THE APPLICATION OF SOFTWARE PRODUCT LINE ENGINEERING TO ENERGY MANAGEMENT IN THE CLOUD AND IN VIRTUALISED ENVIRONMENTS

by

I MADE MURWANTARA

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
The University of Birmingham
October 2016

Abstract

Modern software is created from components which can often perform a large number of tasks. For a given task, often there are many variations of components that can be used. As a result, software with comparable functionality can often be produced from a variety of components. The choice of software components influences the energy consumption. A popular method of software reuse with the components' setting selection is Software Product Line (SPL). Even though SPL has been used to investigate the energy related to the combination of software components, there has been no indepth study of how to measure the consumption of energy from a configuration of components and the extent to which the components contribute to energy usage. This thesis investigates how software components' diversity affects energy consumption in virtualised environments and it presents a method of identifying combinations of components that consume less energy. This work gives insight into the cultivation of the green software components by identifying which components influence the total consumption of energy. Furthermore, the thesis investigates how to use component diversity in a dynamic form in the direction of managing the consumption of energy as the demand on the system changes.

The contribution of the thesis can be divided into five parts. Firstly, the thesis introduces a method of measuring energy usage for each process in virtualised environments such as a Cloud system. Secondly, the thesis suggests a technique using various graph-based approaches to create a feature model from a large software repository. Thirdly, the thesis presents a method for predicting the consumption of energy from a large number of combinations of software components with the help of machine learning algorithms. Fourthly, the thesis offers a new approach called Energy Prediction Trees (EPTs) for dealing with the prediction of energy for a feature model that is retrieved from a large software repository. And finally, the thesis proposes a method to build a self-adaptive system that adjusts to the change of workload and energy usage by reconfiguring the

software architecture to have combinations of components that use less energy, in real-time. The suggested approaches are evaluated using case studies that are developed from a software repository and virtualised environments.

[John 15:5] "*I am the vine, ye are the branches: He that abideth in me, and I in him, the same bringeth forth much fruit: for without me ye can do nothing*".

This thesis is dedicated to my parents, Christina Yatinah and I Made Murka Wirana.

To my wife Andriyani, my daughters Vanya and Nathania, for their unconditional love.

To my brothers, I Wayan Muryantana and I Nyoman Murbawa Saputra.

ACKNOWLEDGEMENTS

First and foremost, praise and thanks goes to my savior Jesus Christ for his richest grace and blessing bestowed upon me.

I would like to acknowledge the support I have received from the Ministry of Research, Technology and Higher Education of the Republic of Indonesia and the Universitas Pelita Harapan.

I also would like to say a very big thank you to my supervisor Dr. Behzad Bordbar for all the support and encouragement he gave me. Without his guidance and constant feedback, this PhD would not have been achievable.

Many thanks also to the thesis group members (Dr. Steve Vickers, Dr. Rami Bahsoon and Dr. Peter Hancox) for their helpful discussions. Thank you to Prof. Irfan Awan and Dr. Robert Hendley for their comments on my thesis.

Thanks to Dr. Leandro L. Minku and Dr. João Bosco Ferreira Filho and to all my colleagues during my stay in Birmingham. The experience that I gained during those days was very worthwhile and enriched my life, socially and scientifically.

Thank you also to the Open Source communities and many research groups for making their tools available.

My deepest gratitude goes to my family for their unconditional love, prayers and support throughout my life; this PhD is simply impossible without them.

Finally, I am very blessed to enjoy life with the Indonesian Christian Community and Pengajian in Birmingham and Maria Gutama family for their constant support and prayers during my PhD journey.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

1.1 Introduction

In recent years, a rapid increase in the number of businesses that use Cloud computing for their main activities [144, 50], such as virtual offices and online stores, has established a large demand for data centres across the globe. A data centre, which typically hosts thousands of servers, consumes a large amount of electrical energy to power the servers [48, 213, 205]. When the energy to produce electricity is generated using fossil fuel (e.g. coal, oil and natural gas), it will lead to a large amount of Carbon Dioxide (CO_2) emissions. Furthermore, the CO_2 emissions [70] increase global warming - an increasing global average temperature at the earth's surface - that is argued to affect climate change [52] resulting in stronger hurricanes and severe heat waves – that reduce water supplies.

In 2013, the power consumption of data centres reached approximately 10% [48] of the worldwide power consumption. It is predicted that in the year 2020 [213] the power consumption of global data centres may increase to 100 Gigawatt. In the United States [205] alone, the data centres consumed 91 billion kilowatt-hours of energy in 2013 producing 97 million metric tonnes of CO_2, and it is projected that in the year 2020 they will consume 139 billion kilowatt-hours of energy producing 147 million metric tonnes of CO_2, a 53% increase, costing American businesses $13 billion per year of electricity bills [57]. In the United Kingdom, in the year 2012 the average energy bill to run a corporate data

centre was about £5.3 million per year [32]. Considering the on-going increase in the cost of energy, the efficient energy management of applications within the Cloud computing system is an active area of research [222, 228, 135, 25].

There are a number of ways to reduce energy consumption and to increase energy efficiency in Cloud systems. One method is to utilise renewable energy [93, 151, 136] which includes wind, solar and fuel cell as alternative energy sources. One of the largest Cloud providers claimed efficiency, using renewable energy, of 35% on its data centre operations [93]. By raising the temperature of the data centre to 80^oF and using outside air for cooling, Google [93] reduces 12% of the consumption of energy. A study shows [146] that a small increase of temperature for cooling a high-powered server reduces the consumption of energy, when compared to a low-power server. Therefore, it is not only using the renewable energy as a reason to have a green data centre, but also its level of operations temperature can give advantages in terms of the cost and the consumption of energy.

The location of data centres spreading all over the world may also take advantage of the electricity cost differences by using the smart-grid [158] - "the smart grid uses two-way flow of electricity and information to create an automated and distributed advanced energy delivery network" [74]. The Cloud providers can integrate [217] into the smart-grid network in a different region to seek to minimise their total energy cost, where the electricity prices and workload assignment may become an option of which, as an example, a data centre should operate a low workload.

Measurement is a key aspect of energy management. A typical large data centre uses thousands of sensors, which collect information such as temperature, electricity usage and server performance for monitoring purposes [84]. In order to support the reduction of energy, researchers have proposed techniques which work by predicting the performance of a data centre based on the sensors data to provide an opportunity to improve the energy efficiency [84, 133, 134]. In these techniques, a policy-based system uses information such as the energy costs, peak power costs, the impact on energy cooling costs and current

electricity prices – from the smart-grid network – to provide suggestions about whether or not to migrate the workload within a data centre to a different location.

Currently, autonomic computing [166, 46, 227, 35, 69] is widely used in the data centre for providing dynamic provision based on the change of environment, which is capable of scaling the infrastructure up and down. Such autonomic computing provides an integrated intelligent strategy to manage Cloud system services [35]. One of the autonomic computing approaches is a self-adaptive system which includes feedback and an estimation of response [118] to automate the mechanism within the system, where a Rule-Based System [11] may be involved in order to provide an accurate decision.

To have an energy efficient data centre, one of the requirements is to have better server technology [149, 229, 145, 88]. In a typical data centre, the servers consume 40% of the overall power [108]. Although turning off an idle server is the best option to reduce energy [145], the dynamic and rapid user demand within the Cloud computing system makes a system shut down difficult. One of the methods to reduce energy in a server is to adjust the Central Processing Units (CPUs) job-scheduling [88] that needs to monitor continuously the server power consumption in order to provide the data to manage this task. Another technique to reduce energy is via balancing power consumption [149] and minimising the total execution time [229], using an energy-aware scheduling policy to assign tasks to CPUs in order to increase the system's throughput. Furthermore, scheduling tasks [138, 2] to a minimum numbers of servers, while keeping the task response to its time constraints, will also reduce the consumption of energy.

Most existing solutions focus on hardware [146, 84], location [217], policy [229, 138, 2] and the cooling system [93] to reduce the consumption of energy. This thesis focuses on the role of software in the management of energy. With increases in the growth of Open Source software and public repositories, a software developer has options to create a system, such as a web application, that perform the same action from different combinations of software components. For example, we can build a web application using more than 20 different options of web (HTTP) server on top of the Ubuntu Server operating system, and it

will not change the mechanism regarding access to and performance of the system. As a consequence, this combination of components increases the possibility of having various applications by reusing the artefacts of the software products, within the same domain, that are members of a product line. Such a combination of software components may have commonality and variability, and with a selection technique, components can be reused to configure new software products known as Software Product Line (SPL) engineering [49, 27, 179].

Capra et al. show [37] that a different combination of software components with a similar job, such as web services, run in a similar operating system and execute similar tasks and deployed in a similar platform, may consume a different amount of electrical energy. This gives insight into the selection of software components that may affect the consumption of energy. Many approaches have been developed to reuse the software artefacts such as code or software libraries that can be deployed by organisations to meet their business needs, such as Object Oriented Programming (OOP) [72] and Software Product Line (SPL) [49, 27, 179].

Among these approaches, the SPL is receiving increasing attention as a reliable approach to compose the software components from a number of products' artefacts [62, 195, 194, 196]. A study carried out by Eriksson et al. [71] shows that the traditional software reuse approaches, such as OOP, has failed to fulfil its promise to increase the productivity, produce high quality software and reduce the budget, compared with the SPL that supports large-grained intra-organisation software reuse. This thesis is built on the existing SPL technique to identify combinations of software components that consume less energy.

As the cost of energy rises, it is crucial to adapt to the change of environment. In order to support dynamic systems, the Dynamic Software Product Line (DSPL) [104, 107, 36] - an SPL that is capable of adapting to the change of environment, user needs and evolving resource constraints - is also receiving attention. As an example, a web service has options to change its HTTP servers when the incoming request to the system has reached 100,

1000 or 10000 users per second. For this, the system should adapt to the change of the incoming workload to the system, and also provide a set of valid combinations of software components that allows the system to have options in response to the change of environment, with or without minimum human intervention. Therefore, a system that is developed using the DSPL approach should be able to monitor the information from the current running system as sensor data and provides action as feedback, simultaneously, in response to the sensor data.

The main thesis of this research is to develop methods to reduce the consumption of energy in a virtualised environment by identifying combinations of software components which consume less energy using a Software Product Line (SPL) development framework, and then derive methods for the reduction of energy for a dynamic system. This thesis raises three research questions, as follows:

(Q1) Virtualised environments such as the Cloud system is a shared infrastructure. In order to obtain knowledge of energy usage in virtualised environments, we need to develop a technique to measure energy in the virtual machine without the use of Wattmeter devices.

(Q2) A typical SPL model may have a large number of combinations of components. It is infeasible to measure the energy usage of all possible combinations. As a result, we need a method to approximate the consumption of energy for a large combination of components, such as combinations that are retrieved from a large software repository.

(Q3) In a typical Cloud system, workloads are dynamic which may influence the consumption of energy. As the cost of energy rises, we need a technique to deal with a dynamic system that is based on the SPL method for reducing the consumption of energy.

Measurement is an essential part of planning and management of energy consumption. Some techniques for the measurement of energy [228, 135, 25] make use of hardware devices commonly known as Wattmeters. An example of such a hardware devices is WattsUp [220]. It is also possible to use software products such as Intel Power Gadget [115] and Joulemeter [150] to measure energy usage of a running system. Most of these measurement

methods use workload to simulate the users activities when they interact with the system under test. However, there is a problem with the current measurement methods: these methods are not designed for measuring the energy consumption of individual processes of the running system. In virtualised environments, we cannot use Wattmeter devices to measure energy consumption per process of a virtual machine (VM) because the resources are shared and virtual machines can move from one host to another. In a shared infrastructure, it is difficult to isolate each VM and measure the energy consumed by the particular VM. In addition, VMs can be allocated to other hosts dynamically to allow load-balancing within the Cloud system. As a result, we need a method of measurement that is not reliant on Wattmeter devices.

In this thesis, we propose a technique to measure energy in the virtualised system which is software-based and combines software tools for measuring energy and the workload tool to generate a load to the system being measured. For example, we can measure energy consumed by different HTTP servers to find out which one is more efficient on energy usage and make a recommendation based on the measurement results.

To measure energy consumption of any given application, a suitable workload must be provided to simulate its usage. We use a workload generator tool, such as JMeter [6], to simulate the load of users incoming to a system. Under a given workload, each one of the processes within the system consumes a different amount of energy. We then capture the energy usage using a software power meter, Powertop [200]. This measurement method can be applied within a Laptop and a Cloud environment, which makes use of Advanced Control and Power Interface (ACPI) [152] that enables us to monitor the power usage in a Cloud system. This measurement method answers **Q1** , providing a technique to obtain the information of energy from a virtualised environment.

The above methods of measurement are suitable when dealing with a small number of features. For a large number of variations, there are two challenges, as follows:

1. A method is required that will automatically capture dependencies and variabilities within large SPL models. This is necessary because capturing such information

manually would be infeasibly time consuming when dealing with large SPL models. In addition, as repositories frequently change over time, this task has to be repeated over and over again, making automated methods even more desirable.

2. As the number of product configurations is high, it is impossible to use methods that answer the Question **Q1** to measure energy consumption for every single combination. An understandable method that can scale to large repositories is needed.

This thesis proposes an approach to deal with these two challenges in the context of the software repositories, to extract feature models automatically from the software repository and to create an understandable and scalable machine learning model design for providing insights into the configuration to choose.

Most large software repositories have metadata, which may consist of the name, version, dependencies and size, to provide information about the existing resources and how they can be useful for a deployment. These metadata are beneficial in order to obtain the dependencies between components. In a feature model, the dependency between features describes how a feature can be configured to a particular combination. By using the information contained in metadata, a feature model can be built and reconfigured. Using a feature model [179, 49], the categories of software components and the component dependencies from the information of metadata can be modelled and visualised. Furthermore, this will enable the stakeholder to choose which software components should be included in the final product by analysing the feature model.

We propose a method to build a feature model for SPL development automatically, i.e., without human intervention, from a software repository. In previous work, the process of building the feature model required human intervention, which is particularly expensive when dealing with a large repository. Our method also facilitates analysing the consistency between feature dependencies in the feature model and component dependencies in the original repository, making the checking process straightforward. This is particularly important because the package repository evolves with time. Therefore, after creating a feature model from a repository, we need to check whether the feature model is still

consistent with new versions of the repository, i.e., if it is still valid. Our approach can also adjust the size of the feature model generated from the repository, in order to manage the complexity of a feature model, by adding some constraints, such as excluding the conflicts and common features – this includes the library components that are used in most software configurations, to the retrieval procedure as a script of constraint. This is useful because it allows the management of the feature model to include only relevant packages.

In the large package repository, the number of possible combinations of packages is enormous. Measuring the energy consumption of all possible combinations is inefficient and impractical. Machine learning models to predict the CPU power consumption of configuration can be created based on a reduced number of energy measurements. The thesis proposes a new machine learning model design to predict CPU power consumption when there is a large number of product configurations. This model overcomes the issue of existing approaches, being accurate and interpretable. Users can thus use this model to decide what combinations of packages to use in order to reduce energy consumption. This technique answer to the Question **Q2** to measure energy from a large number of components and select the combination that consume less energy.

We create a comparative study between different machine learning approaches, such as Linear Regression, Regression Tree, Random Forest, Bagging + Regression Tree, Bagging + Linear Regression, and our new model – a new Regression Tree design tailored for energy prediction, Energy Prediction Trees (EPTs) – to find out which one performs better to predict the consumption of energy in a virtualised environment, including a large number of combinations and components. The comparison is based on the case study of two feature models that are retrieved from the Ubuntu package repository.

The energy consumed in the Cloud system is directly affected by the amount of resources used to cope with a workload. A challenging characteristic of Cloud systems is that their workload is highly dynamic [180, 4] as the number of concurrent users changes rapidly over time. One solution to handle a dynamic workload is to rely on elastic com-

puting, increasing or reducing the available resources to meet a Service Level Agreement (SLA). Scaling up a service leads to providing and using more resources, which also increases the cost of operation. We believe that it is possible to leverage techniques for reconfiguring the system in order to cope with different workloads without directly scaling up and yet saving energy.

