_c^L, U_d^L]$	$P_a^{L1} = U_c^L + U_d^{L1}$
CPU,Interface	$RU^L = [U_c^L, U_i^L]$	$P_a^{L1} = U_c^L + U_i^{L1}$
CPU,Memory,Disk	$RU^L = [U_c^L, U_m^L, U_d^L]$	$P_a^{L1} = U_c^L + U_m^{L1} + U_d^{L1}$
CPU,Memory,Interface	$RU^L = [U_c^L, U_m^L, U_i^L]$	$P_a^{L1} = U_c^L + U_m^{L1} + U_i^{L1}$
CPU,Memory,Disk,Interface	$RU^L = [U_c^L, U_m^L, U_d^L, U_i^L]$	$P_a^{L1} = U_c^L + U_m^{L1} + U_d^{L1} + U_i^{L1}$

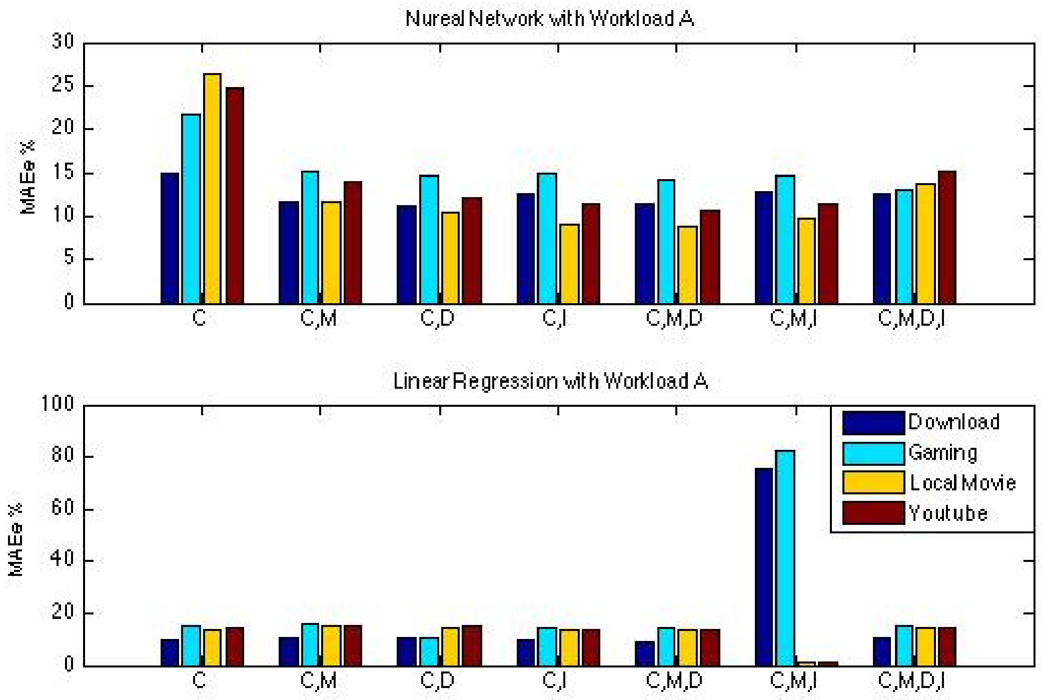


Figure 5.1: Errors of using different inputs for modelling in Server 1 using Workload A

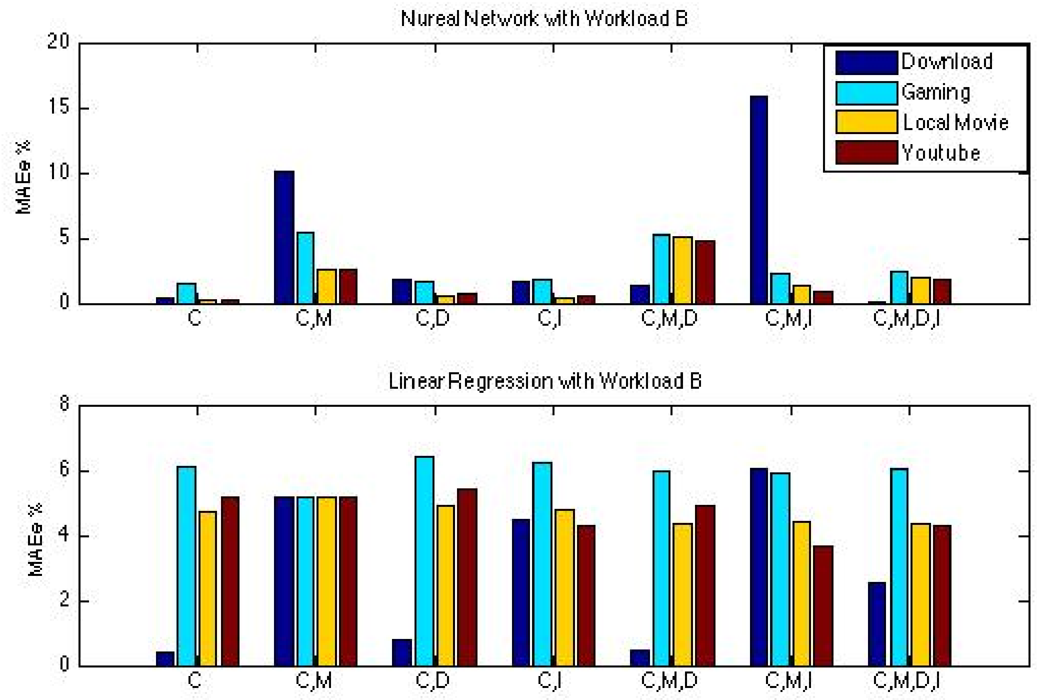


Figure 5.2: Errors of using different inputs for modelling in Server 1 using Workload B

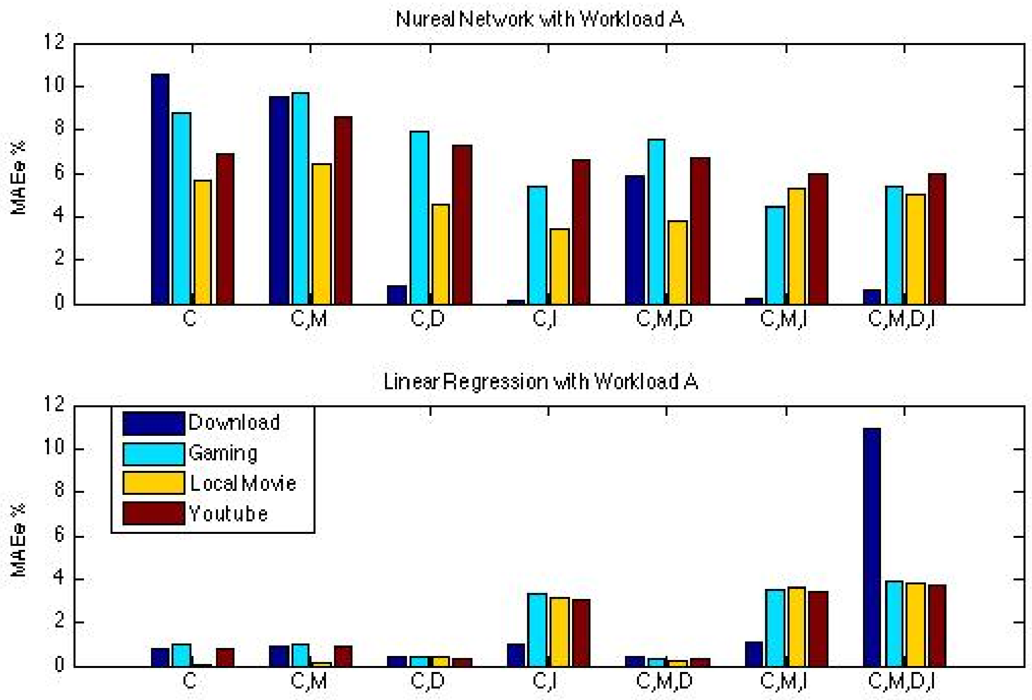


Figure 5.3: Errors of using different inputs for modelling in Server 2 using Workload A