In order to answer **Q3**, to build a dynamic system, the thesis proposes an approach to building a self-adaptive system in a virtualised environment. The architecture of this system reconfigures the combination of software components, automatically, based on the change of environment. In this work, the architecture has a sensor module that constantly monitors energy consumption and workload, which then feeds the information from the sensor to a Rule-Based System (RBS) [11] that helps to reconfigure the system dynamically based on the workload and energy consumed. In this approach, the thesis combines the SPL, models@runtime [24], and Rule-based system [11] to provide a self-adaptive energy aware system. The SPL, as core architecture, gains its resources from a software repository, such as Ubuntu package repository, that can be retrieved and reconfigured into a system based on the change of environment. A consolidation of the models@runtime and SPL provides a mechanism to change the configurations that is triggered via a decision made by the RBS, where a real-time monitoring of workload and energy usage of the servers are the input for the RBS. To evaluate this approach, a case study is created that build a self-adaptive web system that uses DROOLS [11] as the rule-engine, and Ansible [101] as a manager to configure and deploy the combination of packages that are retrieved from the Ubuntu package repository, in real-time.

1.2 Overview

Figure 1.1 shows an outline of our approach to managing the energy usage of a Software Product Line (SPL) and a Dynamic Software Product Line (DSPL) model within a Cloud and a virtualised system. The first approach measures energy usage from the virtualised

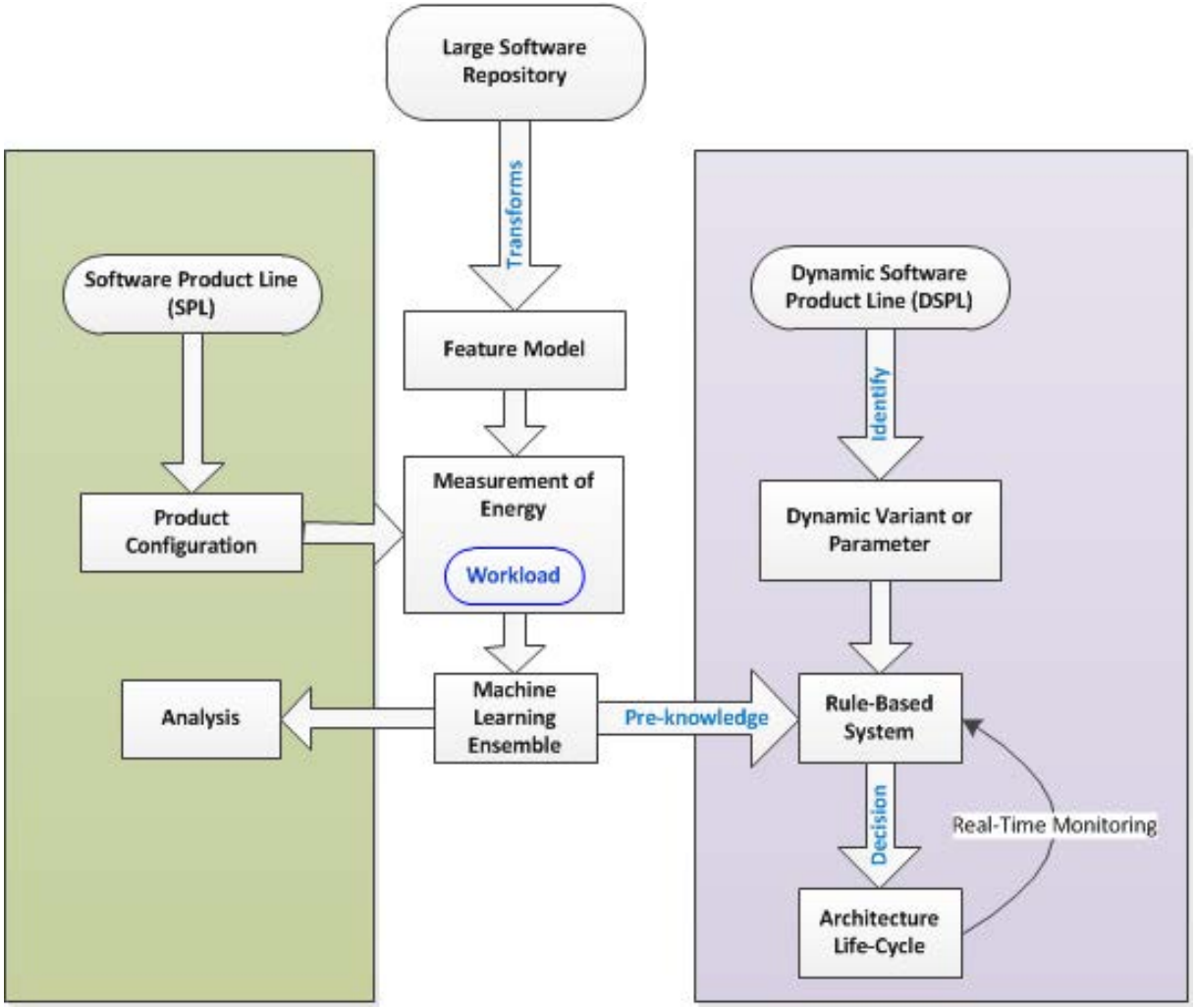


Figure 1.1: Overview of our approach.

environment with a workload to execute tasks within a system. In order to generate suitable workloads for each domain of application, a network traffic model is developed using a scenario to simulate users accessing the system. In the second approach, we create a data set for the Machine Learning process that combines the product configurations, as Boolean, and workload. These data sets use the CPU power values as the prediction targets. The Machine Learning ensembles present the prediction of energy usage within the product configurations of a Software Product Line. The third approach creates a Dynamic Software Product Line with energy that adopts the models@runtime [24] to adapt to the environment change, such as workload and energy. This approach reconfigures the software architecture, based on the change of the adaptive parameters that perform

resilient to the variation of energy usage corresponding to the workload before scaling up their infrastructure. In order to develop a Dynamic Software Product Line (DSPL) with energy, a self-adaptive system with a rule-based system within the virtualised environment is developed. The dynamic system uses real-time monitoring of energy as feedback to make a decision whether or not the architecture should be reconfigured.

It is worth noting that the feature model of product configurations for Software Product Line (SPL) and Dynamic Software Product Line (DSPL) with energy, in this thesis, are built using an approach that retrieves package dependencies from a software repository.

The detail of the contributions is defined in the following section.

1.3 Contribution

The contribution of this thesis is the following:

- A technique to measure energy usage for an individual process under a given workload in a Cloud and virtualised system was developed (Chapter 3). In particular:

    1. The thesis introduces a technique to generate workload using a scenario to execute tasks, i.e., end-users accessing an e-Learning system and doing several activities, in the virtualised system.

    2. The thesis introduces a measurement technique in the virtualised environment in a Laptop, where the software application for measuring energy monitors the electric current drainage from the battery.

    3. The thesis presents a technique to measure energy usage in the Cloud environment where the implementation of Advanced Control and Power Interface (ACPI) enables us to monitor the power usage in a Cloud system.

- A method to find the variability of and create a feature model from a large software repository, automatically (Chapter 4). In particular:

1. A technique using an Open-Source application to retrieve and merge packages from the large software repository is presented.
2. The mapping from the text graph file into a constraint of a feature model is presented.
3. The transformation model from the text graph structure into a constraint of a feature model is defined.
4. The mapping of the package dependency and structure from the software repository into a cross-tree and hierarchy structure of a feature model of Software Product Line (SPL) is defined.

- The machine learning model to predict energy consumption of product configurations of an SPL.

1. We introduce a technique to prepare a data set for the Machine Learning process from the product configuration of a Software Product Line, the workload of the measurement of energy, and the CPU power usages. (Chapter 3, 5 and 6)
2. The configurations to compose the data set can be selected randomly, or partially randomly, by ensuring that each package is included in at least one configuration, and popular configurations are included, and are presented. (Chapter 5)
3. The thesis introduces a predictive model to estimate energy usage using Machine Learning that predicts a numeric target variable (CPU Power) given a set of input variables (Workload and Product Configuration). (Chapter 5)
4. We present a technique using Regression Trees (RTs) and the Bagging+Regression Trees ensemble to predict energy usage of product configurations of a product line. (Chapter 5)

- A method to enhance the prediction of energy usage in a data set from product configurations of an SPL model using the machine learning ensemble. In particular:

    1. The thesis introduces the Energy Prediction Trees (EPTs) by restricting the split of variables representing the product configurations of an SPL. The EPTs provide an easy interpretation of the prediction results using a combination of the regression tree and linear model. (Section 5)
    2. The thesis discusses the model-based recursive partitioning procedure to provide a linear regression model of EPT's leaves to predict a single CPU power value. (Section 5)

- A Dynamic Software Product Line (DSPL) with energy. In particular:

    1. The thesis presents an approach for the autonomous software architecture configuration by using the variant of the feature model where the monitoring of real-time energy usage and workloads become the input. (Chapter 6)
    2. A mechanism to manage and reconfigure a dynamic system using broker-agent software is developed. (Chapter 6)
    3. A self-adaptive load balancer, with the Rule-Based System in a virtualised environment for the DSPL with energy, is built. (Chapter 6)

1.4 Structure of This Thesis

This thesis is structured as follows:

- Chapter 2 introduces the essential background material of this thesis to the reader. These are the Cloud computing, the Software Product Line Engineering, the Feature Model, and the Dynamic Software Product Line, the power model, energy measurement, machine learning and the autonomous system for the virtualised environment. In addition, reviews on related work for the static and dynamic Software Product Line with energy are discussed.

- In Chapter 3, we present the energy measurement technique in the Cloud and virtualised system. This chapter, then, discusses a method to generate workload with a scenario to execute tasks within the system under measurement that uses a traffic model to simulate users accessing the system. We conduct an experiment to measure energy usages for virtualised environments and present the results.

- Chapter 4 presents a technique to create a Feature Model from a Large Software Repository. This technique describes how to retrieve packages and their dependencies from a software repository. The retrieval results are then transformed into a Feature Model.

- Chapter 5 discusses a method to model a Software Product Line with Energy. The machine learning ensemble is used to approximate and predict the consumption of energy from the combination of features. In addition, the comparative consumption of energy of features is also discussed in this chapter. We conduct an experiment using feature models that are retrieved from a large software repository. In this chapter, we compare the results of our prediction technique to several other machine learning algorithms.

- Chapter 6 presents the management of energy by reconfiguring, at runtime, the software architecture based on the change of environment (i.e. workloads and energy usages) using the Dynamic Software Product Line (DSPL). An experiment is conducted on a self-adaptive Load-Balancer with a Rule-Based System (RBS) to demonstrate our approach. We also compare our DSPL approach to a simplified elastic virtualised environment approach.

- Chapter 7 discusses the conclusion and the future works of our research.

1.5 Publications Resulting from this Thesis

Parts of the research in this thesis have been published in several papers:

1. I Made Murwantara and Behzad Bordbar,

   A Simplified Method of Measurement of Energy Consumption in Cloud and Virtualised Environment, *In 4th IEEE Big Data and Cloud Computing Conference*, 2014.

2. I Made Murwantara, Behzad Bordbar and Leandro L. Minku,

   Measuring Energy Consumption for Web Service Product Configuration, *In ACM 16th Integrated Information and Web Application Services (iiWAS) Conference*, 2014.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, firstly we present an introduction to the Cloud computing system. Next, we present available techniques to measure energy in virtualised environments. We then introduce Software Product Line (SPL) because the approaches that we present in this thesis are derived from SPL. Afterwards, we introduce and discuss Dynamic Software Product Line (DSPL). Then, we introduce a large software repository. Finally, we introduce the Machine Learning approach.

2.1 Cloud Computing

This section provides an essential background of Cloud computing systems. It begins with essential characteristics of the Cloud system, then service models, followed by deployment models.

There are two well-known definitions of Cloud computing that are widely cited:

(1) The US National Institute of Standard and Technology (NIST) defines Cloud computing [147] as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction".

(2) According to Rajkumar Buyya [33] "Cloud [computing] is a type of parallel and distributed system consisting of a collection of inter-connected and virtualised computers

that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers".

In practical, as depicted in figure 2.1, Mell and Grance [147] stated that a cloud model is composed of five essential characteristics, three service models, and four deployment models.

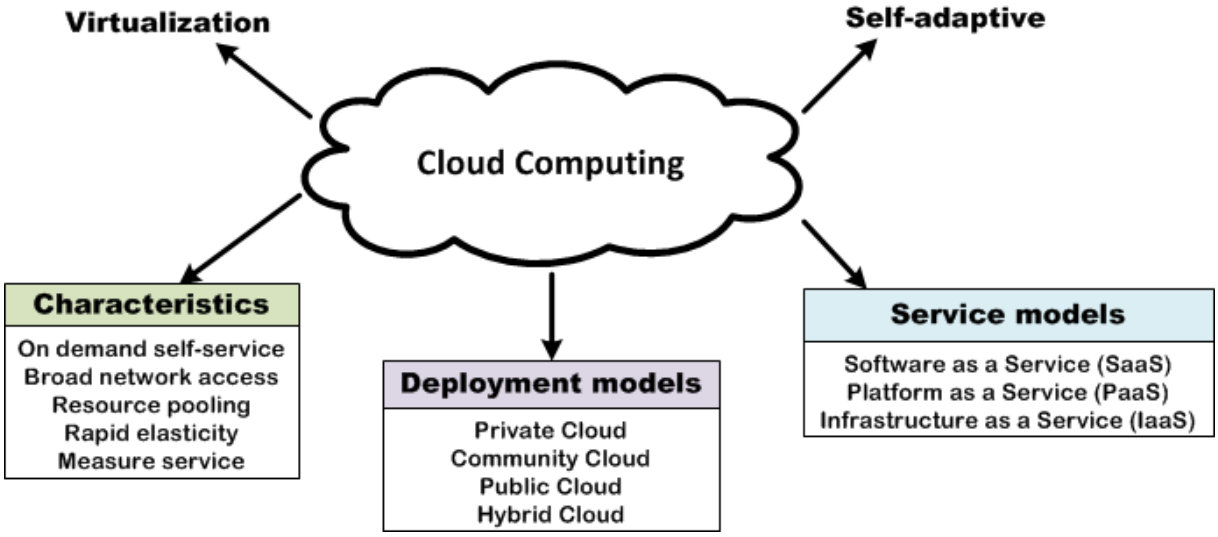


Figure 2.1: Cloud computing, adopted from [33, 147] .

2.1.1 Essential Characteristics of Cloud Computing

Cloud technology has options that are enticing various businesses around the globe. Therefore, it is important to know the essential characteristics of the Cloud [147].

1. **On-demand self-service** enabling users to provision resources, such as network storage and server time, without requiring human interaction for each cloud provider service.

2. **Broad network access** provides an accessible network through standard mechanism, such as heterogeneous thin and thick client platforms, which enable devices such as mobile phones, tablets, laptops and workstations to have access to the virtual system.

3. **Resource pooling** enables consumers to enter and use the data within a Cloud at the same time, from any different location and at any time. This capability leverages the efficiency for businesses that have offices around the world.

4. **Rapid elasticity** provides scalable resources with affordable prices to any business needs, which enables consumers to add and remove resources easily and quickly.

5. **Measure service** provides transparency for both the consumer and provider of the utilised service. The Cloud systems have a metering capability at some level of abstraction, such as storage, processing and bandwidth, where this typical services exist as pay-per-use basis.

2.1.2 Service Models of Cloud Computing

The Cloud computing service model is divided into layers, as shown in Figure 2.2, where *Platform* and *Ecosystem* views promote a new way of computing [47].

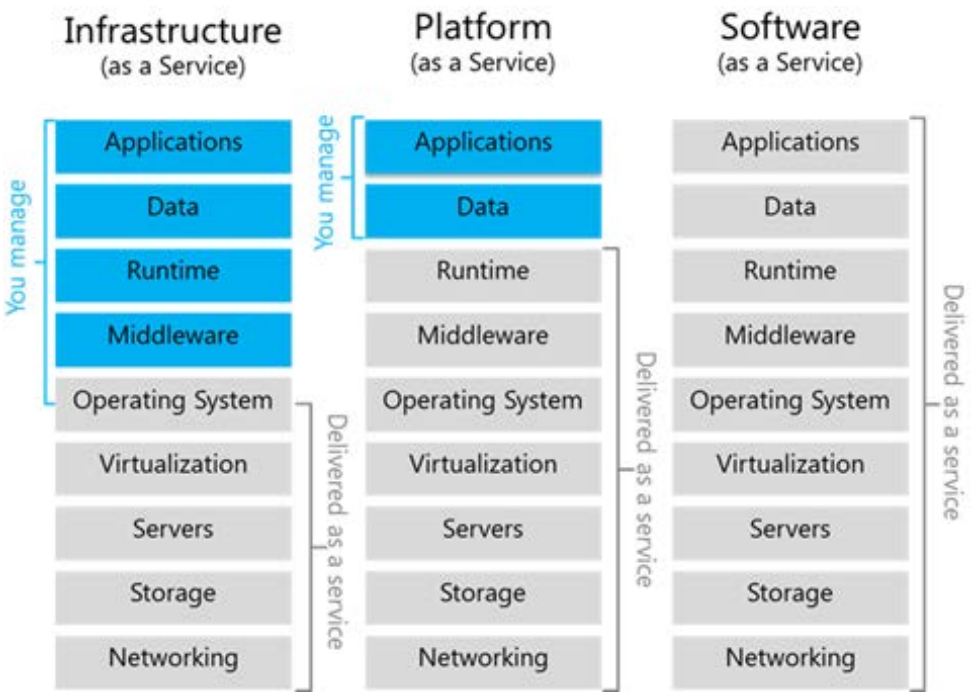


Figure 2.2: Cloud computing service models, adopted from [47, 147].

1. **Software as a Service (SaaS)** allows consumers to use a provider's application running in a Cloud infrastructure. SaaS enables various clients to access applications through a thin client interface, e.g. internet browsers. Nevertheless, the consumer cannot manage the cloud infrastructure such as networking, operating system and storage, only a limited application configuration setting may be allowed to the end-users. For example, Google Apps, Salesforce.com are SaaS providers.

2. **Platform as a Service (PaaS)** enables capabilities that are provided for the consumers to manage the software stacks, which include integration of web services and databases, and build a collaborative platform for software development. For instance, Force.com and Google App Engine are PaaS providers.

3. **Infrastructure as a Service (IaaS)** is a way of delivering infrastructure on-demand services, such as storage, CPU, network and operating systems, in the Cloud computing environment. This service model allows for dynamic scaling of Cloud infrastructure. For instance, Amazon Web service (EC2,S3,others), Rackspace, Windows Azure are IaaS providers.

2.1.3 Deployment Models of Cloud Computing

Cloud deployment models represent the environment category, prominently distinguished by their ownership, distribution, physical location and the purpose of their services. It is categorized as follows:

1. **Private cloud** is implemented for the exclusive use of the particular organizations. This deployment model only permits the authorized users, so that the organization can have direct control over their data and applications. Such models are largely deployed using Openstack and Opennebula.

2. **Community cloud** is deployed for the exclusive use of a community, e.g. a research group, where a different organisation may be involved in using this cloud model.

A community cloud can be internally managed or use a third party, where their intention is to achieve their related business objectives.

3. **Public cloud** is a deployment model that is delivered over a network that is open to the public. This model is a representation of commercial Cloud providers, where the consumers do not have control over the location of the infrastructure. For instance, Amazon EC2 and Windows Azure are public clouds.

4. **Hybrid cloud** is a type of Cloud deployment model that is integrated, which is the combination of two or more deployment models. Hybrid cloud has the benefit of overcoming the limitation in the public, community and private cloud. Such a hybrid cloud manages and provides their resources either in-house or by external providers. For example, an organization can use the cloud for archiving old data, and keep operational data exclusively.

2.2 Energy Usage in a Cloud and Virtualised Environment

Considering the ongoing increase in the cost of energy, efficient energy management of applications within a Cloud system has attracted many researchers [197, 12, 61, 92, 111, 17, 16]. This section describes existing techniques to measure energy usage in the Cloud and virtualised environment. In particular, this section explains a technique to measure energy using a software power meter and workload.

2.2.1 Power and Energy

It is essential to know the terminology in order to understand the energy measurement. Energy is the capacity or power to do work [90] that exists in a variety of forms, such as electrical and mechanical. The electrical energy is the amount of power over a period of time. It is worth noting that the International System Unit - Système international

d'unités (SI) - of power is the watt (W), and energy is the joule (J), where 1 watt second is equal to 1 joule [90]. To describe this formally, power (**P**) and energy (**E**) can be defined as in 2.1 and 2.2.

$$P = I \cdot V \tag{2.1}$$

$$E = P \cdot T \tag{2.2}$$

where **P** is the power, **I** is the electric current, **V** is the electric potential difference, **T** is a period of time and **E** is the energy. Energy differs from Power in terms of the period of time or time length of an execution of tasks running in a system.

2.2.1.1 Power Consumption in a Computer System

In a UNIX-like operating system, a system call [203, 137] (sometimes referred to as a Kernel call) is a request made via a software interrupt by an active process for a service performed by the Kernel – a program that constitutes the central core of a computer operating system, and it has complete control over all resources on the system and their activity. Further, an active process utilises the CPU by sending tasks through the Kernel to run services, which include process control, communication, device management, file management, and information maintenance.

According to Intel Labs [153], in a server architecture, the CPU consumes the highest amount of energy, followed by the memory and the power supply efficiency loss, as depicted in Figure 2.3. Current CPU technology uses a multi-core architecture that is more efficient than a conventional older technology that implements a single core technology. In addition, the CPU power efficiency and power saving techniques have improved, enabling low-power active modes.

Most current processor architectures and operating systems have adopted Advanced Configuration and Power Interface (ACPI) [152] for power management. The ACPI of-

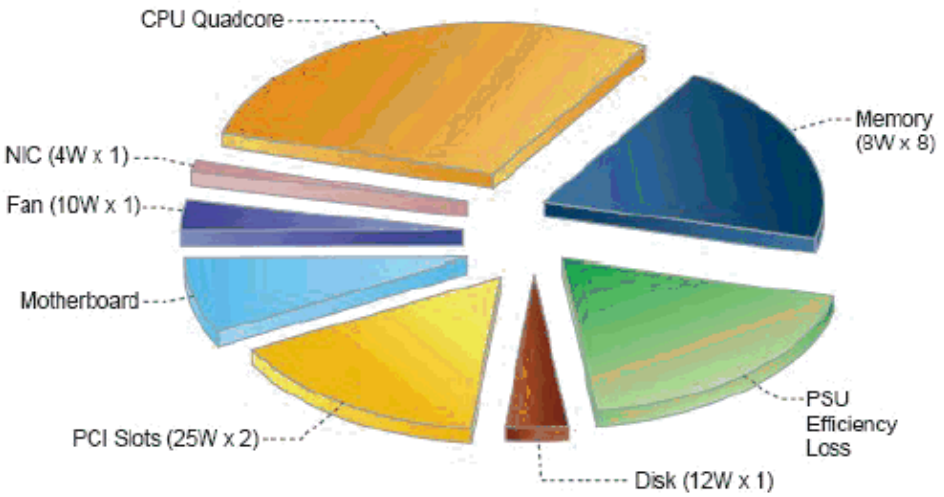


Figure 2.3: Intel server energy usage [153].

fers the structure and mechanism to design an operating system that directed power management, and also provides control and information needed to perform device power management such as turning-off or -on a virtual machine.

The CPU power states (C-states) [152] are a CPU idle state to save power by deactivating CPUs that are not in use. The C-states are defined in a sequence of number to represent the power saving. In C-States, the greater the power saving, the higher the number, as follows:

C0 – CPU is working in the normal operating state of a core to execute code, not idle.

C1, Halt – CPU is not executing any instructions, not in a lower power state and will continue the process with no delay. The core halts.

C2, Stop Clock – CPU clock is stopped. In this state, CPU maintains all software visible by the system, but may take longer to wake up.

C3, Sleep – CPU is in a deep sleep state, and to wake it up takes a longer time than C2, where the CPU does not keep the cache of the system.

It is worth mentioning that ACPI defines the power state of CPU as being either active (executing) or sleeping (not executing) [152].

In the old Linux Kernel operating systems, they periodically interrupted the CPU on a system call to pre-determine their working frequency. Such action gathers the information

of tasks that are being executed to manage the processes and load balancing, which is known as the timer tick, regardless of the power state of the CPU. As a result, a CPU always responds to this request, preventing the CPU to stay idle long enough. To address this, in the Linux kernel, since Version 2.6.21 [116, 98], it runs *tickless* to allow idle CPUs to remain idle, and to stay in the lower power states longer. When a new task is queued for processing, then the CPU is awakened to execute tasks. This power saving mechanism uses ACPI that are interfaced with C-states.

2.2.2 Measurement of Energy

The measurement of energy consumption in Information Technology is an active area of research [228, 26, 25, 123, 77, 214, 222, 167]. Kansal et al. [123] proposed a tool to monitor the resource usage of runtime software components. Jason et al. [77] and Vergara et al. [214] use an advanced combination of a software tool and hardware power meter, which involves specific hardware to measure energy usage for more specific purposes, such as Analog to Digital Converter (ADC) that is combined with Wi-Fi or communication systems. An example of a hardware device to measure the energy consumption of a computer system is WattsUp [220]. These devices are commonly known as Wattmeters. Chen et al. [45] measure a cluster of computer systems' energy consumption using a Wattmeter. In their measurements, they simulate the user via a workload tool.

Most existing techniques involve the use of a Wattmeter [228, 25, 45] to measure ***total*** energy usage of the computer system. As a result, we need a different method of measuring energy consumption by individual processes. For this reason we can group a number of processes, such as HTTP server and Database system, and measure their energy consumption. However, this is non-trivial as we cannot use Wattmeter products to measure individual processes within a computer system; they do not distinguish between processes. We need to identify the energy that is being used when a process is running. In a virtualised environment, the Wattmeter devices cannot measure the energy consumption of a virtual machine (VM) because the resources are shared and VM can move from one

host to another. As a result, we need to consider using a software power meter that is not dependent on Wattmeter devices, i.e. Powertop, as explained in the next section.

2.2.3 Measurement of Energy Consumption via Software and Workload

There are a number of software applications for measuring power consumption in a computer system [150, 200, 77, 114]. Some of these software products use battery drainage to measure how much energy is consumed. Among these tools are Joulemeter [150] and Powertop [200]. Joulemeter [150], developed by Microsoft Research, measures the specific running services energy consumption in real time. It estimates power usage of a computer system by reading the log file of CPU utilization, the Monitor brightness and the Disk usage. The Joulemeter measures energy usage of a typical laptop on battery mode and other infrastructure, such as servers, using a Wattmeter device.

Powertop [200] is an Intel open-source tool designed to measure, inform and reduce the electrical power consumption of a computer system. This tool analyses the device drivers, kernel and programs running on a Linux based system and estimates – using a machine learning [157] approach – the power consumption resulting from their use. In predicting a measurement, first, a data set is created and projected to build a reference model as a calibration activity. Then, based on the reference model, Powertop estimates the power consumption of a computer system. It is worth noting that the sources of data to calibrate and measure power come from a virtual file system – sysfs – provided by Linux. Sysfs [106] provides a set of virtual files by exporting information about various kernel subsystem, hardware devices and associated device drivers from the kernel's device model to user space. Such a file system allows Powertop to capture information that relates to power usage and system activities, such as battery condition and CPU status.

2.2.3.1 Workload for Computer System

The workload model for a computer system, such as web workloads [176], is an abstraction of the real workload that reproduces users' behaviours and ensures that a particular task within the target of the workload performs as it would do when working with real users. Further, workload appears in many contexts to aim at different interests [76]. According to Feitelson [75], in order to schedule a CPU, each job arrival and running time should correspond to the relevant attributes. If memory usage is the interest then the total memory usage and locality of tasks that utilise the memory should be considered. In addition, a repeatable method to provide similar workloads, i.e. network traffic, should be consistent with the system configuration being tested to obtain persistent results [76]. A traffic model is a combination of network traffic with some amount of loads to provide

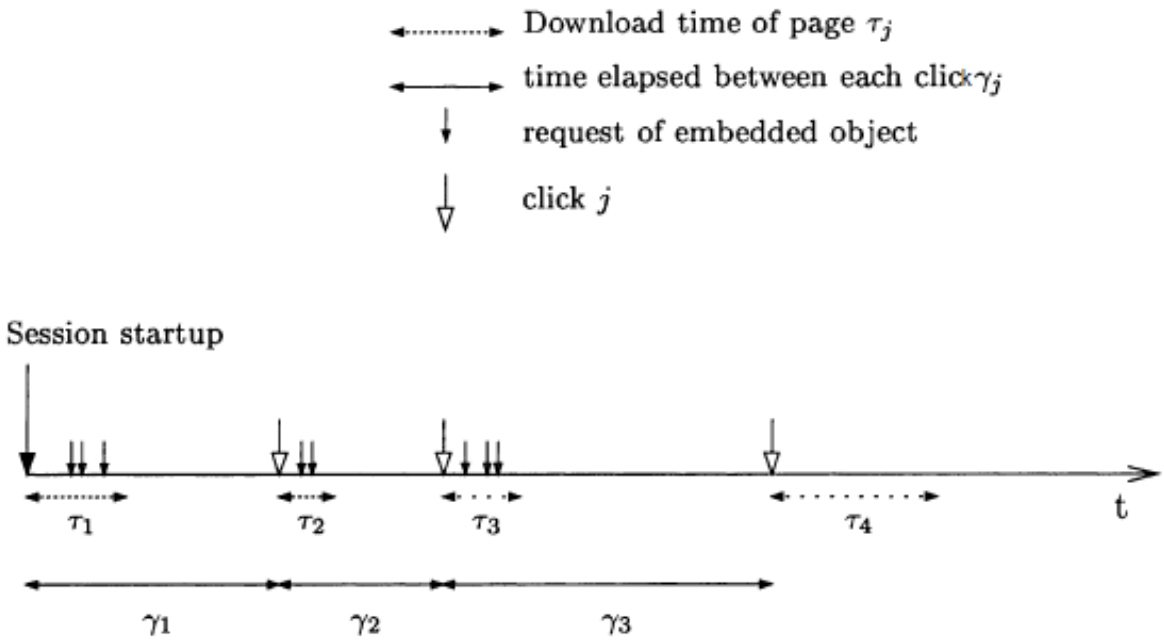


Figure 2.4: An example of the user session in Tsung, Liu et al. [139].

a suitable workload to a system [139]. A user session, as depicted in Figure 2.4, is used by Liu et al. [139] to distribute a load in a traffic model.

In order to measure the consumption of energy of any given application, a suitable workload must be provided in order to simulate its usage. Such a workload should be able

to simulate the end-users' request traffic to a system. Liu et al. [139] built an approach for the application of web traffic to evaluate the Web servers performance, which was adopted as a workload tool [209]. Ortiz et al. [176] are concerned with dynamic load in order to evaluate the current and the incoming workload of a web system. They use a traffic model with a session level to simulate users arriving at the server and how end-users use an internet browser, such as Mozilla and Chrome, to browse the HTTP server. It is worth noting that a session is a sequence of an end-user clicks the hyperlink in their browser pointing to the same server. Figure 2.4 shows a user session using a Web browser accessing an HTTP server to execute tasks involved in each scenario, where τ represents the time taken to retrieve Web pages, and γ represents the inter-arrival times of clicks that describe the time durations elapsed between the clicks on the hyperlink and the completion of loading the requested page.