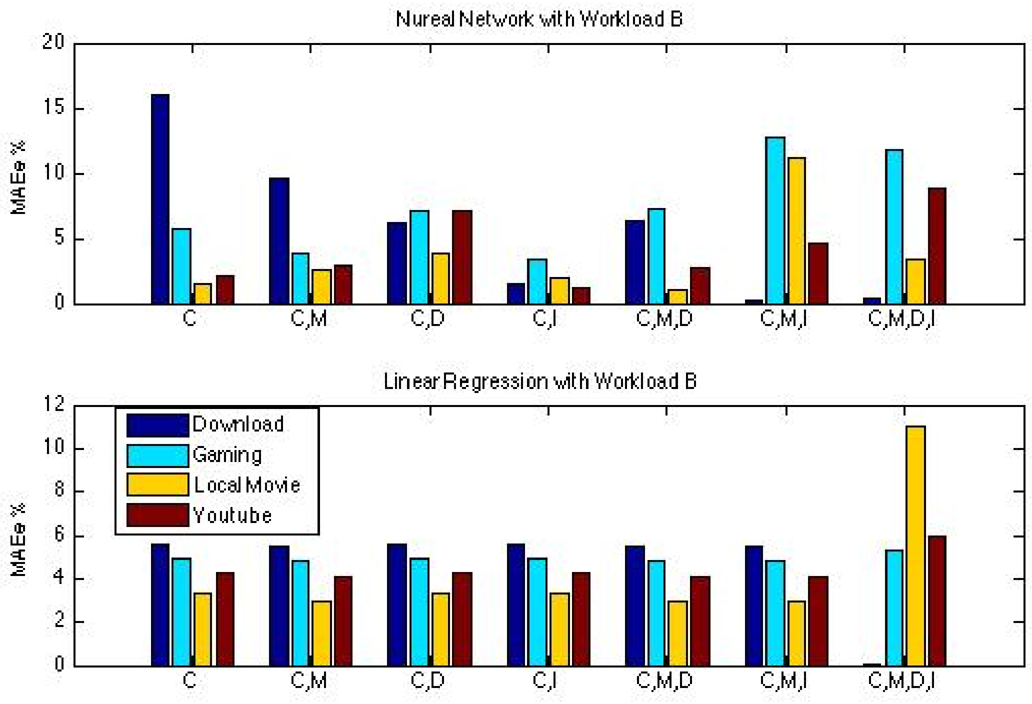


Figure 5.4: Errors of using different inputs for modelling in Server 2 using Workload B

Chapter 6

Automation Test Bench

Automating our test bench is very important for developers, as it saves time, is easy of use and reduces complexity. In this chapter the automation of the test bench–which has two phases–will be explained. The first phase is on the server side where the developer can design the workload and stress the subsystems they selected as the model inputs. The second phase involves collecting/simplifying the data, creating the models and estimating the power consumption of various applications.

The workflow of the entire work is shown in Figure 6.1. This figure summarizes the work in three steps. These steps are explained in detail in Chapter 4. The first process in this graph includes the following:

* Connect TED to server n.
* Install *Collectd* in server n.
* Design and run the workloads and collect the system performance while running the load on server n.
* Collect the actual power consumption data by TED while running the workload on server n.
* Develop the models based on the collected data about the system performance and the actual power consumption.

The second process on the graph includes the following:

* Run application A on server n.

* Collect the system performance while application A is running on server n by *Collectd*.

The third process is to estimate the power consumption of application A using the models that have been developed in process 1.

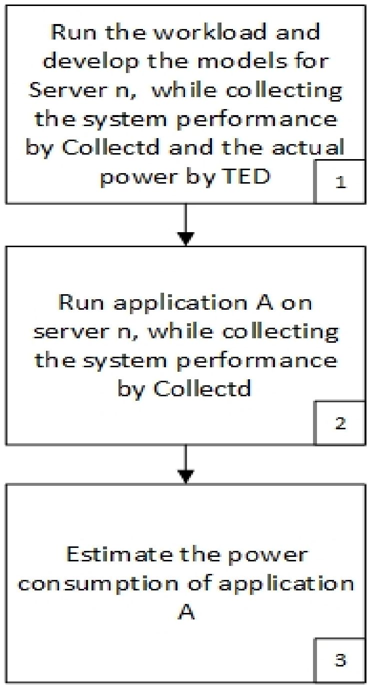

Run the workload and develop the models for Server n, while collecting the system performance by Collectd and the actual power by TED
1
Run application A on server n, while collecting the system performance by Collectd
2
Estimate the power consumption of application A
3

Figure 6.1: Flowchart summary of the entire work

As mentioned the automation test bench steps were divided into two phases as shown in Figure 6.2. The first phase will be performed on the server side, and the second phase will be performed on the client side. The explanation of this graph is as follows:

1. Phase 1, Process 1:

   * Connect TED to server n.

   * Install *Collectd* in server n.

   * Design and run the workloads and collect the system performance data while running the load on server n.

   * Collect the actual power consumption by TED while running the workload on server n.

2. Phase 2, process 2:

* Develop the models based on the collected data about the system performance and the actual power consumption.

3. Phase 1, Process 3:

   * Run application A on server n.
   * Collect the system performance while application A is running on server n by *Collectd*.

4. Phase 2, Process 4:

   * Estimate the power consumption of application A using the models that have been developed in Phase 2, Process 2.

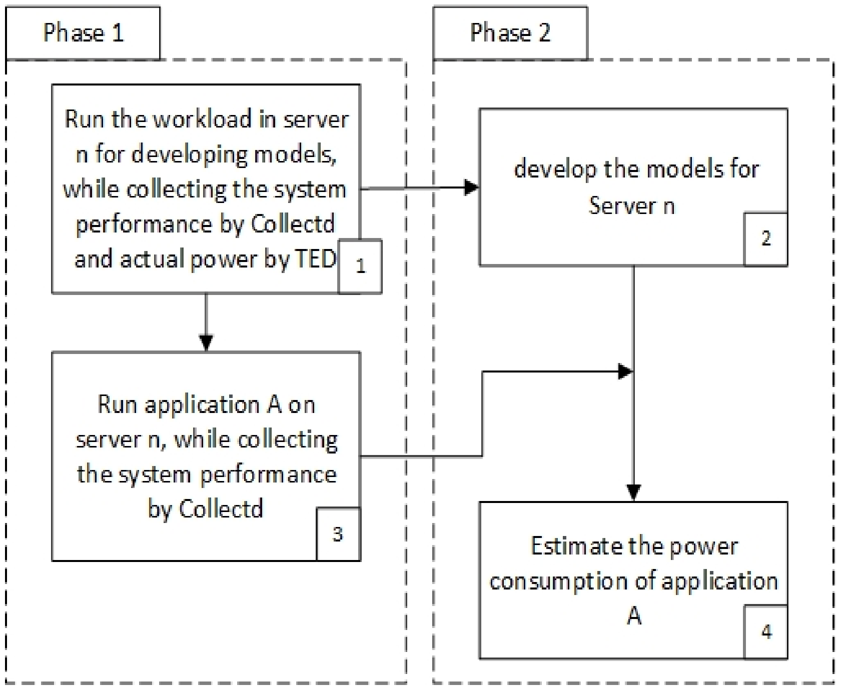

Phase 1
Phase 2
Run the workload in server n for developing models, while collecting the system performance by Collectd and actual power by TED
1
develop the models for Server n
2
Run application A on server n, while collecting the system performance by Collectd
3
Estimate the power consumption of application A
4

Figure 6.2: Flowchart summary of the entire work in two phases

6.1 Systems Requirements

To run the test bench on any server, there are some tools must be installed first. The tools are divided into two phases: server and client. Table 6.1 shows all the tools that should be installed before running the developed tools.

Table 6.1: Systems requirements

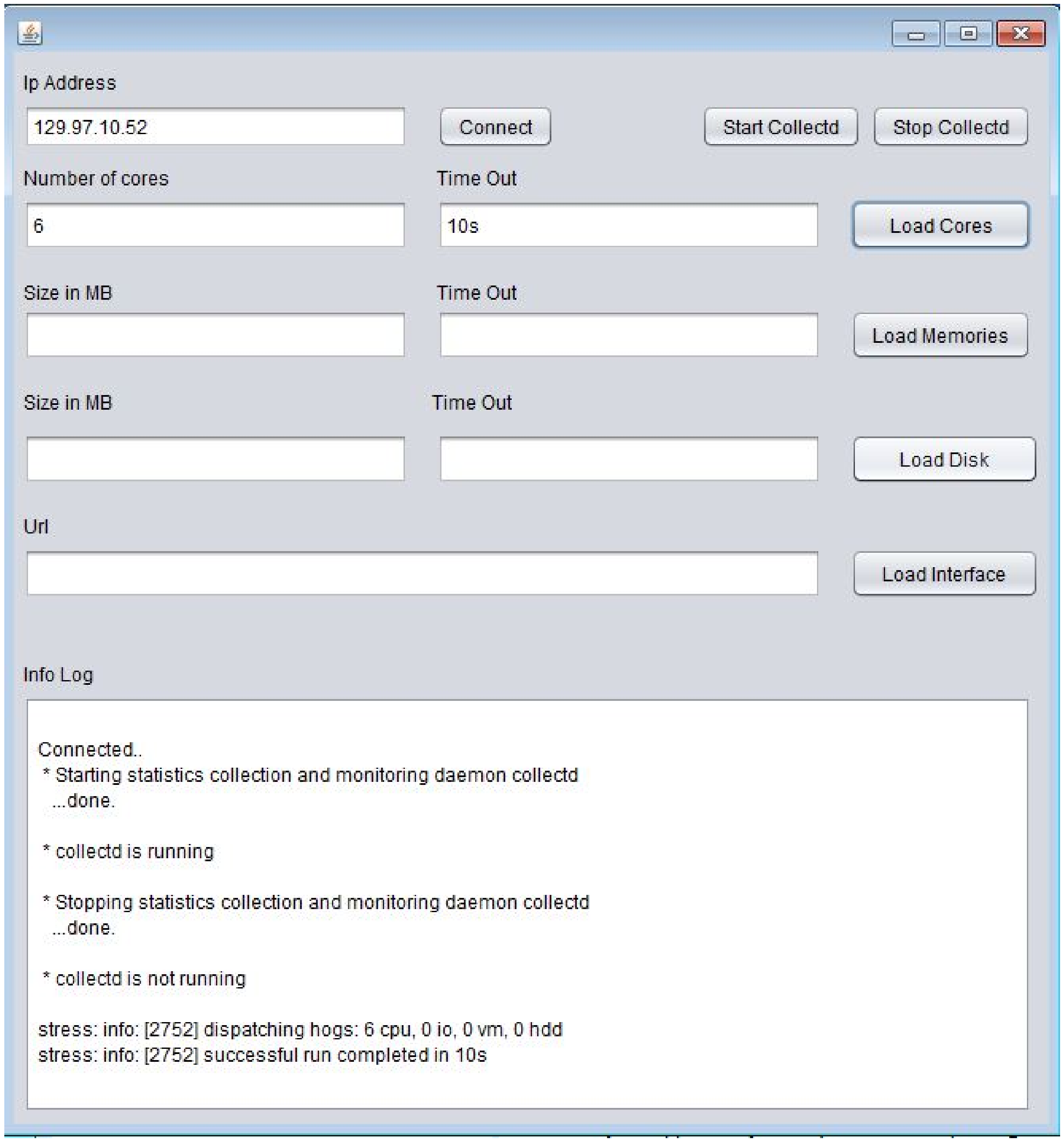
Tool	Description
	Phase 1: Server
Collectd	Collects the subsystem performance and stores the data in CSV format (explained in detail in Chapter 3).
Stress	Is a workload generator for POSIX systems. It imposes a configurable amount of CPU, Memory and Disk stress on the system [61] (explained in details in Section 4.2).
JDK	Java Development Kit is a program development environment for writing java applications. It provides a runtime environment that sits on top of the operating system layer so developers can compile, debug and run applications written in the Java language [62].
JRE	Java Runtime Environment is a set of programming tools for developing java applications. It provides the requirements for executing a java application[62].
Samba	This is a software suite that provides seamless file and print services to SMB/CIFS clients[63]. Samba uses the TCP/IP protocol that is installed on the host server where it runs Linux or Unix. It allows the host to interact with a Microsoft Windows client to share files or printer.
	Phase 2: Client
Matlab	This is a high-level language and interactive environment [64]. Matlab has been used for collecting and simplifying the data. It can also be used to create models and estimate the power consumption of various applications.
JDK	Explained above. JDK in the server and in the client should be the same version.
JRE	Explained above. JRE in the server and in the client should be the same version.

6.2 Phase 1: Client/Server GUI

The first phase of the test bench is the Client/server Graphical User Interface (CS-GUI). Before running CS-GUI a connection should be made between the server and the development machine so that *Collectd* data can be shared between the server and the client side. This step can be done by using Sampa service that allows sharing files between Windows and Linux since our server has Linux OS and our development machine has Windows OS. Using Sampa service is safe because a username and password must be entered before accessing the data from the development machine. The CS-GUI is shown in Figure 6.3. Also, our code has two parts, one on the client side and the other on the server side.

In this phase, the user from the development machine can enter the IP Address of the server that he/she wants to develop a model for. Then, there are two bottoms where *Collectd* can be started or stopped. The importance of these bottom is to start/stop Collected from collecting the system utilization. The second step is loading the system. Here, there are four subsystems that can be loaded. Loading the CPU is done by using the tool, Stress, where the number of cores of the CPU to stress can be selected and for how long. Also, for the Memory and the Disk Stress being used, we can select how much of the subsystem to stress and for how long. However, stressing the Interface can be done by downloading large files from the Internet where we can enter the Url and click load interface. Moreover, for every bottom, the server response will be showing the Info Log as shown in Figure 6.3.

The message sequence of this phase is shown in Figure 6.4. TED500 is the power meter connected to the server. *Collectd* is also installed in the same server. The first step is connecting to the server. Then *Collectd* can start to collect the subsystem performance. Now the process of stressing the selected subsystems can be started. After finishing the stress, *Collectd* can be stopped. After this step, the data of the subsystem and the actual power has been saved by *Collectd* in a folder in the server, and this folder is shared with the development machine. Now the data in the development machine is ready for the next phase of the test bench.



Ip Address: 129.97.10.52

Number of cores: 6

Time Out: 10s

```
Connected..
 * Starting statistics collection and monitoring daemon collectd
   ...done.

 * collectd is running

 * Stopping statistics collection and monitoring daemon collectd
   ...done.

 * collectd is not running

stress: info: [2752] dispatching hogs: 6 cpu, 0 io, 0 vm, 0 hdd
stress: info: [2752] successful run completed in 10s
```