2.2.3.2 Workload Tool

There are a number of workload tools [209, 6, 123]. Among others are Tsung [209] and JMeter [6]. Tsung is a distributed load testing tool that implements a stochastic model for real user simulation, and is built in the Erlang [39] programming language, where the user event distribution is based on a Poisson process [139], and uses a traffic model [165] to simulate real-world user behaviour. JMeter [6], a java based application, is designed to generate a load to test functional behaviour and measure performance. Both Tsung and JMeter are capable of distributing workload for static and dynamic resources, such as Web systems and Database management systems.

A workload scenario [187] represents the way an application executes tasks in real-life systems. Such a workload scenario provides a kind of mechanism to send a load to the object of measurement in a consecutive way. For example, steps to commit a transaction in an online shopping system. It begins with a user login into the system, then selecting one or more items. After that, it makes a payment with a banking agent system attached to the shopping system. Both Tsung and Jmeter support the use of workload scenarios

within their script. Despite this, there are some limitations, among others that neither of the workload tools can handle JavaScript request.

Discussion

In measuring energy usage for virtualised environments such as a Cloud computing system, researchers have provided approaches by using Wattmeter devices and a software power meter. Kansal et al. [123] and Bohra et al. [25] have developed techniques using a combination of Wattmeter devices and a software power meter. Their approach includes a prediction method that obtains its sources by measuring the CPU, memory and disks energy usage of physical servers. The Kansal [123] method only measures one process at a time, where the total consumption of energy associated with the software components cannot be included. As a result, total energy caused by executing tasks is not measured. Bohra [25] developed a power model by adding the consumption of energy in CPU, memory and disk, and also includes the interdependence between them. However, it is hard to isolate processes within virtualised environments, and also a unique combination of software components within a system may execute different tasks. In this work, we propose a technique to measure energy usage for individual tasks within a virtualised environment, where a workload is used to execute combinations of components within the system under measurement.

There exist approaches that use models to migrate the underutilised or idle virtual machine in a Cloud system [16, 34, 17, 13, 4]. Beloglazov et al. [16, 34, 17] developed algorithms to detect a host with an overload problem using the Markov Chain model, and also implemented a distributed dynamic approach for the virtual machine consolidation. This approach migrates the virtual machine instances to improve the utilisation of resources and to reduce energy. Basmaidjan et al. [13] and Alansari et al. [4] use policies to build the energy management in the Cloud system. They optimise the energy usage by managing the business rules that are associated with the consumption of energy of a virtual machine and by continuously monitoring the Cloud environment. Even though

promising, these approaches still require some adjustment in order to reduce energy usage. For example, although Alansari et al. enforce the energy policy into a business rule system through interaction with the Cloud manager to migrate under-utilised virtual machine, the workload to the Cloud system dynamically changes over the time line. Moreover, when the system under management is a dynamic system, it is inefficient to predict the consumption of energy based on a growing number of data, such as Basmaidjan et al. have proposed.

Smith et al. [199] and Yu et al. [224] introduced methods to measure energy in a virtualised system by using a workload to execute tasks to utilise CPU, memory, hard drive and network. These methods monitor the energy usage of a private Cloud through agents that are installed in the servers to find out the behaviour of the system when a workload arrives at the system. Smith et al. simulate processes such as file compression to perform tasks in the system under measurement in order to measure the consumption of energy. Yu et al. propose to use the relationship between the measurement result and workload to manage the Service Level Agreement (SLA) for Cloud provider services. Even though the workload that is used in both the Smith and Yu techniques do not aim to execute individual tasks of combination of software components in the system, these approaches are worthy of mention because their methods continuously monitor the consumption of energy without interrupting the running services.

2.3 Software Product Line and Dynamic Software Product Line

This section provides a brief introduction to software component reuse and its extension to a dynamic system. Two approaches are reviewed: Software Product Line engineering and Dynamic Software Product Line engineering. This section aims to provide the literature's perspectives and to show potential uses of these approaches to pursue the goals of this thesis.

2.3.1 Software Product Line Engineering

According to Clements and Northrop [49] "A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way". A Software Product Line (SPL) engineering builds new competitive and distinctive products from assets or artifacts of software products' members that are included into a product line, and it aims to configure new high quality products that reduce the time and cost of software development. In recent years, SPL has been succesful in many different domains, such as avionics [190], electronic consumer products [168], firmware [208] and the health systems [221].

Software Product Line (SPL) engineering focuses on the reuse of assets of products with a mass customisation of multiple or more similar products. Further, a family of software products [174] could be treated as related sets, in modules or component forms which can be composed or decomposed [173] to achieve better results. Therefore, reuse of assets in a family of products [122] in the same domain should be packaged, managed and reused as software modules for each variant product in the domain to develop a domain architecture and reuse the software components.

In pursuing its goals, software product line engineering is divided into two processes, namely Domain engineering and Application engineering, as depicted in Figure 2.5. The task of Domain engineering is responsible for identifying or creating reusable assets and Application engineering is to reuse assets to build new products in a similar domain. It is worth noting that both Domain and Application engineering are interrelated without any specific order.

2.3.1.1 Domain Engineering

Pohl et al. [179] state that "domain engineering is the process of software product line engineering in which the commonality and the variability of the product line are defined and

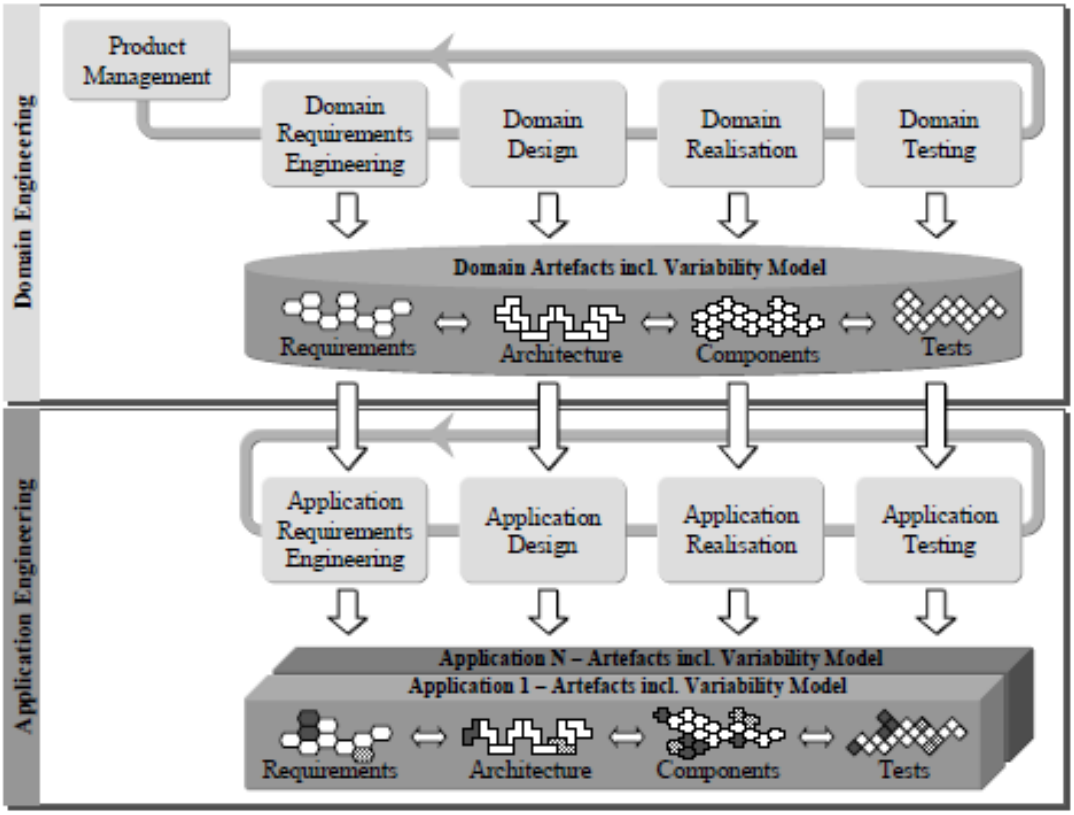


Figure 2.5: Software product line engineering framework [179].

realised". Further, domain engineering identifies and categories the artefacts of members' products of a product line in order to plan a software product line development. Among other aims is to set a scope for development, and to construct reusable artefacts. In domain analysis, the commonality and variability of assets are accomplished to obtain the desired construction of the reference architecture. The reusable software components are designed and configured in the domain realisation phase, and then created into the variability of software product line.

2.3.1.2 Application Engineering

"Application engineering is the process of software product line engineering in which the applications of the product line are built by reusing domain artefacts and exploiting the product line variability" [179]. In the application for the requirement engineering, aims to find the commonality and variability between the application and the domain

requirement to build a product configuration. After that, in the application realisation phase, a software product is built by assembling the reusable and application-specific assets that are derived from the reference architecture.

2.3.1.3 Variability Management

According to Kang et al. [121], variability is the notion of relationship, data-flow across domain, binding times, application domain, environmental differences and assumptions. Variability management defines and represents the commonality and variability artefacts and is the key part of software product line development [44]. It is one of the most important activities that distinguishes software product line engineering from traditional software reuse and other software development approaches [27].

Variability management is the key factor in software product line development. The Variability model is dedicated to manage variability in a software product line. In this report, we use Feature Modelling to create our variability model.

2.3.2 Feature Model

Feature modelling is the most common technique to model the variability of a Software Product Line development. It represents commonality and variability as a set of features and their dependencies. Kang et al. [121] introduced Feature Oriented Domain Analysis (FODA) to perform the domain analysis. They defined a feature as "a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems". Thus, the feature model is considered the prominent standard to describe the variability in a software product line.

2.3.2.1 Notation

According to Czarnecki and Eisenecker [54], a feature model should represent the intention of a concept throughout a set of instances, which are the assets of a product line. A feature diagram, which is represented in a hierarchical structure, derives the feature's description

as a concept, as depicted in Figure 2.6.

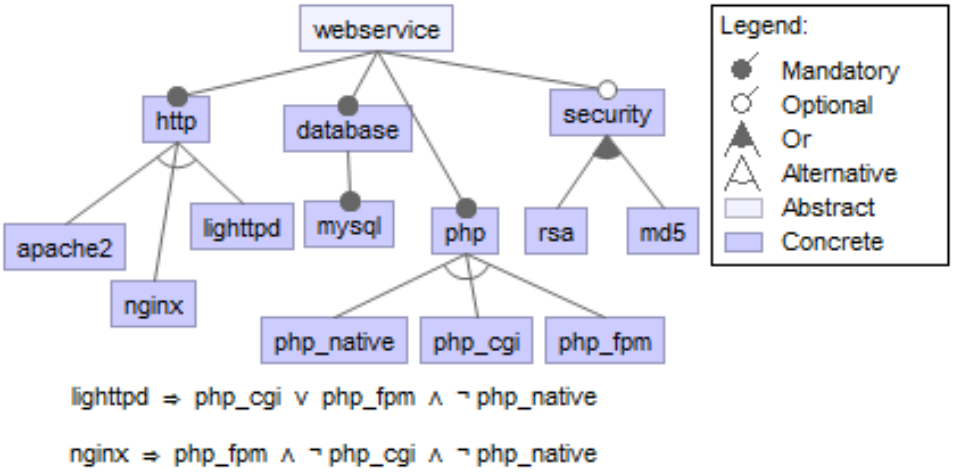


Figure 2.6: A feature model of web services [built using FeatureIDE [206]].

A feature model is organised in a tree like structure. Features are represented as a rectanglular box with a title inside to indicate specific goals. The root feature describes the main feature under development, e.g. web service. A feature's edge is used to show the parent-child relationship among features. There are two ways to describe variability; firstly, features can be decomposed into sub-features, where a sub-feature may be optional or mandatory. For example, http and security are mandatory and optional features, respectively, that are illustrated with a filled black circle and unfilled circle. Secondly, expression of variability using "or" and "alternative" relationship, where "or" means at least one or more sub-features can be selected, and "alternative" means only one sub-feature is allowed if the parent feature is selected. Further, in a feature model, a cross-tree relationship is presented as constraints, as shown under the tree-relationship in Figure 2.6. As an example, when "Nginx" is selected then "php-cgi" and "php-native" cannot be included in a product configuration.

2.3.3 Dynamic Software Product Line Engineering

Dynamic Software Product Line (DSPL) engineering augmented the Software Product Line (SPL) engineering approach by moving its variability binding to runtime [104, 107,

41, 18, 24]. In the DSPL [107], variability management describes different adaptations of the system, where the adaptability scoping identifies the range of adaptation. Such an approach enables the system to reconfigure itself during operation, adaptive to the environment changes [107].

Monitoring the current situation and controlling the adaptation are a key tasks of the DSPL [104, 40]. Moreover, a DSPL should have at least the following [107]: the properties that support dynamic variability, variation points change during runtime, it deals with changes made by the user or unexpected changes, it contains an explicit representation of the configurations of components and be self-adaptive. It is worth noticing that the changes of environment should minimize or not disrupt [130] the current running system.

2.3.3.1 Dynamic Adaptation

Dynamic adaption, e.g. self-adaption, in a software system allows the software structure to improve performance and increase availability corresponding to the changes of environment [175, 36, 85]. Moreover, according to Alférez et al. [5] dynamic adaptations of a software system need to include context awareness, policies, infrastructure support, and verification. One of the well-known reference models for an adaptive model, developed by IBM, is MAPE [113], as depicted in Figure 2.7, which organises the internal structure into a set of capabilities or functions that includes **M**onitor, **A**nalyse, **P**lan and **E**xecute (MAPE). Furthermore, IBM's MAPE model has been defined for control loops runtime. However, adaptation not only covers the architecture of a system, it also pinpoints the change of software architecture at the runtime phase.

Dynamic adaption refers to the change of architecture or software architecture while a system is running or being executed, where components of a system or application such as software structures and configurations can be replaced or modified without restarting the whole running system [125]. This adaptation capability is usually based on feedback on a control loop that manages the characteristics of the system to meet their goals, where an architecture transformation may emerge during execution or operation.

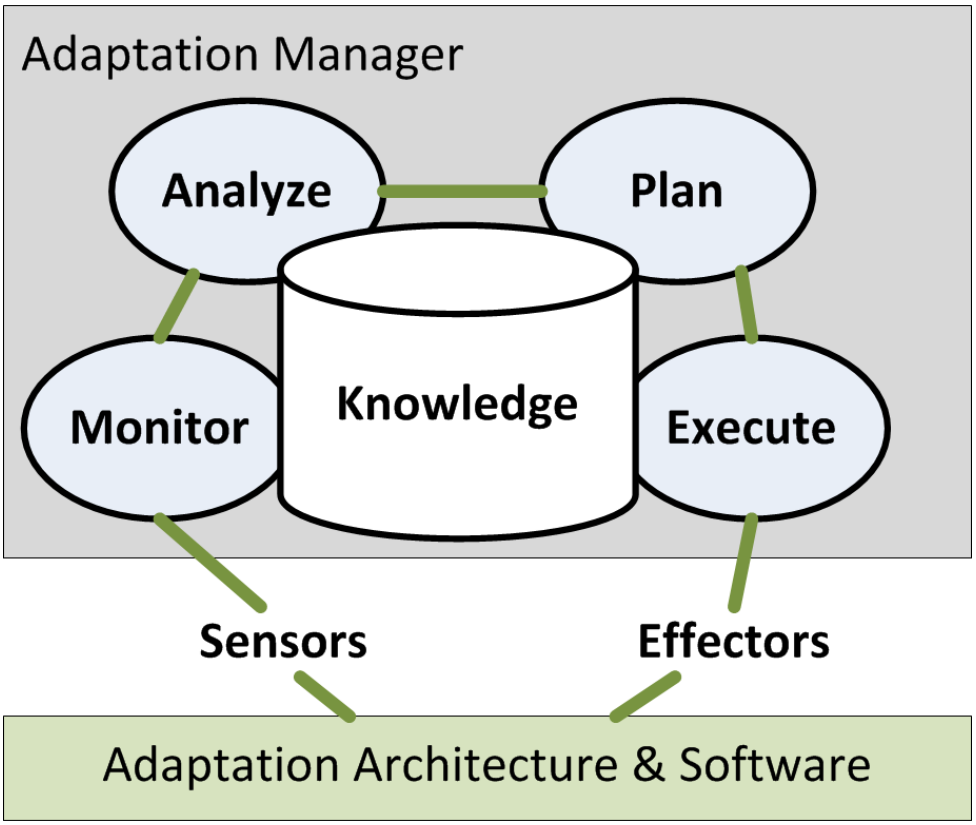


Figure 2.7: An autonomic system MAPE-K, adopted from [113, 215].