Figure 6.3: The GUI for the Client/Server tool

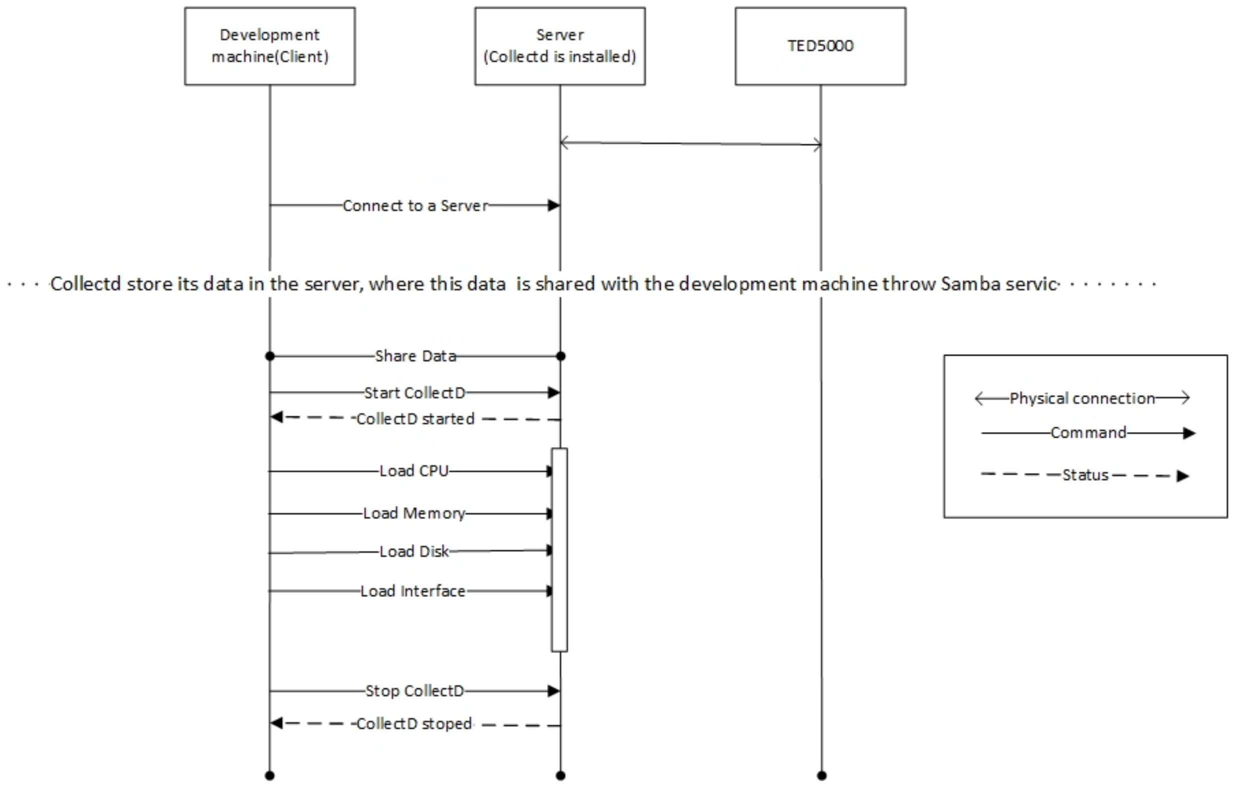

Development machine(Client)
Server (Collectd is installed)
TED5000
Connect to a Server
Collectd store its data in the server, where this data is shared with the development machine throw Samba servic
Share Data
Start CollectD
CollectD started
Load CPU
Load Memory
Load Disk
Load Interface
Stop CollectD
CollectD stoped
Physical connection
Command
Status

Figure 6.4: The message sequence of the Client/Server tool

6.3 Phase 2: modelling GUI

The next phase of the test bench includes three parts as shown in Figure 6.6 (Data Collection, Creating the Models, and Estimation). Since there are three real servers used in these experiments, the first step in the modelling Graphical User Interface (M-GUI) is to choose the server. Each server has a name such as NV21. The explanation of the three parts of M-GUI is as follows:

1. Data Collection Part: This part has two bottoms that import the data and simplify it as follows:

   - Import & Simplify Data Bottom: when clicking this bottom the data will be imported from the shared data of *Collectd* and the data will be simplified as explained in Chapter 3.
   - Add Data for the Model Bottom: The importance of this bottom is that more data can be added to the previous data at different times. This means that each subsystem can be stressed separately.

2. Creating the Model Part: This is where different techniques of modelling based on the collected data can be created. The two techniques of modelling are: Neural Network and Linear Regression.

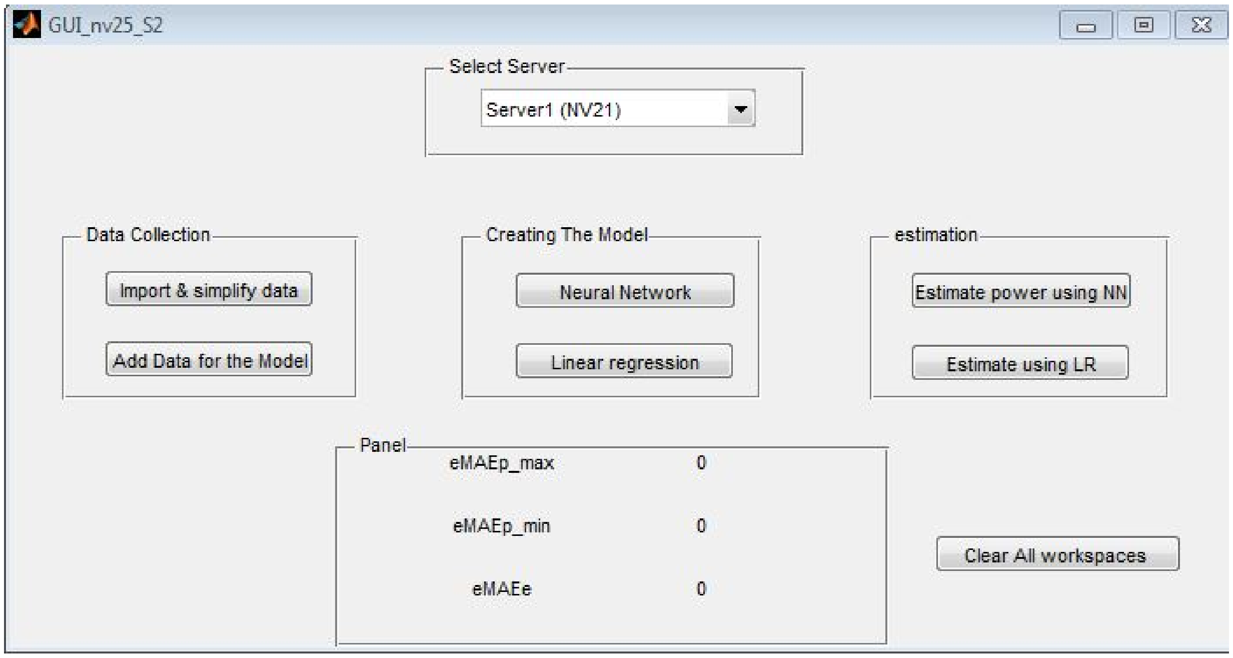
3. Estimation Part: After running an application on the server that has been chosen, the power consumption of this application can be estimated using the two models that were created in the previous part. The power can be estimated using the NN model or LR model. The Errors of the estimation power will be shown in M-GUI.

4. Clear All workspaces bottom: This bottom will erase all the data and the models that have been imported and created.

The message sequence of M-GUI is shown in Figure 6.6. This message sequence has been divided into three phases as follows:

1. Phase a: Creating the models: In this phase, the first command will be used to import the data from *Collectd* folder into the workload workspace, where this includes the simplification. The second command will train the data to create an NN or LR model. The created models will be imported to NN Model workspace and LR Model workspace.

2. Phase b: Estimate the power consumption of application A by NN technique: The only command in this phase will be 'estimate by NN'. That means that Application A's data will be imported from Collected folder to Application A workspace and the NN model from NN workspace. After estimating the power consumption of Application A the errors will be imported to MAEe & MAEp workspace.



eMAEp_max	0
eMAEp_min	0
eMAEe	0

Figure 6.5: The GUI for the modelling tool

3. Phase c: Estimate the power consumption of application A by LR technique: The only command in this phase will be 'estimate by LR'. That means that Application A's data will be imported from Collectd folder to Application A workspace and the LR model from LR workspace. After estimating the power consumption of Application A, the errors will be imported to MAEe & MAEp workspace.

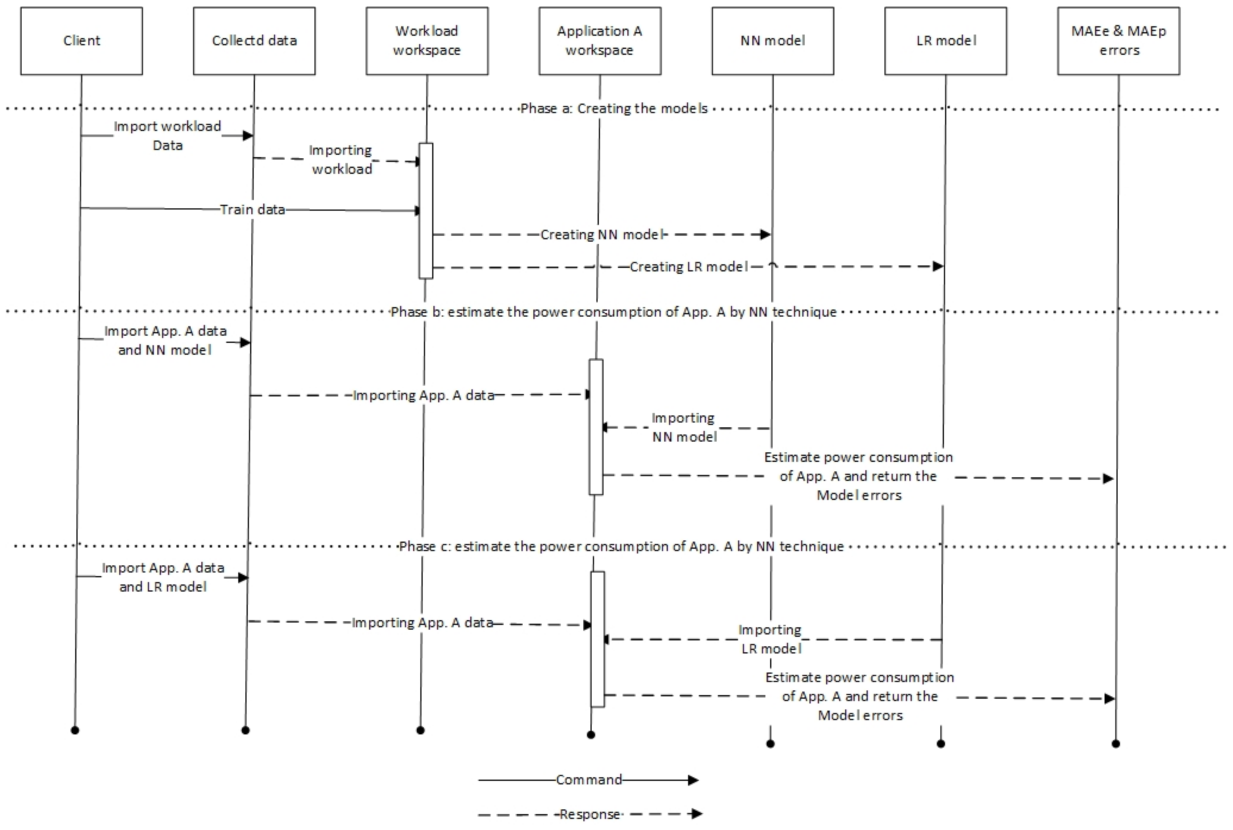

Client
Collectd data
Workload workspace
Application A workspace
NN model
LR model
MAEe & MAEp errors
Phase a: Creating the models
Import workload Data
Importing workload
Train data
Creating NN model
Creating LR model
Phase b: estimate the power consumption of App. A by NN technique
Import App. A data and NN model
Importing App. A data
Importing NN model
Estimate power consumption of App. A and return the Model errors
Phase c: estimate the power consumption of App. A by NN technique
Import App. A data and LR model
Importing App. A data
Importing LR model
Estimate power consumption of App. A and return the Model errors
Command
Response

Figure 6.6: The message sequence of the modelling tool

Chapter 7

Conclusion and Future Work

In this thesis, we have designed a test bench for modelling the power consumption in servers. In addition, we designed two workloads for modelling (Workload A and Workload B). Workload A has four loading steps, which are loading the main subsystems (CPU, Memory, Disk, and Interface). Further, in Workload A, each subsystem has been loaded for a short time and all the subsystems took 64 minutes to load. On the other hand, Workload B has seven steps of loading. In addition to the four subsystems, we designed three more loads as follows:(i) Idle load, where we run the server for two hours and keep it idle from the user prospective. (ii) Daily activity load, where we simulate the daily activity that occurred in the servers such as running some applications. (iii) Load all, where we load all the four subsystems (the CPU, memory, Disk, and Interface) in addition to the daily activity load. Workload B has long time loading, which took a total of 390 minutes.

Further, two modelling techniques have been used for training the data: Neural Network and Linear Regression. We created models for three real servers from a data center, with different configurations. These models have been validated with real loads by four scenarios on each server. In conclusion, Neural Network and Linear regression techniques perform better in creating the models based on Workload B. Neural Network technique has fewer errors compared to Linear regression technique in two out of three servers.

Further, we showed the importance of subsystem selection for modelling. The developer's selection of the subsystems will have an direct impact on the accuracy of the models. Finally, we have automated our test bench and explained the importance of the automation for application developers as it reduces the complexity of the modelling process. We have also explained in detail how our automation test bench can be used.

Future Work:
Problems arise when we installed *Collectd* on a real server running Windows OS. On linux,

Collectd has about 90 plugins, but on Windows there are only nine plugins. One of the most important missing plugins is the TED plugin. There is also no way to writing a new plugin on the Windows, whereas on linux, the user has a choice to write a plugin. Thus we cannot connect the power meter TED to *Collectd* on Windows platform. In addition, *Collectd* on Windows does not support saving the performance utilization in CSV format. These limitations limit our test bench as ability to work across different operating systems. In the future, we are aiming to transcend these limitations by developing a tool on the Windows platform that can collect subsystems utilization. Moreover, creating such models for different machine will be considered in the future work. Creating these models in Desktops and Smartphones will help developers for creating more power-aware's softwares.

APPENDICES

Appendix A

All the Error Tables Used for Comparing Neural Network and Linear Regression Models in Chapter 4

Table A.1: MAEp and MAEe errors of models in Server 1.

Server 1, (The server configuration is shown in table 4.3)					
		Model A		Model B	
		Neural Network	Linear Regression	Neural Network	Linear Regression
Download	MAEp(Min,Max)	19.24%,15.89%	9.54%, 11.72%	2.36%,1.92%	3.32%, 2.70%
	MAEe	12.64%	10.60%	0.07%	-2.56%
Gaming	MAEp(Min,Max)	14.42%,11.74%	18.18%, 15.02%	2.98%,2.46%	7.11%, 5.87%
	MAEe	13.04%	15.51%	2.39%	6.01%
Local movie	MAEp(Min,Max)	17.11%,12.99%	17.95%, 13.63%	2.56%,1.94%	5.50%, 4.18%
	MAEe	13.67%	14.34%	1.89%	4.32%
Youtube	MAEp(Min,Max)	17.87%,14.35%	17.52%, 14.06%	2.79%,2.24%	5.86%, 4.71%
	MAEe	15.07%	14.77%	1.83%	4.30%

Table A.2: MAEp and MAEe errors of models in Server 2.

Server 2, (The server configuration is shown in table 4.4)					
		Model A		Model B	
		Neural Network	Linear Regression	Neural Netwrok	Linear Regression
Download	MAEp(Min,Max)	3.20%,2.43%	13.99%, 10.62%	3.19%,2.42%	4.08%, 3.09%
	MAEe	0.62%	10.92%	0.14%	0.01%
Gaming	MAEp(Min,Max)	6.52%,5.14%	4.74%, 3.73%	2.74%,2.16%	6.36%, 5.02%
	MAEe	5.43%	3.90%	2.05%	-5.26%
Local movie	MAEp(Min,Max)	6.22%,4.89%	4.74%, 3.72%	4.69%,3.69%	13.20%, 10.38%
	MAEe	4.98%	3.78%	-3.33%	-10.99%
Youtube	MAEp(Min,Max)	7.88%,5.99%	5.14%, 3.91%	3.58%,2.72%	8.09%, 6.15%
	MAEe	5.95%	3.67%	1.70%	-5.99%

Table A.3: MAEp and MAEe errors of models in Server 3.