2.3.3.2 Self-adaptivity using models@runtime

The Models@runtime [24] technique is defined as "a casually connected self-representation of the associated system that emphasises the structure, behaviour, or goals of the system from a problem space perspective". This technique is built – for engineering self-adaptive software systems – to support a continuous adaption to the changes in the environments or requirements. A self-adaptive system, which adopts IBM's MAPE approach, splits the software into domain and adaptation logic [215, 183]. The domain logic implements the adaptable software, and the adaption logic performs as a feedback loop to represent the self-adaptation. As depicted in Figure 2.8, the transition diagram [96, 97, 18] shows how a system adapts to correspond to the feedback, using automaton as the state and edges as the transition rules. For example, State 1 changes to State 3 when the workload increases from 1 to 5000 requests per second within 10 minutes.

According to Aßmann et al. [8], the models@runtime model should be based on the

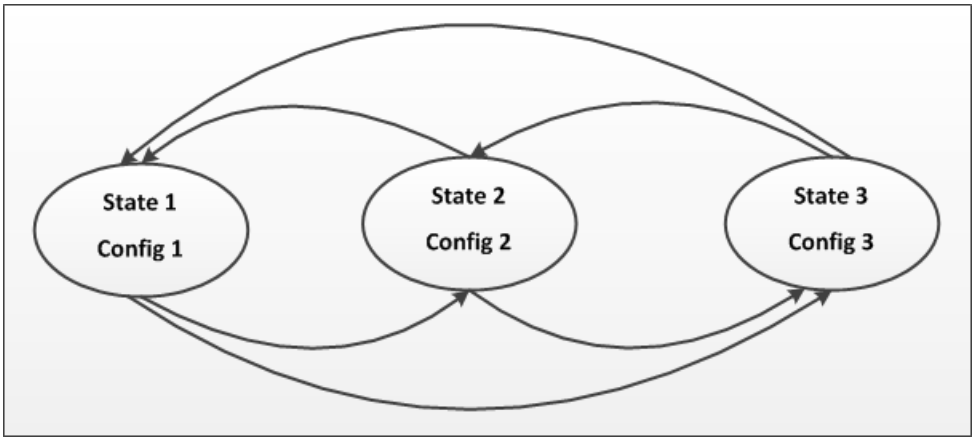


Figure 2.8: Transition diagram, adopted from [96, 97, 18].

reflection principles that includes introspection and intercession by avoiding complexity, danger and irreversibility of reality, where the perceived reflection, modelling and separation of concerns are the main requirements. Furthermore, models@runtime should comprise layers for managing models of the system, configuration management and a goal management for all these layers to work in a feedback loop. In order to make a system adaptation that is driven by models [19], models@runtime has a framework that is slightly similar to an SPL, except in the dynamic behaviour, that consists of two parts, the application and the runtime platform, which define the delivery of an adaptive system and the nature of infrastructure can make a system adaptable to the environment.

In models@runtime, the evolution of a software system [38], such as the Debian/Ubuntu package repository, is managed as a crosscutting concern - the concern that affects many other parts of the system - where the software components that evolve are not included at the design-time. This is because components that evolve are not known at design-time or do not yet exist, where the human intervention is needed to reflect these new components into the design of the system.

Discussion

Several approaches to reducing energy consumption through SPL exist [194, 196, 62, 160]. Siegmund et al. [194, 196] optimise product configuration by selecting features

according to user-defined non-functional requirements that include energy usage. Their approach is based on building a feature library containing information that can be used to compute energy consumption. However, their method is impractical when the number of possible features is large, because the number of combinations of feature for which energy consumption would need to be measured is very high. In addition, they do not specify how exactly to measure energy consumption based on the feature library.

Dubslaff et al. [62] introduced a feature composition framework using Markov Decision Processes (MDP) to reason on energy consumption based on the feature-dependent behaviour between actions of the system. However their approach only considers the energy consumption of features in isolation; It does not consider the energy consumption of configurations.

Martin et al. [160] reduce energy consumption of supplementary lighting for greenhouses. They used weather forecasts and electricity prices to compute cost and energy efficiency of supplementary light plans. The SPL approach is then used to tune the lighting so as to reduce energy consumption. Their method is only suitable for greenhouse development.

Dougherty et al. [61] created an auto-scaling technique to reduce energy in a virtualised System. Their approach uses pre-booted virtual machine instances to handle periods of high demand. Moreover, they adopted model-driven engineering to pre-configure the pre-booted virtual machines. This technique uses a constraint solver to optimise the virtual machine configurations and energy usages. However, an idle virtual machine still consumes a significant amount of energy; if there are many combination options to create the pre-configured instances, then number of the pre-booted virtual machines will be high. As a result, this condition leads to a high cost and energy expenditure.

Bencomo [18] and Gamez et al. [82] built the runtime model of a software product line for a middleware system. Both models emphasize that energy reduction can be managed using a self-adaptive technique. They used the dynamic variability model that relates to the change of environment as a constraint for their runtime model. Their

approach considers communication, such as Wi-Fi, as an energy exhaustive component. Both approaches built a transition mechanism to select which communication component consumes less energy corresponding to the change of environment.

Götz et al. [95] developed a contract negotiation approach to match user requirements that allowed them to identify the relationship between energy usage and software components. This approach uses a runtime model on a combination of hardware and software systems. Their model realizes energy effeciency by selecting which resources match the software configuration in order to serve the user's demands. Moreover, this model requires a runtime adjustment that takes its knowledge from the system's variability. Götz et al. claim that this approach allows them to optimise a dynamic system.

2.4 Large Software Repository as a Source for Software Product Line

This section provides an essential background about large software repositories and their usage as the source for the software product line development. An overview of current research that uses different repositories is also presented. This section also investigates the evolution of an Open-Source distribution.

2.4.1 Introduction to Large Software Repository

A large software repository is a collection of software components [63], artefacts and information that relate to deployment, configuration and sometimes bugs. The large software repository has been used as the resource artefact in Software Product Line development, such as Ubuntu/Debian package repository [216, 81, 58], Linux Kernel Kconfig [119, 191] and the eCos embedded system component repository [21]. These repositories evolve over time to achieve better performance and to remove bugs within a timeline.

Package dependency describes the relationship between packages to provide a convenient way for end-users to check the software deployment. Large software repositories

can have different methods to describe dependencies. For example, Debian/Ubuntu uses package description files to describe package dependencies, and eCos uses the Component Definition Language (CDL) [78] to describe component dependencies.

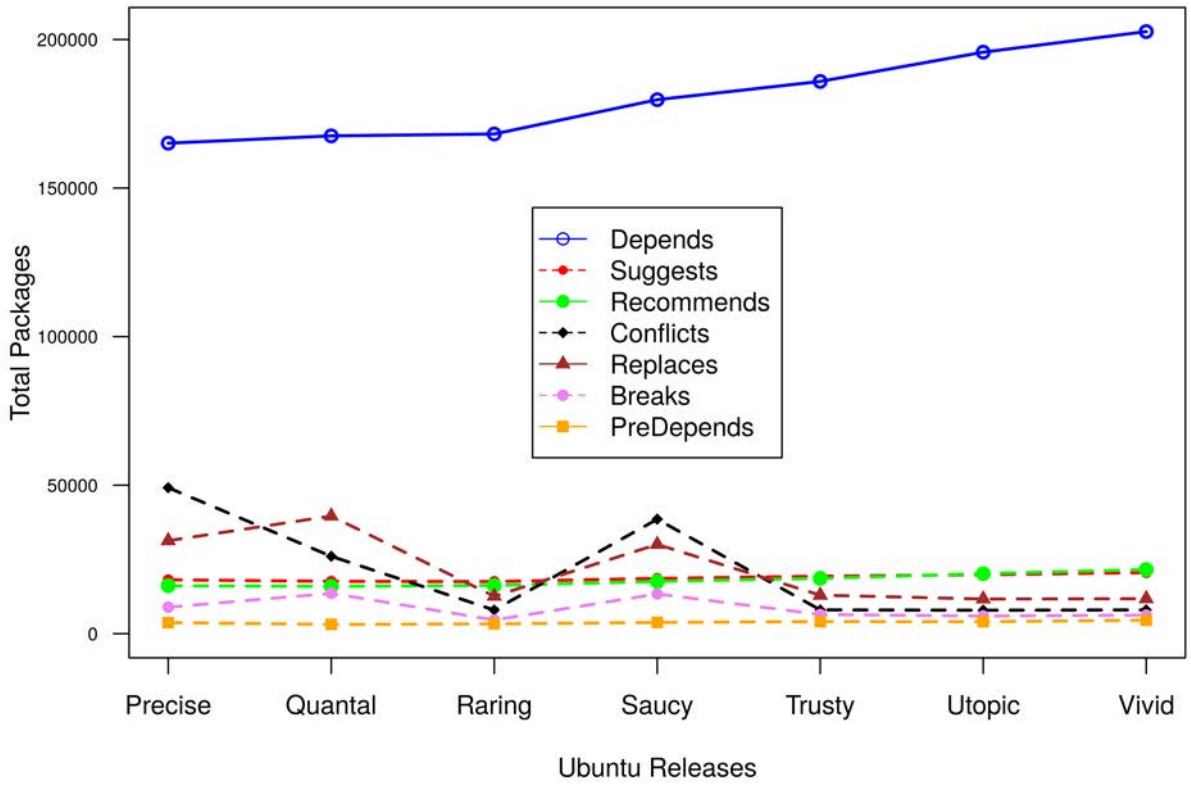


Figure 2.9: Ubuntu Release Evolution - Categorized into Their Dependency Rules

A large software repository such as the Ubuntu/Debian package repository, is an example of Software Product Line development [3, 58]. The package repository provides information about package dependencies. In addition, the Ubuntu/Debian package repository, like any other large software repository, provides information about product evolution that can be reconfigured and reused [63] for the benefit of further development. For example, a package may have functionality similar to others or replace other package functionality [106]. As shown in Figure 2.9, absolute package dependencies have increased to more than 200,000 dependencies and the conflict dependencies have decreased to less than 15.000 dependencies over the last 7 Ubuntu releases for a x64 bit environment. Figure 2.9 also

```
Package: wordpress
Priority: optional
Section: universe/web
Installed-Size: 10600
Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>
Original-Maintainer: Giuseppe Iuculano <iuculano@debian.org>
Architecture: all
Version: 3.4.2+dfsg-1
Depends: libjs-cropper (>= 1.2.2), libjs-prototype (>= 1.7.0), libjs-scriptaculous (>
= 1.9.0), libphp-phpmailer (>= 5.1), libphp-snoopy (>= 1.2.4), tinymce (>= 3.4.8+dfsg
0), apache2 | httpd, mysql-client, libapache2-mod-php5 | php5, php5-mysql, php5-gd
Pre-Depends: dpkg (>= 1.15.7.2)
Recommends: wordpress-l10n
Suggests: mysql-server (>= 5.0.15)
Filename: pool/universe/w/wordpress/wordpress_3.4.2+dfsg-1_all.deb
```

Figure 2.10: Debian Package Control Script - Wordpress in Ubuntu 12.10

shows that the Ubuntu Precise and Saucy distributions have conflict that has affected almost 50,000 packages, and the trend descents in the next Ubuntu distribution. Furthermore, the graph also shows that the Ubuntu Quantal and Saucy distribution has a higher number of replace and break packages, which exist to overcome the high number of conflict packages. In the case of the Ubuntu Precise distribution, the next distribution, the Ubuntu Quantal, has a high number of replace packages to reduce the conflict in the Ubuntu Precise. This further motivates the need for an automated approach for extracting feature models.

2.4.2 Debian/Ubuntu Package Repository

This section provides a description of the package control script that manages the package dependency within the Debian/Ubuntu package repository. This information is important because the configuration and deployment of this repository, prominently, is handled by these two elements. Further, this subsection also explains how to resize numbers of components that are retrieved from a large software repository.

Debian is a Linux distribution that provides software for installation in a package-based structure, where the packages are kept in a software repository. Each package is associated with a configuration file to manage dependencies, conflicts and suggestions of other Debian packages to be installed [106, 81], as shown in Figure 2.10.

In the Debian/Ubuntu package repository, each package is described in a package

control script. The control script file consists of information of the following,

- A **package name** as a unique identifier of each package.
- The **priority** to describe how important the package is for the installation of Ubuntu/Debian, that is categorized into optional, essential, required, important, standard and extra.
- The **section** informs the location of the package in the repository.
- The **installed-size** to indicate disk space that is required to install the package.
- The **maintainer** of the package that shows the contact name and email address.
- The **architecture** specifies the hardware system environment which this particular package was compiled for, such as i386.
- The **version** field gives the version number.
- The **depends** field presents a list of packages that need to be in place to satisfy dependencies in order to have a successful installation.
- The **description** gives a brief summary.
- The **filename** field gives information on the full name of file and its location in the software repository.

Overall, the relationship among packages from the Debian package system [106] is described by some rules, which are designed to indicate the level at which a package can operate independently of the existence of other packages in a system. Even though the relationships can be of different types and only one of them represents a strict dependency between packages, we will refer to all of these rules as dependency rules. Figure 2.11 shows an excerpt from the WordPress package's dependencies. The dependency rules are the following:

- *Package A pre-depends on Package B:* This rule forces the dependency to complete installation of the pre-depending packages before starting the installation of a package. For example, in order to successfully install A, B must be installed.

- *Package A depends on Package B:* B must be installed for A to run properly. In some cases, a package may have more than one dependency or also have dependencies to specific versions of a package. For example, Package A depends on version 1.1 of package B or the latest version of package B.

- *Package A recommends Package C:* the package maintainer may have privileges to add or change dependency to other packages that have similar functionality. For example, A is recommended to specifically use C, even though B has similar functionality to C.

- *Package A suggests Package D:* this rule lists related packages that may be installed to enhance the system, but that are not essential for the original package to work properly. As an example, D may enhance the functionality of A; however, A can run without D.

- *Package A conflicts with Package B:* a package may conflict with a particular package installed on the system, i.e., the installation of both packages on the same system will cause a problem. For example, if A is installed, then B will not operate properly. This may happen when a package has some improvement that may violate other functionality.

- *Package A replaces Package C:* this rule may be used to handle conflicts within packages. It is usually used to overwrite other files or to force an installation.

- *Package A breaks Package B:* the package maintainer software will not allow a broken package to be included in a configuration. For example, if A and B have problems to be configured together because of bugs within their code, then only either A or B can be included into a configuration.

- *Package C provides Package D:* several packages may have similar functions that may provide the same services. For example, C and D may have similar functions that provide web services. If both C and D are installed, only one of them will run as the default.

As an illustration, the Ubuntu 14.04 package repository has more than 46,000 packages that may increase and evolve continuously. For example, the "WordPress" package in "Ubuntu 12.10" and "Ubuntu 14.04" have differences in their packages library and database connections, as shown in Figure 2.12. Moreover, as each package may have dependencies, the relationship among packages is very complex. Therefore, a method to build a feature model from the Debian package repository *automatically* and to facilitate checking conflicts between an existing feature model and evolved versions of a system is necessary.

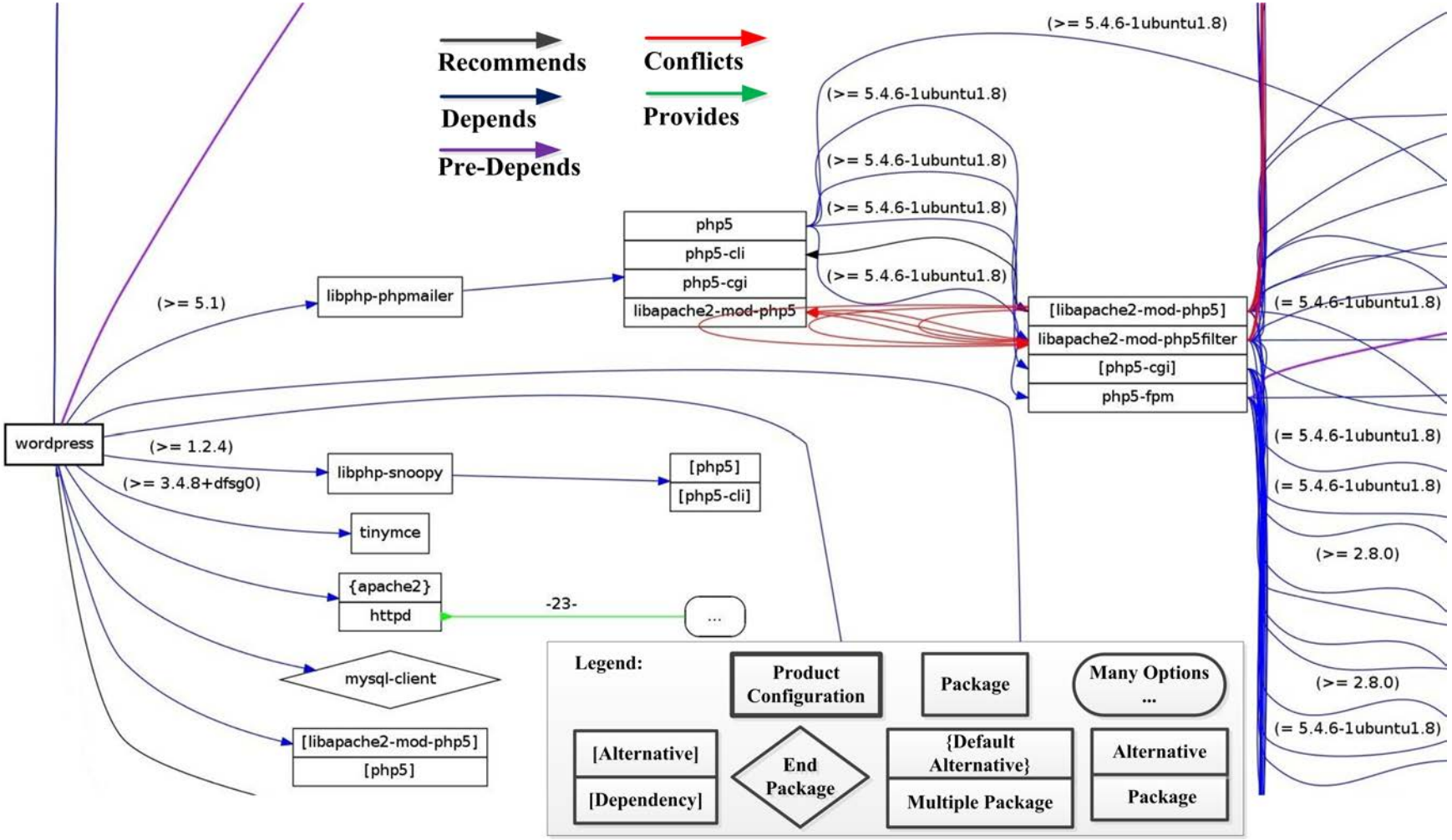


Figure 2.11: Excerpt from WordPress Package Dependency – Ubuntu 12.10

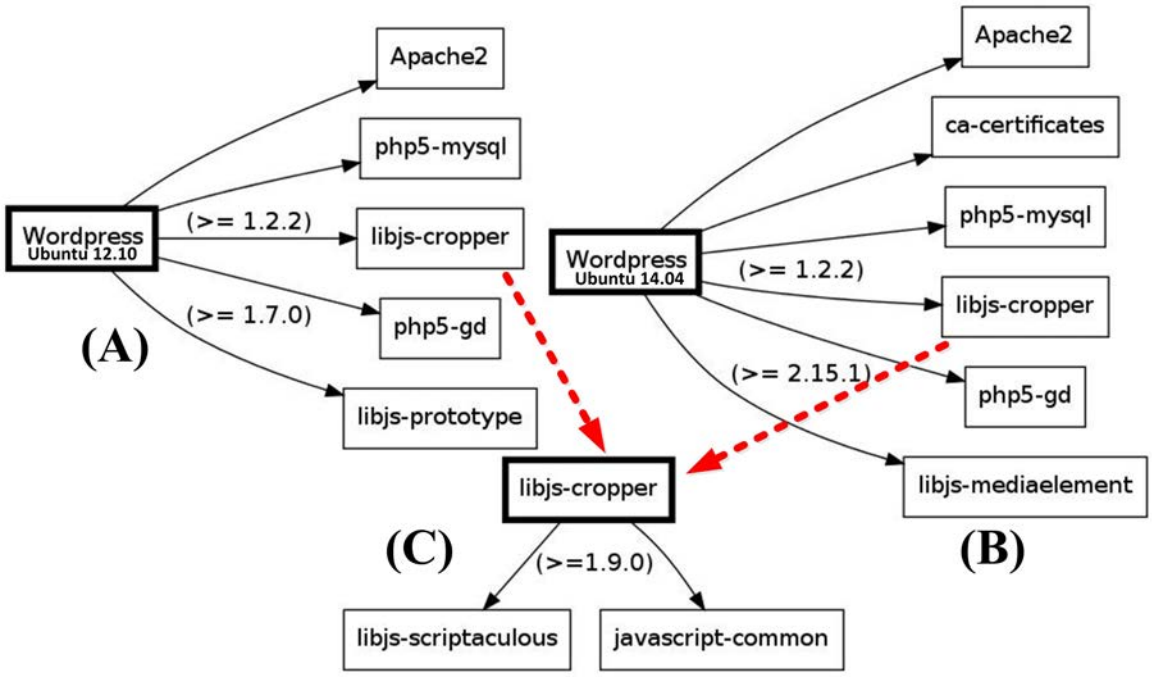


Figure 2.12: Package repository evolution – (a) Wordpress in Ubuntu 12.10, (b) Wordpress in Ubuntu 14.04, (c) Subgraph of a and b. Arrows represent dependency rules. All packages may have dependencies, but we show only the direct dependencies of WordPress and libjs-cropper.

2.4.3 Commonalities and Variabilities

We determine the commonalities and variabilities based on graph isomorphism, which describes the similarities between two or more graphs. Our graphs are ordered pairs of finite sets $G = (V, E)$, where V are the vertices or nodes and E are the edge sets. In our context, nodes represent packages and edges represent relationships between packages. According to Vasudev [212], two graphs G_1 and G_2 are isomorphic to each other if there is a one-to-one correspondence between their vertices and between their edges. Formally, graphs G_1 and G_2 are isomorphic if there is a bijection $f : V(G1) \rightarrow V(G_2)$ such that, whenever $uv \in E(G_1)$, then $f(u)f(v) \in E(G_2)$. If $G1$ and $G2$ are isomorphic, we write $G_1 \cong G_2$ or $G_1 = G_2$.

According to Harary [105], the two graphs G_1 and G_2 may have intersections if at least a subset of graph G_1 is isomorphic to a subset of G_2. The packages and their dependencies are sub graphs of the graph representing our cross-tree relationship. The

intersections of the sub graphs representing two or more packages and their dependencies are viewed as the commonalities of these packages, and will become mandatory features. The dependencies not in the intersection are the variability among these packages.

For example, in figure 2.12, sub graphs a and b are built from "Wordpress" dependencies of "Ubuntu 12.10" and "Ubuntu 14.04", consecutively, where the edge label is the version of the package. Graph c is the isomorphic subgraph of a and b, which represents the commonalities. The remaining features, "libjs-prototype", "ca-certificates" and "libjs-mediaelement", are the variability.

Discussion

In creating feature models using large software repository as a source, several researchers have provided approaches for modelling the Debian package repository for software product line development. Galindo et al. [81] developed a technique to detect uninstallable packages by analysing package dependencies using the SAT solver. Their approach includes detection of dead features in a Debian package configuration using a variability model language. They also map their Debian variability language into propositional formulae to make it compatible with the feature model constraints. Galindo et al. built a tool [73] using a meta-model to represent the packages and their relationships. DiCosmo et al. [58] managed the evolution within the repository by analysing the interdependent packages. However, these approaches require many steps to build the feature model from the repository, several of which require human intervention. Given that the Debian package repository is very large and evolves with time, creating and evolving feature models using existing approaches is an expensive task. In this work, we propose a graph-based approach able to automate the entire process of creating a feature model from the Debian package repository.

There are approaches that use graphs to retrieve dependencies automatically in order to build feature models [141, 189, 9]. Lopez-Herejon et al. [141] proposed a Graph Product Line (GPL) approach with basic components consisting of weighted, unweighted, directed

and undirected graph structures. They also used Edge-Neighbour Representation (GEN) to represent graph attributes such as edges and neighbours. Segura et al. [189] have automated the merging of features into a feature model using a graph transformation approach. They built a feature model with the help of the Attributed Graph Grammar (AGG) system [201]. Bachmeyer et al. [9] built a feature model from a conceptual graph to produce a natural and more easily expressed mapping to the problem domain. Even though promising, these approaches still require a good amount of human manual work for creating the feature model, which is a problem for large-scale systems. For instance, even though Segura et al. automates the merge process itself, each graph corresponding to a feature must be manually provided to the tool. Moreover, as none of the existing graph approaches have been developed for the Debian repository, they are not directly applicable to this repository.

Automated feature model extraction has been done on the Linux kernel and eCos embedded system [191, 119, 21]. Even though the Linux kernel and eCos embedded system work in a very different way from the Debian repository and their corresponding automated approaches for feature model extraction are not applicable to the Debian/Ubuntu packages repository, these automated approaches are worthy of mention because of their minimal need for human intervention during the extraction process.

She et al. [191] introduced a method to build a feature model automatically for the Linux kernel configuration by retrieving configuration options, which are known as configs, using the Kconfig model. The types of value allowed in the configs are boolean, tristate, integer (int or hex), and string [21]. She et al. transformed the dependency of configs into a cross-tree relationship and the Kconfig model into a hierarchy relationship.

Sincero et al. [119] built the Linux Kernel Configurator (LKC) to enable feature selection for the Linux Kernel. LKC parses and checks the dependencies of the kernel configuration. Sincero et al. map the feature relation to LKC to build the feature model.

Berger et al. [21] transformed the eCos components into a feature model using the Component Definition Language (CDL) [78], which defines the component relationship

of the eCos repository. The eCos system is an embedded system that supports more than 100 hardware architectures, where each package consists of a CDL file describing the variability of the packages. Berger et al. map the similarity of Kconfig and CDL into a feature model and constraints. In CDL, a component is a feature or group of features with options for distinct or individual configuration. The eCos configurator is a common tool to configure the eCos environment. To check the eCos configuration, a method called range fix was proposed by Xiong et al. [223] to validate the variability. For instance, it initiates the pre-defined rules when a component is added or activated in the eCos configurator.

The approaches of She et al. [191], Sincero et al. [119] and Berger et al. [21] are able to generate a feature model automatically, even though the scale-in/out of the feature model does not consider the options of pre-dependency or conflict packages before retrieving the package dependencies. Instead, they retrieve information on all existing packages, even the ones that would be not necessarily included in the feature model due to pre-dependencies or conflicts. This makes the retrieval process and the process of checking dependencies in the feature model expensive.

2.5 Machine Learning

This section provides a brief but essential background of Machine Learning. It begins with definitions and essential concepts, and continues with methods to build a machine learning ensemble. Then, methods to measure the performance of the machine learnings' outcome is presented. The definitions are as follows:

(1) According to Arthur Samuel [188] Machine learning is "the field of study that gives computers the ability to learn without being explicitly programmed".

(2) In a more formal definition, Tom Mitchell [157] proposed that "a computer program is said to learn from experience E with respect to some tasks T and some performance measure P, if its performance on T, as measured by P improves with experience E",

where tasks that can be performed by machine learning approaches include prediction, classification and planning.

2.5.1 Machine Learning Essentials

The Machine learning algorithms are typically divided into two broad categories, which are Supervised and Unsupervised learning.

2.5.1.1 Supervised Learning

In Supervised learning, the algorithm uses a dataset to make predictions, which includes input data and response values. These algorithms build a model to make predictions of response values from a new dataset. If necessary, a test data set is used to validate the model. Decision tree, Regression tree and Random Forest are examples of Supervised learning.

Decision tree learning is one of the most widely used and practical methods in machine learning for inductive inference [157]. A decision tree [185] represents a function that has input is a vector of attribute values and output is a decision, in a single output value. In the decision tree, a learning method to approximate the discrete-valued, the learning function is represented as a decision tree. Further, the input and output values of a decision tree can be discrete or continuous. The decision tree performs a sequence of tests before making a decision, where by each node in the tree corresponds to a test that has the value of an input attributes. It starts at the root node of the tree, moving down to the branch that corresponds to the value of the attribute after testing the attribute specified by the root node. This process is then repeated for the whole subtree.

A regression tree is a decision tree with a target variable that can take typical real numbers of continuous values, making predictions [132] based on the average value of examples that reach a leaf of a tree. A regression tree combines the strength of decision trees to model numeric data. This algorithm also operates an automatic feature selection to allow the approach to be utilised with a very large number of features, and also interpret

a model without statistical knowledge [132]. However, this algorithm requires a large amount of training data to provide a good prediction result.

Random forest [31] is an ensemble learning approach that combines several tasks, e.g. classification and regression, and operates by constructing decision trees in a form of nearest neighbour predictor. The random forest algorithm begins with a decision tree, where an input is traverse down the tree, the data grows the tree into smaller sets. A random forest is able to deal with unbalanced and missing data, and its runtime is also quite fast. Nevertheless, random forest has weaknesses when it is used for regression, such as it cannot predict beyond the range in the training data, and when it deals with noisy data, it may over-fit the dataset - over-fitting arises when a statistical model shows noise instead of the basic relationship.

2.5.1.2 Unsupervised Learning

The unsupervised learning algorithms address a machine learning task that aims to describe the associations and patterns in relation to a set of input variables [172] without explicit supervision, as exist in the supervised learning. These algorithms use a descriptive model, i.e. ranges of ages between 2 - 5 is toddler and 8 - 16 is teenager, that gives insight into summarising data in new and interesting ways. Cluster analysis is the most common method used by unsupervised learning to find hidden patterns or grouping data in an exploratory data analysis. Furthermore, the cluster models a similarity using defined metrics such as probabilistic distance. Hierarchical and K-means clustering are the most popular in clustering algorithms.

Hierarchical clustering [172, 86] aims to build a hierarchy of clusters by grouping a dataset over a variety of scales. Typically, this cluster algorithm uses a bottom-up and top-down approach. The bottom-up approach begins the observation in its own cluster, and then merges pairs of clusters as one moves up to the hierarchical structure, recursively. Furthermore, the bottom-up approach requires a distance metric to measure a distance between observations, where distance in the same cluster should be much smaller than the

distance of different clusters. Subsequently, the bottom-up approach also requires linkage methods that use a linkage rule to calculate the inter-cluster distances, to determine distance between two clusters. In the top-down approach all observation begins in a cluster, then splits recursively as one moves down to the hierarchical structure.

K-means clustering [172] aims to partition a set of data points into clusters in order that each dataset belongs to the cluster which acquires the nearest mean. In general, given n data points $X_i, i = 1..n$, K-means clustering aims partitioning in k clusters. The aim is to assign a data point to each cluster, so that n data point is partitioned into $k(\leq n)$ sets $S_i, i = 1..n$. K-means is a clustering method which aims to find the positions $\mu_i, i = 1..k$ in the clusters, to minimise the square of the distance from the data points to the cluster. The K-means clustering objective is to find:

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{x \in S_i} ||x - \mu_i||^2 \tag{2.3}$$

where μ_i is the mean of data points in S_i

2.5.2 Machine Learning Ensemble

A machine learning ensemble combines the predictions of several base estimators built with a given learning algorithm to improve robustness and accuracy over a single estimator. The ensemble method [59] is used to improve the performance of a model, e.g. classification and prediction, to select the optimal features and error-correcting. This system is quite useful when dealing with a large volume of data that have problems with accuracy, such as when a single classifier has performed poorly with too much data. Then, the data sets can be partitioned into smaller subsets.