Server 3, (The server configuration is shown in table 4.5)					
		Model A		Model B	
		Neural Network	Linear Regression	Neural Netwrok	Linear Regression
Download	MAEp(Min,Max) MAEe	3.98%,3.13% -3.12%	4.37%, 3.44% -3.28%	8.33%,6.56% 3.67%	9.79%, 7.71% 7.98%
Gaming	MAEp(Min,Max) MAEe	34.76%,3.89% -4.30%	4.01%, 3.28% -3.60%	0.87%,0.71% 0.40%	0.85%, 0.69% 0.46%
Local movie	MAEp(Min,Max) MAEe	8.31%,6.35% -6.11%	2.46%, 1.88% 0.23%	3.83%,2.93% -0.99%	3.04%, 2.32% 1.46%
Youtube	MAEp(Min,Max) MAEe	6.74%,5.43% -5.78%	2.65%, 2.13% -2.12%	4.44%,3.57% 3.50%	2.40%, 1.93% 1.75%

Appendix B

All the Error Tables Used for Comparing Different Inputs for Creating Models in Chapter 5

Table B.1: Comparing different model input in Server 1 with Workload A

server 1 (NV21) short model (model A)									
		Download		Gaming		Local movie		Youtube	
Technique	Model Input	MAEp(Min,Max)	MAEe	MAEp(Min,Max)	MAEe	MAEp(Min,Max)	MAEe	MAEp(Min,Max)	MAEe
NN	C	16.53%, 13.46%	14.96%	25.31%, 20.91%	21.58%	32.93%, 25%	26.31%	29.30%, 23.53%	24.71%
LR		10.93%, 8.90%	9.89%	17.65%, 14.58%	15.05%	17.34%, 13.17%	13.86%	16.90%, 13.57%	14.25%
NN	C,M	12.83%, 10.45%	11.60%	17.82%, 14.72%	15.19%	14.64%, 11.11%	11.70%	16.50%, 13.25%	13.91%
LR		11.56%, 9.41%	10.45%	18.65%, 15.40%	15.90%	19.05%, 14.46%	15.22%	18.23%, 14.64%	15.37%
NN	C,D	12.24%, 9.97%	11.06%	17.15%, 14.17%	14.62%	13.16%, 9.99%	10.51%	14.31%, 11.49%	12.07%
LR		12.07%, 9.83%	10.92%	12.07%, 9.83%	10.92%	18.39%, 13.96%	14.69%	17.93%, 14.40%	15.12%
NN	C,I	13.99%, 11.39%	12.64%	17.47%, 14.43%	14.90%	11.26%, 8.55%	9.00%	13.45%, 10.80%	11.34%
LR		10.40%, 8.47%	9.41%	16.79%, 13.87%	14.32%	16.68%, 12.66%	13.33%	16.27%, 13.07%	13.72%
NN	C,M,D	12.69%, 10.34%	11.47%	16.74%, 13.82%	14.27%	11.01%, 8.36%	8.80%	12.62%, 10.14%	10.64%
LR		9.91%, 8.07	8.96%	16.91%, 13.97%	14.42%	16.99%, 12.90%	13.57%	16.38%, 13.15%	13.82%
NN	C,M,I	14.07%, 11.46%	12.72%	17.07%, 14.10%	14.55%	12.05%, 9.15%	9.63%	13.36%, 10.73%	11.27%
LR		83.64%, 68.13%	-75.68%	96.94%, 80.08%	-82.66%	1.43%, 1.09%	-1.15%	1.35%, 1.09%	-1.14%
NN	C,M,D,I	19.24%, 15.89%	12.64%	14.42%, 11.74%	13.04%	17.11%, 12.99%	13.67%	17.87%, 14.35%	15.07%
LR		9.54%, 11.72%	10.60%	18.18%, 15.02%	15.51%	17.95%, 13.63%	14.34%	17.52%, 14.06%	14.77%

Table B.2: Comparing different model input in Server 1 with Workload B

server 1 (NV21) Long model (model B)									
		Download		Gaming		Local movie		Youtube	
Technique	Model Input	MAEp(Min,Max)	MAEe	MAEp(Min,Max)	MAEe	MAEp(Min,Max)	MAEe	MAEp(Min,Max)	MAEe
NN	C	2.75%, 2.24%	-0.35%	1.97%, 1.63%	1.48%	1.74%, 1.32%	0.22%	1.87%, 1.50%	0.20%
LR		2.10%, 1.71%	0.40%	7.19%, 5.94%	6.08%	6.02%, 4.57%	4.72%	6.13%,4.96%	5.17%
NN	C,M	11.24%, 9.16%	-10.11%	6.27%, 5.18%	-5.33%	3.36%, 2.55%	-2.54%	3.42%, 2.57%	-2.65%
LR		6.18%, 4.94%	5.17%	6.18%, 4.94%	5.17%	6.18%, 4.96	5.17%	6.18%, 4.96%	5.17%
NN	C,D	2.91%, 2.37%	-1.86%	2.10%, 1.73%	1.60%	1.49%, 1.13%	-0.52%	2.02%, 1.62%	0.72%
LR		2.15%, 1.75%	0.77%	7.55%, 6.24%	6.39%	6.26%, 4.75%	4.93%	6.49%, 5.15%	5.42%
NN	C,I	4.81%, 3.91%	-1.67%	2.32%, 1.92%	1.82%	1.54%, 1.17%	-0.42%	2.84%, 2.28%	0.53%
LR		5.31%, 4.33%	-4.49%	7.33%, 6.05%	6.20%	6.11%, 4.64%	4.80%	6.57%, 5.28%	4.27%
NN	C,M,D	2.58%, 2.10%	1.34%	6.12%, 5.05%	5.17%	6.40%, 4.86%	5.05%	5.73%, 4.60%	4.79%
LR		2.05%, 1.67%	0.49%	7.09%, 5.86%	6.00%	5.51%, 4.18%	4.32%	5.87%, 4.71%	4.91%
NN	C,M,I	18.46%, 15.04%	-15.86%	2.80%, 2.31%	2.26%	1.99%, 1.51%	1.31%	3.88%, 3.12%	0.85%
LR		6.98%, 5.69%	-6.06%	6.99%, 5.78%	5.91%	5.60%, 4.25%	4.38%	6.45%, 5.18	3.65%
NN	C,M,D,I	2.36%,1.92%	0.07%	2.98%,2.46%	2.39%	2.56%,1.94%	1.89%	2.79%,2.24%	1.83%
LR		3.32%, 2.70%	-2.56%	7.11%, 5.87%	6.01%	5.50%, 4.18%	4.32%	5.86%, 4.71%	4.30%

Table B.3: Comparing different model input in Server 2 with Workload A

server 2 (NV25) short model (model A)									
		Download		Gaming		Local movie		Youtube	
Technique	Model Input	MAEp(Min,Max)	MAEe	MAEp(Min,Max)	MAEe	MAEp(Min,Max)	MAEe	MAEp(Min,Max)	MAEe
NN	C	19.50%, 14.81%	-10.59%	10.46%, 8.52%	8.78%	7.09%, 5.58%	5.70%	8.82%, 6.70%	6.88%
LR		6.16%, 4.68%	-0.82%	1.29%, 1.02%	-0.98%	1.21%, 0.95	0.01%	1.50%, 1.14%	-0.81%
NN	C,M	10.56%, 8.02%	9.52%	11.60%, 9.15%	9.74%	7.88%, 6.20%	6.41%	11.01%, 8.37%	8.62%
LR		6.12%, 4.56%	-0.87%	1.31%, 1.03%	-1.00%	1.20%, 0.94%	-0.13%	1.56%, 1.19%	-0.89%
NN	C,D	10.09%, 7.66%	-0.78%	9.84%, 7.76%	7.89%	6.87%, 5.40%	4.58%	11.22%, 8.53%	7.29%
LR		5.89%, 4.47%	-0.43%	0.77%, 0.60%	-0.45%	1.31%, 1.03%	0.43%	1.19%, 0.90%	-0.33%
NN	C,I	3.45%, 2.62%	-0.15%	6.47%, 5.10%	5.37%	4.39%, 3.45%	3.40%	8.92%, 6.79%	6.60%
LR		3.52%, 2.67%	0.94%	4.05%, 3.19%	3.29%	4.01%, 3.16%	3.14%	4.45%, 3.38%	3.04%
NN	C,M,D	8.18%, 6.21%	5.87%	8.98%, 7.08%	7.51%	4.85%, 3.81%	3.83%	8.70%, 6.61%	6.74%
LR		5.72%, 4.35%	-0.42%	0.68%, 0.54%	-0.35%	1.22%, 0.96%	0.22%	1.20%, 0.91%	-0.37%
NN	C,M,I	3.34%, 2.53%	-0.20%	5.42%, 4.28%	4.49%	6.59%, 5.19%	5.28%	7.87%, 5.98%	5.93%
LR		3.55%, 2.69%	1.11%	4.31%, 3.40%	3.53%	4.52%, 3.56%	3.60%	4.81%, 3.66	3.39%
NN	C,M,D,I	3.20%, 2.43%	0.62%	6.52%, 5.14%	5.43%	6.22%, 4.89%	4.98%	7.88%, 5.99%	5.95%
LR		13.99%, 10.62%	10.92%	4.74%, 3.73%	3.90%	4.74%, 3.72%	3.78%	5.14%, 3.91%	3.67%