Bootstrap aggregating (bagging) [29], is an example of a machine learning ensemble technique designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. Bagging reduces variance of a prediction by generating additional data for training from the original dataset using combinations

with repetitions to produce multisets of the same cardinality or size as the original data.

In bagging, given a standard training set S of size n, bagging generates W new training sets S_i, each of size n', by sampling from S uniformly and with replacement. By sampling with replacement, some observations may be repeated in each S_i. If $n' = n$, then for large n the set S_i is expected to have the fraction $(1 - 1/e)$ of the unique examples of S, the rest being duplicates.

2.5.3 Performance Measurement

Performance measurement of machine learning evaluates the learning algorithms[157, 117, 110] for their accuracy (percent correct over all test instances) and precision. The performance measures used in this study were Mean Absolute Error (MAE), Root Mean Square Error (RMSE) and Median Magnitude of the Relative Error (MdMRE):

- $MAE = \frac{1}{T}\sum_{i=1}^{T}|\hat{y}_i - y_i|$;
- $RMSE = \sqrt{\frac{\sum_{i=1}^{T}(\hat{y}_i - y_i)^2}{T}}$; and
- $MdMRE =$ Median $\{MRE_i | 1 \leq i \leq T\}$, where
  $MRE_i = 100 \cdot \frac{|\hat{y}_i - y_i|}{y_i}$,

where $\hat{y}_i$ is the predicted CPU power, y_i is the actual CPU power and T is the number of examples used for testing. MAE is a measure recommended for being unbiased towards under and overestimations [192]. RMSE is a popular measure in the machine learning community which emphasizes large errors more. RMSE is suitable for evaluating how close the observed data points are to the models' predicted values [124, 43] and MAE to describe uniformly distributed errors [43]. Both, RMSE and MAE are sufficient to measure performance using the data set that is created, based on a combination of workload, configuration of software components and CPU power usage. MdMRE can be biased towards underestimations, i.e., it may favour methods that make underestimations [192]. So, MAE and RMSE are more appropriate for comparing methods, whereas MdMRE was

included only to give a general idea of the performance in terms of percentage of the actual CPU power levels.

Discussion

Machine Learning (ML) approaches have been investigated as decision support tools for performing several software engineering tasks. In order to contextualise our work, this section explains a few of these. An example of a software engineering task that can be formulated as a ML problem is software defect prediction [148, 103]. This task can be seen as a classification problem (prediction of whether or not a software component is likely to contain faults), regression problem (prediction of the number of faults in a software component) or ranking problem (task of providing a ranking of software components according to how faulty they are likely to be). Software defect prediction is frequently a problem with skewed distributions, where there are much fewer examples of defective software modules than non-defective ones. This has been shown to affect prediction results considerably [143]. Conclusions in this context can therefore be very different from conclusions obtained in studies for predicting energy consumption, such as our work. Examples of approaches used for software defect prediction are naive bayes, decision trees and logistic regression, as well as other approaches specifically designed to deal with skewed distributions [218].

There have also been several studies in the area of ML for software effort estimation [148]. One of the similarities between effort estimation and prediction of energy consumption is that both can be formulated as regression learning problems. A key difference is that the data sets available for building software effort estimation models are typically very small. For instance, it is not uncommon to have data sets with only around 15 projects collected within the company for which estimations should be provided. ML algorithms frequently struggle to achieve good predictive accuracy when training sets are very small, as small training sets do not represent the whole data distribution well. As a result, locality-based approaches that performed estimations by finding the software

projects most similar to the project being estimated (e.g., RTs and k-nearest neighbour) have been shown to achieve better results [126, 154]. Combinations of locality-based approaches with ensembles such as bagging have also been shown to improve software effort estimations [154]. Transfer learning approaches, which use data from different companies to create predictive models, have also been showing promising results [127, 155, 210].

Kolesnikov et al. [128] investigated the prediction of software quality attributes such as performance, reliability and footprint associated to configurations. Different from previous work on energy prediction [162], their approach performs qualitative rather than quantitative predictions. For that, it requires the definition of quality categories. For example, a given configuration could be assigned to either the category of "high risk of high-severity failures" or "low risk of high-severity failures". According to the authors, their approach could result in low predictive accuracy if quantitative predictions had been attempted, because it does not use runtime information such as workload as input variables, only the configuration.

A problem more closely related to the prediction of energy consumption is quantitative prediction of software systems' performance (e.g., response time in seconds). An example of work in this area is Guo et al. [99] who apply Classification and Regression Trees (CART) [30] to model the nonlinear correlation between configurations and performance. CARTs were then used to predict the performance associated with configurations. Guo et al. [99] showed that CARTs were effective and to be recommended over a previous approach called SPLConqueror [193] when the number of features is large, due to shorter time needed for building the predictive model. CARTs are RTs very similar to the ones used for measuring energy consumption [162].These results therefore corroborate the good predictive accuracy results obtained using RTs in the context of energy prediction [162]. Nevertheless, one of the key differences between the problem investigated by Guo et al. [99] and the problem being investigated in this thesis is that we consider not only the configuration, but also the workload when making predictions of energy consumption.

2.6 Chapter Summary

This chapter provides a literature review of energy reduction and reuse of combination software components within virtualised environments. Following this, it provides a brief introduction into software product line engineering and dynamic software product line engineering. In relation to the enhancement of a software product line model with energy usage, our survey revealed that existing works had proposed several approaches, such as a feature library and a feature composition framework, to design and investigate the reduction of energy for a product line. However, none of the existing works provide a method or an approximation approach to find energy expenditure from a combination of features associated with workload and energy usage from the virtualised environments. The following chapters will discuss our approaches to fill this gap.

CHAPTER 3

MEASUREMENT OF ENERGY USAGE IN A VIRTUALISED ENVIRONMENT

This chapter introduces an approach to measuring energy by simulating a workload for the virtualised system using a set of systematic steps, which include environment settings and parameters, such as a web directory and authorisation elements. One of the main outcomes of this measurement is the energy usage for each distinctive process within a virtualised system. This chapter also provides the basic material of this thesis, the energy usage of combinations of software components, in order to evaluate the methods for Chapter 5 and 6, which discuss the Software Product Line and Dynamic Software Product Line.

Our work is achieved through the following: Section 3.1 presents the scope of our technique that requires the system to allow the activity of the measurement of energy in a virtualised environment. In Section 3.2, a technique to generate a workload with a set of a plan is presented. This is considered an important effort to measure energy usage, as it helps to execute tasks involved within the target system. Section 3.3 introduces an approach to measuring energy in virtualised environments in order to obtain the energy expenditure of the Virtual Machine instances, in response to the workload that is sent in an incremental way. This measurement captures energy usage of each individual process in the system, such as the web server and the database management system. We save these results in files for analysis via Machine Learning. In addition, our measurement

often results in a large number of files. For example, in one case a single configuration produces 1000 files. As a result, in Section 3.4, we explain how to filter these files to find suitable information on the corresponding product configurations, and how these files can be used to prepare data sets for the Machine Learning process. In order to evaluate our measurement approach, we conduct two experiments using the technique explained in this chapter. Section 3.5 reports the first experiment comparing the measurement results within the Cloud and virtualised environment. Section 3.6 reports the second experiment within the virtualised environment that uses the iteration technique to reduce the effect of the C-state CPU [116] in the measurement results. Finally, Section 3.7 summarises the chapter.

3.1 Description of Our Approach

This section presents the description of our technique to measure energy in a virtualised environment. Such a technique runs on top of the Ubuntu Server operating system, and the target of measurement is accessible within the system or through the computer network, such as a virtual machine running a web application in the Cloud system. Firstly, a workload can be generated to execute tasks within a virtualised environment. This technique should provide a workload that increases gradually and smoothly - in a linear way - in order to simulate the increase in the number of requests to a virtualised system. Secondly, there is a mechanism to measure energy, which captures energy usage for each individual process, within a virtualised environment. In addition, the component of measurement includes the system environment and operating system within the system. For example, an HTTP server such as "apache" consumes a different amount of energy when running on top of a different hypervisor and Linux distribution.

Now we give an example of our approach. In our measurement, we use Powertop [200], an Intel Open-Source software power meter tool, to capture energy usage within a virtualised system. This measurement tool is required to run on top of the Ubuntu (Linux)

operating system that uses the Linux Kernel (not older than the version 2.6.38) [200]. Furthermore, in a measurement that includes two or more Virtual Machines (VM), a workload is sent using network traffic to execute tasks within the system under examination. An example of this measurement is a Web-based system that receives requests from a Client browser. The details of the workload will be explained in Section 3.2.

This work measures energy in a Laptop and Cloud system. In a Laptop, the software power meter captures the information of the electrical power drain from the battery, which is provided within the log files of the Linux system. In the Cloud system, the measurement uses the Advanced Configuration and Power Interface (ACPI) [152] as an interface between the software power meter and the operating system.

Our measurement approach can be implemented in two modes, manual and automatic. In the manual mode, the system under examination runs a number of activities within its environment, such as a drawing application. During drawing activities, the application executes tasks within the system, where the software power meter runs continuously to capture the consumption of energy. In the automatic mode, this technique requires having at least two Virtual Machine (VM) nodes; One VM node to send the workload and the other one to respond to the requests. Further, the workload tool sends a load in order to execute tasks within the target system. As a result, this technique requires an application that receives input from an external or remote system, such as a web-based system. Moreover, the workload sends a load to the system under examination using network traffic as a communication medium. For example, a typical HTTP server uses port 80 of the Transmision Control Protocol (TCP) to receive requests from clients.

3.2 Workload

This section introduces a technique to generate a workload for a virtualised environment. Such a technique simulates an end-user accessing an HTTP server. In generating the workload, a set of steps is implemented for each specific domain of application, in the

direction of the execution of tasks within the system under measurement.

According to Ortiz [176] "a workload model, such as a web workload, is an abstraction of the real workloads that reproduce user behaviour and ensure that a particular (web) application performs as it would do when working with real users". In addition, a workload reproduces the behaviour of a system [139]. To measure the consumption of energy for any given application, a suitable workload must be provided to trigger the execution of tasks within the system under examination; otherwise, the existing combination of software components cannot be measured properly. For example, a web application consists of a database management system and an HTTP server. Hence, a workload should execute tasks that associate to both the database management system and the web system. These tasks may be presented as a sequence of activities, such as a request to log in, open a web page and then fill in a message in a web forum, before logging off from the system. However, a system may perform well in one workload, and poorly for another workload [75]. Therefore, the workload should target the system under measurement based on its existing component configuration.

In our approach, a workload scenario [187] enables us to simulate the end-user activity to execute tasks that are associated with the components in the system under examination. In addition, a workload scenario [187] also has the capability to simulate the steps of an application, which executes tasks as required by the system. In an online learning system, an example would be an end-user login to the system that executes the authorisation feature. This feature interacts with the database management system and security system to get the credential information, such as the username and password that are saved in the database storage. Furthermore, each password is encrypted using an encryption technology, e.g. a database management system such as MySQL supports Secure Hash Algorithm 2 (SHA2) [198], to secure the credential entity. When an end-user has been granted access to the online learning system, they will have the privilege of accessing some web pages, such as courses and discussion forum web pages. To finishing, the end-user can log out from the online learning system.

The illustration of the workload scenario, as shown in Figure 3.1, uses JMeter [6] as a workload generator tool that supports different protocols and concurrent threads. JMeter uses a Java [64] application to provide load testing, and was originally designed for testing web applications. Nevertheless, JMeter does not perform all the features supported by web browsers, such as Javascript. As our measurement is independent from the workload technique, we can use different workload tools that support distinctive web utilities, such as HP LoadRunner [109] and Apache benchmark [7].

The workload tool predefines all target information, such as folders of the web pages, web server port, and user credentials, as references to execute tasks within a system under measurement. As an example, the target of the system being measured requires credential data, e.g. a distinct username and password. These data, which are consistent with the workload generator and the target of measurement, should be provided beforehand. As an example, for the measurement of the online learning system, in the workload tool side the user credential data is setup as a file reference. In the target of measurement side, the credential data is stored in the system, where both sides must have a similar and consistent credential data. In an online learning system [159], a test course generator tool has options to generate 100, 200 or more pairs of distinct usernames-passwords as credential data for workloads, where numbers of users accessing the system correspond to the time spent for the whole workload to finish the activities. Therefore, the greater the number of user credential data are added into the workload, the more loaded the infrastructure will be.

In a workload scenario, a load is sent to the object of measurement via a network. This requires information on how a load can be sent, what is allowed for each transaction and what protocol should be used. We adopted a traffic model [139] in order to simulate the activity of sending a workload to a system, as explained in the next subsection.

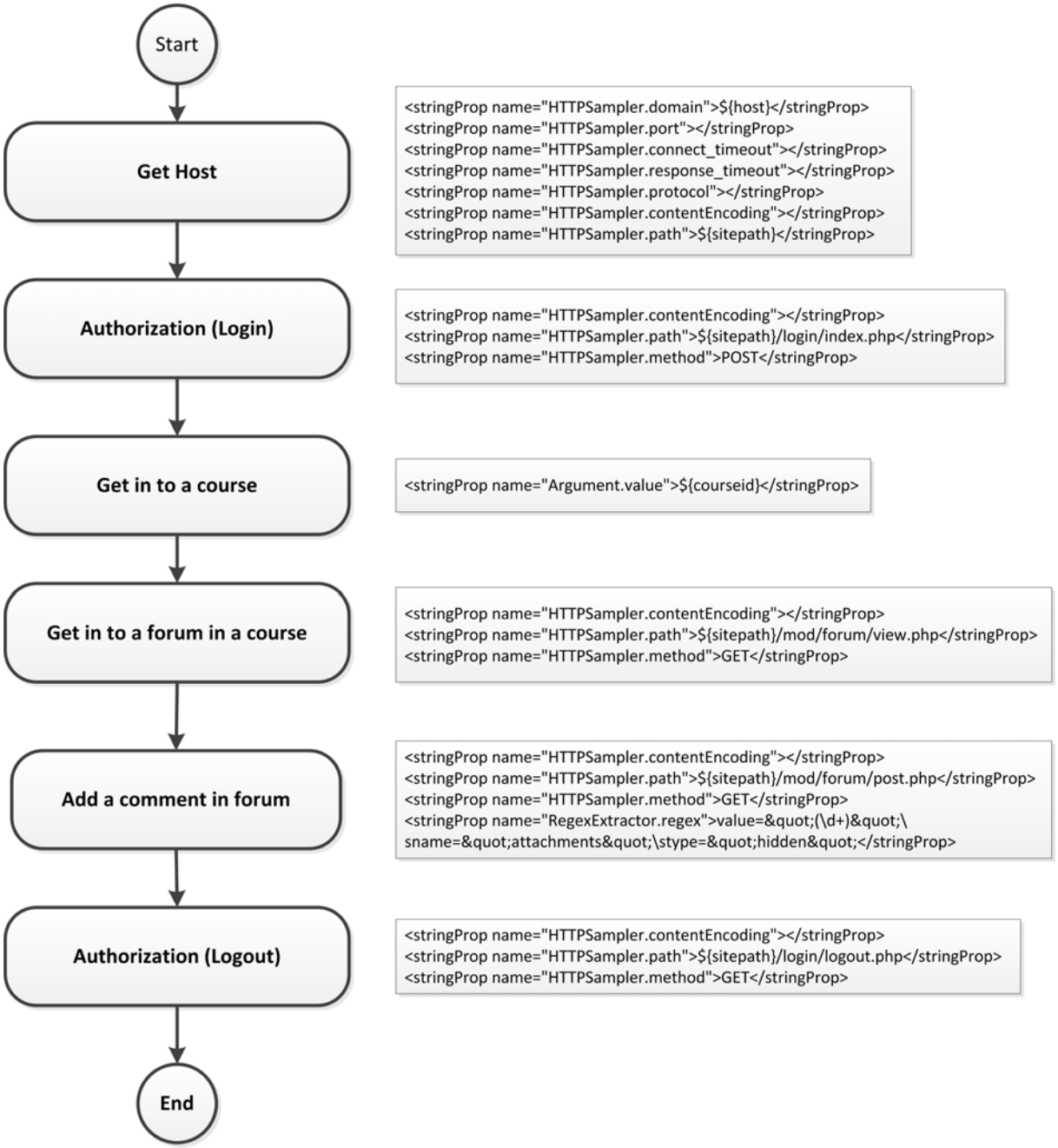


Figure 3.1: An example of a workload scenario for an Online-Learning System - MOODLE [182], the left part is the scenario flow of the system and the right part is the example of the scripts in the workload tool [6].