Table B.4: Comparing different model input in Server 2 with Workload B

server 2 (NV25) Long model (model B)									
		Download		Gaming		Local movie		Youtube	
Technique	Model Input	MAEp(Min,Max)	MAEe	MAEp(Min,Max)	MAEe	MAEp(Min,Max)	MAEe	MAEp(Min,Max)	MAEe
NN	C	20.93%, 15.90%	-15.98%	7.04%, 5.55%	-5.73%	5.99%, 4.71%	-1.43%	8.85%, 6.73%	-2.06%
LR		6.39%, 4.85%	5.58%	5.93%, 4.68%	4.90%	4.22%, 3.32	3.29%	5.74%, 4.36%	4.25%
NN	C,M	10.72%, 8.14%	9.61%	4.74%, 3.74%	3.88%	4.55%, 3.58%	-2.52%	3.90%, 2.96%	2.87%
LR		6.27%, 4.76%	5.45%	5.83%, 4.60%	4.81%	3.90%, 3.07%	2.99%	5.55%, 4.22%	4.06%
NN	C,D	7.26%, 5.51%	6.16%	8.52%, 6.72%	7.06%	5.32%, 4.19%	3.84%	9.36%, 7.12%	7.16%
LR		6.39%, 4.85%	5.58%	5.93%, 4.67%	4.90%	4.22%, 3.32%	3.29%	5.74%, 4.36%	4.25%
NN	C,I	4.14%, 3.14%	1.45%	4.40%, 3.47%	3.42%	3.54%, 2.79%	1.99%	2.82%, 2.15%	1.18%
LR		6.40%, 4.86%	5.58%	5.93%, 4.68%	4.90%	4.22%, 3.32%	3.29%	5.74%, 4.37%	4.25%
NN	C,M,D	9.02%, 6.85%	6.39%	8.66%, 6.83%	7.19%	1.85%, 1.45%	0.95%	3.95%, 3%	2.74%
LR		6.27%, 4.76%	5.45%	5.83%, 4.60%	4.81%	3.90%, 3.07%	2.99%	5.55%, 4.22%	4.06%
NN	C,M,I	2.92%, 2.21%	0.21%	15.43%, 12.10%	-12.80%	13.35%, 10.50%	-11.14%	6.67%, 5.07%	-4.55%
LR		6.28%, 4.77%	5.45%	5.83%, 4.60%	4.81%	3.90%, 3.07%	2.99%	5.55%, 4.22%	4.06%
NN	C,M,D,I	3.15%, 2.39%	0.33%	14.12%, 11.14%	-11.77%	4.49%, 3.53%	-3.37%	12.07%, 9.18%	-8.86%
LR		4.08%, 3.09%	0.01%	6.36%, 5.02%	-5.26%	13.20%, 10.38%	-10.99%	8.09%, 6.15%	-5.99%

References

[1] Trevor Mudge. Power: A first-class architectural design constraint. *Computer IEEE*, (4):52–58, 2001.

[2] Kshirasagar Naik and David SL Wei. Software implementation strategies for power-conscious systems. *Mobile Networks and Applications*, 6(3):291–305, 2001.

[3] Christoph Mobius, Waltenegus Dargie, and Alexander Schill. Power consumption estimation models for processors, virtual machines, and servers. *IEEE Tran. on Parallel and Distributed Systems,*, 25(6):1600–1614, 2014.

[4] Chang Ge, Zhili Sun, and Ning Wang. A survey of power-saving techniques on data centers and content delivery networks. *Communications Surveys & Tutorials, IEEE*, 2013.

[5] Raffaele Bolla, Roberto Bruschi, Franco Davoli, and Flavio Cucchietti. Energy efficiency in the future internet: a survey of existing approaches and trends in energy-aware fixed network infrastructures. *Communications Surveys & Tutorials, IEEE*, 13(2):223–244, 2011.

[6] Aruna Prem Bianzino, Claude Chaudet, Dario Rossi, and Jean-Louis Rougier. A survey of green networking research. *Communications Surveys & Tutorials, IEEE*, 14(1):3–20, 2012.

[7] Yuan Yao, Longbo Huang, Abhishek B Sharma, Leana Golubchik, and Michael J Neely. Power cost reduction in distributed data centers: A two-time-scale approach for delay tolerant workloads. *IEEE Tran. on Parallel and Distributed Systems,*, 2014.

[8] Adam Wierman, Lachlan LH Andrew, and Ao Tang. proc. in sharing systems power-aware speed scaling. In *INFOCOM*, pages 2007–2015. IEEE, 2009.

[9] Anshul Gandhi, Varun Gupta, Mor Harchol-Balter, and Michael A Kozuch. Optimality analysis of energy-performance trade-off for server farm management. *Performance Evaluation*, 67(11):1155–1171, 2010.

[10] Asfandyar Qureshi, Rick Weber, Hari Balakrishnan, John Guttag, and Bruce Maggs. Cutting the electric bill for internet-scale systems. In *SIGCOMM computer communication review*, volume 39, pages 123–134. ACM, 2009.

[11] Manuj Sabharwal, Abhishek Agrawal, and Grace Metri. Enabling green it through energy-aware software. *IT Professional*, (1):19–27, 2013.

[12] David J Brown and Charles Reams. Toward energy-efficient computing. *Communications of the ACM*, 53(3):50–58, 2010.

[13] Kshirasagar Naik. *A survey of software based energy saving methodologies for handheld wireless communication devices*. Department of Electrical and Computer Engineering, University of Waterloo, 2010.

[14] Frank Bellosa. The benefits of event: driven energy accounting in power-sensitive systems. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*, pages 37–42, 2000.

[15] W Lloyd Bircher, Madhavi Valluri, Jason Law, and Lizy K John. Runtime identification of microprocessor energy saving opportunities. In *Proc. ISLPED'05.*, pages 275–280. IEEE, 2005.

[16] Canturk Isci and Margaret Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proc. 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 93. IEEE, 2003.

[17] Tao Li and Lizy Kurian John. Run-time modeling and estimation of operating system power consumption. *ACM SIGMETRICS Performance Evaluation Review*, 2003.

[18] W Lloyd Bircher and Lizy K John. Complete system power estimation using processor performance events. *IEEE Transactions on Computers,*, 2012.

[19] Karan Singh, Major Bhadauria, and Sally A McKee. Real time power estimation and thread scheduling via performance counters. *ACM SIGARCH Computer Architecture News*, 2009.

[20] Fadwa Abdulhalim, Omar Alghamdi, and Kshirasagar Naik. Modelling the energy performance of application software for deverlopers. In *Proc. of SERP*, 2015.

[21] Pierre Delforge. America's data centers consuming and wasting growing amounts of energy. *Natural Resource Defence Councle*, 2014.

[22] Pierre Delforge. New study: America's data centers consuming - and wasting - growing amounts of energy. http://switchboard.nrdc.org/blogs/pdelforge/. [Online; accessed October-2015].

[23] Chris Thompson, Douglas Schmidt, Hamilton Turner, and Jules White. Analyzing mobile application software power consumption via model-driven engineering. 2011.

[24] Youngjin Cho, Younghyun Kim, Sangyoung Park, and Naehyuck Chang. System-level power estimation using an on-chip bus performance monitoring unit. In *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2008.

[25] Gilberto Contreras and Margaret Martonosi. Power prediction for intel xscale® processors using performance monitoring unit events. In *Proc. ISLPED'05.*, 2005.

[26] Ramon Bertran, Marc Gonzalez, Xavier Martorell, Nacho Navarro, and Eduard Ayguade. Decomposable and responsive power models for multicore processors using performance counters. In *Proc. 24th ACM International Conference on Supercomputing*, 2010.

[27] Russ Joseph and Margaret Martonosi. Run-time power estimation in high performance microprocessors. In *Proc. 2001 international symposium on Low power electronics and design*, 2001.

[28] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mary Jane Irwin, Narayanan Vijaykrishnan, and Mahmut Kandemir. Using complete machine simulation for software power estimation: The softwatt approach. In *Proc. 8th International Symposium on High-Performance Computer Architecture,*, 2002.

[29] Jasmeet Singh, Veluppillai Mahinthan, and Kshirasagar Naik. Automation of energy performance evaluation of software applications on servers. In *Proc. of SERP*, 2014.

[30] Taliver Heath, Bruno Diniz, Enrique V Carrera, Wagner Meira Jr, and Ricardo Bianchini. Energy conservation in heterogeneous server clusters. In *Proc. 10th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 186–195. ACM, 2005.

[31] Dimitris Economou, Suzanne Rivoire, Christos Kozyrakis, and Partha Ranganathan. Full-system power analysis and modeling for server environments. In *International Symposium on Computer Architecture*. IEEE, 2006.

[32] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 13–23. ACM, 2007.

[33] Zahra Abbasi, Georgios Varsamopoulos, and Sandeep KS Gupta. Thermal aware server provisioning and workload distribution for internet data centers. In *Proc. 19th ACM International Symposium on High Performance Distributed Computing*, pages 130–141. ACM, 2010.

[34] Suzanne Rivoire, Parthasarathy Ranganathan, and Christos Kozyrakis. A comparison of high-level full-system power models. *HotPower*, 8:3–3, 2008.

[35] Vinicius Petrucci, Enrique V Carrera, Orlando Loques, Julius CB Leite, and Daniel Mosse. Optimized management of power and performance for virtualized heterogeneous server clusters. In *11th IEEE/ACM International Symposium on CCGrid,*, pages 23–32. IEEE, 2011.

[36] Linux/unix sar command. "http://www.computerhope.com/unix/usar.htm". [Online; accessed October-2015].

[37] Anton Beloglazov and Rajkumar Buyya. Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers. In *Proc. 8th International Workshop on Middleware for Grids, Clouds and e-Science*, volume 4. ACM, 2010.

[38] Ata E Husain Bohra and Vipin Chaudhary. Vmeter: Power modelling for virtualized clouds. In *Proc. (IPDPSW), IEEE International Symposium on Parallel & Distributed*, pages 1–8. Ieee, 2010.

[39] Gaurav Dhiman, Kresimir Mihic, and Tajana Rosing. A system for online power prediction in virtualized environments using gaussian mixture models. In *47th ACM/IEEE Design Automation Conference (DAC),*, pages 807–812. IEEE, 2010.

[40] Aman Kansal, Feng Zhao, Jie Liu, Nupur Kothari, and Arka A Bhattacharya. Virtual machine power metering and provisioning. In *Proc. 1st ACM symposium on Cloud computing*, pages 39–50. ACM, 2010.

[41] Qian Zhu, Jiedan Zhu, and Gagan Agrawal. Power-aware consolidation of scientific workflows in virtualized environments. In *Proc. ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2010.

[42] Bhavani Krishnan, Hrishikesh Amur, Ada Gavrilovska, and Karsten Schwan. Vm power metering: feasibility and challenges. *ACM SIGMETRICS Performance Evaluation Review*, 38(3):56–60, 2011.

[43] Van Bui, Boyana Norris, Kevin Huck, Lois Curfman McInnes, Li Li, Oscar Hernandez, and Barbara Chapman. A component infrastructure for performance and power modeling of parallel scientific applications. In *Proc. HPC-GECO workshop on Component based high performance*, page 6. ACM, 2008.

[44] Xi Chen, Chi Xu, Robert P Dick, and Zhuoqing Morley Mao. Performance and power modeling in a multi-programmed multi-core environment. In *Proc. 47th Design Automation Conference*, pages 813–818. ACM, 2010.

[45] Daniel Molka, Daniel Hackenberg, Robert Schöne, and Matthias S Müller. Characterizing the energy consumption of data transfers and arithmetic operations on x86- 64 processors. In *Green Computing Conference,*, pages 123–133. IEEE, 2010.

[46] Ramon Bertran, Yolanda Becerra, David Carrera, Vicenç Beltran, Marc Gonzàlez, Xavier Martorell, Nacho Navarro, Jordi Torres, and Eduard Ayguadé. Energy accounting for shared virtualized environments under dvfs using pmc-based power models. *Future Generation Computer Systems*, 28(2):457–468, 2012.

[47] Klaus-Dieter Lange. Identifying shades of green: The specpower benchmarks. *Computer*, (3):95–97, 2009.

[48] Weiping Liao, Lei He, and Kevin M Lepak. Temperature and supply voltage aware performance and power modeling at microarchitecture level. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,*, 24(7):1042–1053, 2005.

[49] Perfmon2. http://perfmon2.sourceforge.net. [Online; accessed October-2015].

[50] Larry W McVoy, Carl Staelin, et al. lmbench: Portable tools for performance analysis. In *USENIX annual technical conference*, pages 279–294. San Diego, CA, USA, 1996.

[51] Standard performance evaluation corporation. https://www.spec.org/cpu2006/. [Online; accessed October-2015].

[52] Vishal Aslot and Rudolf Eigenmann. Performance characteristics of the spec omp2001 benchmarks. *ACM SIGARCH Computer Architecture News*, 29(5):31–40, 2001.

[53] W. Saphir R. Van der Wijngaart A. Woo D. Bailey, T. Harris and M. Yarrow. *The NAS parallel benchmarks 2.0.* NASA Ames Research Center, 1995.

[54] Danesh Tafti. Genidlest a parallel high performance computational infrastructure for simulating complex turbulent flow and heat transfer. In *APS Division of Fluid Dynamics Meeting Abstracts*, volume 1, 2002.

[55] collectd – the system statistics collection daemon. https://collectd.org. [Online; accessed October-2015].

[56] Cisco WebEx Social Troubleshooting Guide. *Performance and Health Monitoring*, 3.0 edition, August 2012.

[57] Z Cihan Taysi, M Amac Guvensan, and Tommaso Melodia. Tinyears: spying on house appliances with audio sensor nodes. In *Proc. of the 2nd ACM WKsp on Embedded Sensing Systems for Energy-Efficiency in Building*, pages 31–36. ACM, 2010.

[58] SN Sivanandam and SN Deepa. *Introduction to neural networks using Matlab 6.0.* Tata McGraw-Hill Education, 2006.

[59] Kevin L Priddy and Paul E Keller. *Artificial neural networks: an introduction*, volume 68. SPIE Press, 2005.

[60] Samee Ullah Khan and Albert Y Zomaya. *Handbook on Data Centers.* Springer, 2015.

[61] Amos Waterland. Stress. http://people.seas.harvard.edu/ apw/stress/, 2014. [Online; accessed October-2015].

[62] http://searchsoa.techtarget.com. [Online; accessed October-2015].

[63] Samba. https://www.samba.org. [Online; accessed October-2015].

[64] Matlab. http://www.mathworks.com/products/matlab/. [Online; accessed October-2015].