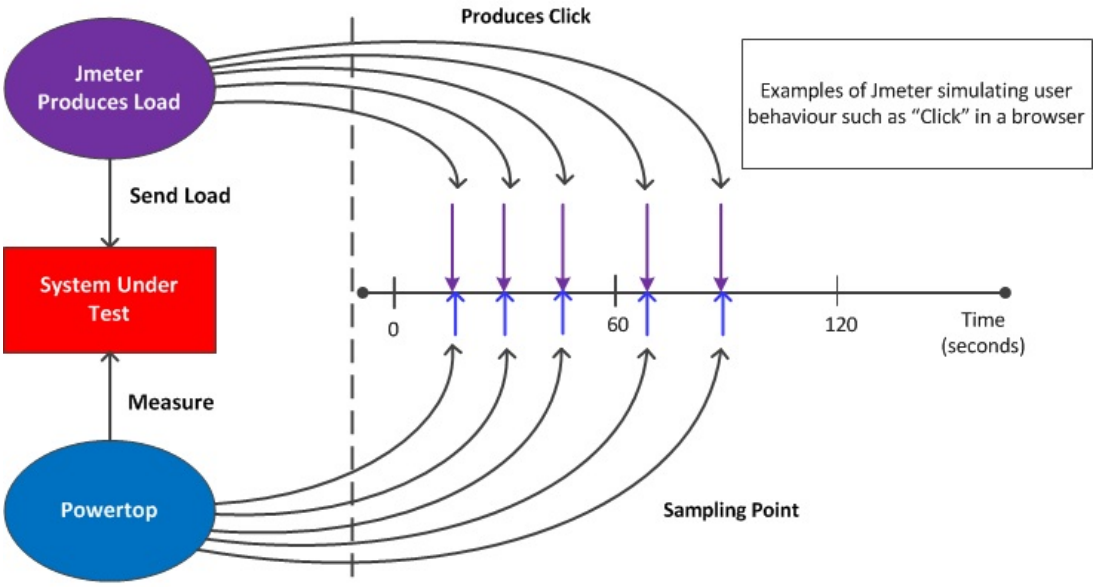


Figure 3.2: Workload and Measurement.

3.2.1 Traffic Model

In generating a load to a system under examination, a workload tool implements a traffic model [139] as a way to simulate network traffic when a client application such as a web browser accesses an HTTP server to execute tasks involved in each workload scenario. A traffic model activity mimics a user navigating the request to an HTTP server, such as when the user arrives and how requests are generated. In this work, we use a workload tool, JMeter [6], to create a traffic model that generates a load to execute tasks within the target system. As illustrated in Figure 3.2, JMeter [6] sends the workload to a system that requests a web page to an HTTP server for a specific time span, such as a request each second for 60 seconds. In the same interval, the software power meter, Powertop [200], captures the consumption of energy. The details of the measurement procedure will be explained in Section 3.3.

In our technique to measure energy, we aim to obtain information about the energy within a virtualised environment. In order to increase the number of users and change the types of their requests, we use a scenario that simulates users accessing a web system, such as an eLearning system. Firstly, users accessing a login page, when they get through, then

secondly, lists all available courses. After that, users fill in the discussion box in the forum page before log out from the system. We generate a workload in a linear smooth way, where the load is sent in an increasing sequence order from low to high, such as from 1 user request per second to 100 user requests per second. As a result, the workload change can be monitored as a linear increase of resource usage within the target of measurement.

3.3 Measurement of Energy Usage in Virtualised Environments

This section introduces a technique to measure energy usage in the virtualised environments with a workload. This measurement technique uses a software power meter tool from Intel open-source project (Powertop [200]) to capture information of power estimation that is spent for each individual process.

As explained in Section 2.1, a virtualised system, such as Cloud, shares its infrastructure to provide the configurable computing resources. It is non-trivial to measure energy in order to investigate the behaviour of software component configurations in a virtualised environment. However, this measurement requires a technique that is different from a conventional energy measurement. For example, we cannot use a Wattmeter device because it is a shared infrastructure and virtual machines can move from one host to another.

In this thesis, we propose a measurement technique for energy consumption in the virtualised environment for individual processes. This technique captures energy usage of an individual process of an application within a virtual machine. For example, Nginx [164] an HTTP Server that processes tasks such as Master and Worker. Under a given workload, each one of these processes consumes a different amount of energy. Such a technique allows us to measure the energy consumed by the load generated from a workload tool [6]. As a result, we will be able to select components in the software application that consume less energy, to be included in a virtualised environment.

This section also presents a technique to measure energy in the virtualised system of a Laptop. We demonstrate the measurement of the virtual machine instance within a different hypervisor, the Kernel-Based Virtual Machine (KVM) [112] and Virtualbox [219]. In this environment, the software power meter [200] captures the electric current drainage from the battery. For the measurement in a Cloud system, we take advantage of the virtual machine power management [207] where a typical power management in the Cloud is used to stop or start the virtual machine instances. The measurement of energy in the Cloud and the virtualised system makes use of the Advanced Control and Power Interface (ACPI) [152], an open standard power management specification. ACPI provides instruction lists which are parts of the system firmware, that are accessible by the kernel of an operating system to execute some tasks. Such firmware instructions allow an operating system to monitor power and electric current, and turn-off or -on a virtual machine.

It is worth noting that Powertop [200] needs to be calibrated once before it measures the energy within a new environment. The calibration generates a Powertop Reference file that is read as a reference every time we start Powertop.

3.3.1 Steps of Measurement

This section presents a set of steps to measure energy in a virtualised environment. We categorise the measurement into two activities; the first is sending the workload to the target of measurement for a specific time-span. This activity generates the workload that sets up the types of load based on the domain of the tested application, such as an online learning system, as described in Section 3.2. The second activity captures energy usage using a software power meter tool within the system under measurement. It is important to note that both the workload and software power meter tool operate over the same interval of time. Therefore, the results of this activity represent the execution of tasks within the target system under measurement.

Figure 3.3 shows that the workload tool sends a given load to the application in-

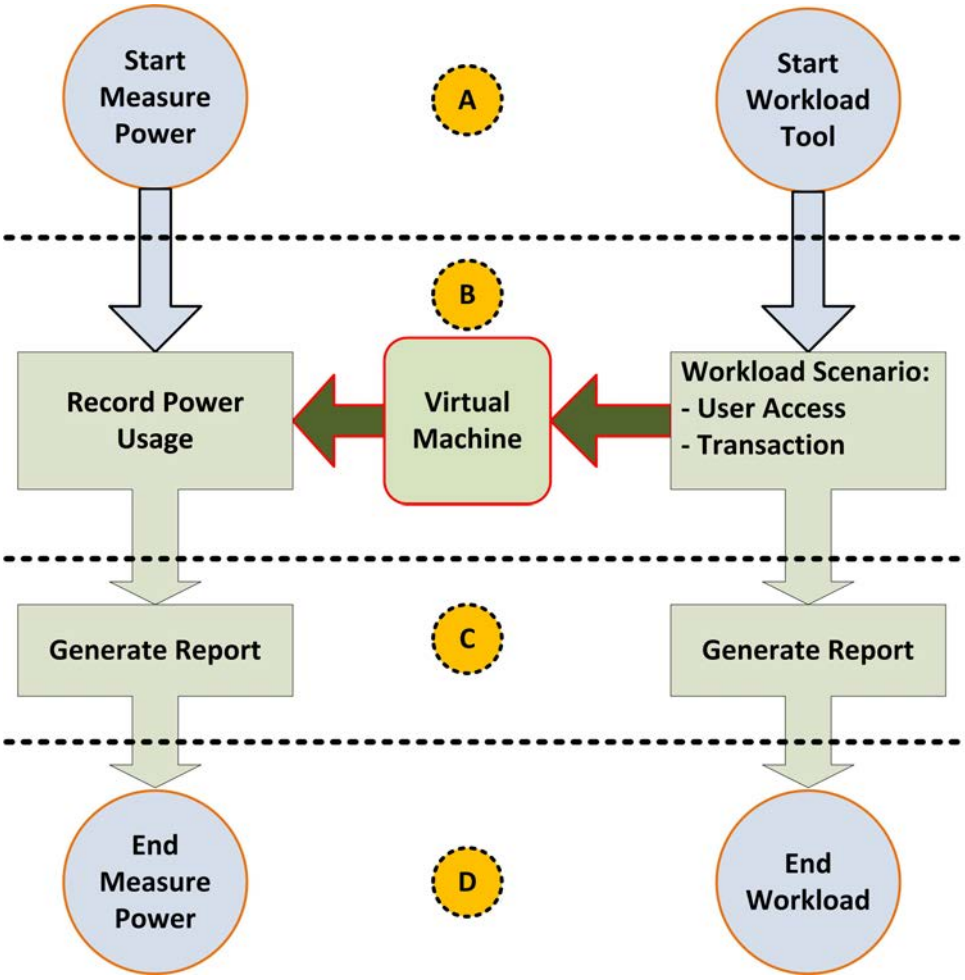


Figure 3.3: Steps to measure energy usages in a virtualised and Cloud environment.

side the virtual machine. The load simulates users, for example, accessing a web page within an HTTP server. At the same time, the software application for measuring energy simultaneously records the energy consumption in the system.

In our measurement approach, we implement a workload scenario to perform several activities in a consecutive way. As shown in Figure 3.3, the steps to measure energy usage in the virtualised environment, are as follows:

(A) We start the software application for measuring energy and the workload tool,

(B) The workload tools (acting as a client) send a request to the system, for example to access web pages and files on the Web server. At the same time, the software application to measure energy captures the power usage of each process running in the virtual machine. In this step, some amount of workload is sent ten times to

overcome the tickless kernel effect [116, 98].

(C) The workload tool and the software application for measurement generate their reports. The software application for measuring energy reports the amount of energy that is consumed per individual process that is generated every one second. The workload tool then generates information from the overall sending process.

(D) The measurement ends when all the reports are generated. The software power meter and the workload tool are switched off.

3.3.1.1 Iteration on Measurement

In our energy measurement, we use Powertop [200] as a software power meter. Powertop was originally designed to diagnose power consumption and power management in a laptop, which includes software applications and active components in the system, by means of an approximation approach.

Powertop monitors the device and software activity [200] that are associated with the CPU processes to predict the energy usage. Since this tool reads the energy usage of a CPU, Powertop is also influenced by the CPU operating state – CPU C-State to save power – where in the Linux operating system is adopted to manage the idle system that is known as Tickless Kernel. In the Tickless Kernel [116, 98], an idle CPU will remain idle until a new task is queued for processing. Further, the CPUs that are in the lowest power state can remain in this condition longer [200]. As a consequence, in capturing the measurement data, Powertop may have a delay that creates high or low spikes in the measurement results.

In order to reduce the possibility and frequency of spikes during the measurement, we repeat the measurement procedure at least ten times for each workload. This is because when we measure less than 10 iterations with low workloads (between 1 – 20 users per second) Powertop fails to capture the information of energy, whereas more than 10 iterations with high workloads (between 80 – 100 users per second) will produce too

much information. This is based on our experience in measuring energy in the virtualised systems [161, 162]. As an example, a workload to simulate five users accessing a system is repeated ten times, producing ten value items of power consumption. We then retrieve the median data for each workload [162] for the next research activity.

3.3.2 Measurement in a Laptop

To measure energy usage of a virtualised environment in a laptop, we use an HP Elite-book 8440P with i5 processors and 4 GB memory. The host computer uses Ubuntu 12.10_x64 with Kernel-based Virtual Machine (KVM) as a hypervisor for the virtual environment. The virtual machine instances are installed with Ubuntu Server 12.10_x64, and configured with 1 CPU core, 1 GB of memory and 10 GB of storage.

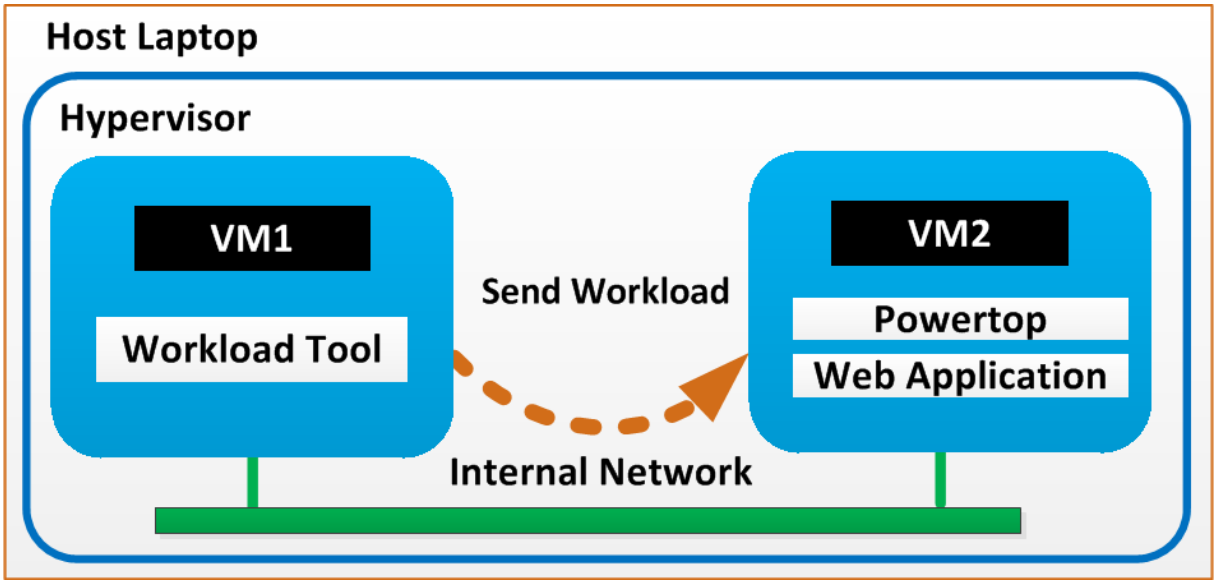


Figure 3.4: To measure energy of a virtualised system in a Laptop.

Figure 3.4 depicts the measurement of energy in a laptop. This measurement includes at least two virtual machines (VM), the first machine (VM1) simulates the incoming requests by sending a workload to the measurement object, the second machine (VM2) running the web application that receives the incoming workload. In the second machine (VM2), the software power meter, Powertop, captures the energy usage for each second during measurement. We take advantage of the ACPI (Advanced Control and Power

Interface) [129, 152] to have the real time power consumption from the battery. To standardise the software application for measurement, we calibrate it against the current drainage of the Laptop's battery for several minutes. The outcome of the calibration is saved into a file called Power Reference.

In the KVM-based laptop, a hypervisor - we use KVM - is installed on top of an Ubuntu Server. The system receives its workload from another Ubuntu Server, and measures the power usage using Powertop [200], where the software application for measurement runs within each system. Therefore, each measurement result is dedicated to a particular software configuration within distinctive VMs.

3.3.3 Measurement in a Cloud System

In a Cloud system, the software that measures the power usage relies on the Advanced Control and Power Interface (ACPI) [129, 152], a platform-independent interface, which is usually used by the Cloud system to switch the virtual machines on or off [55]. As explained in Section 3.3, ACPI exists both in Cloud and Laptop, but it is implemented and used differently. Further, the ACPI in Cloud and virtualised environment has similar features, such as turning a virtual machine on or off.

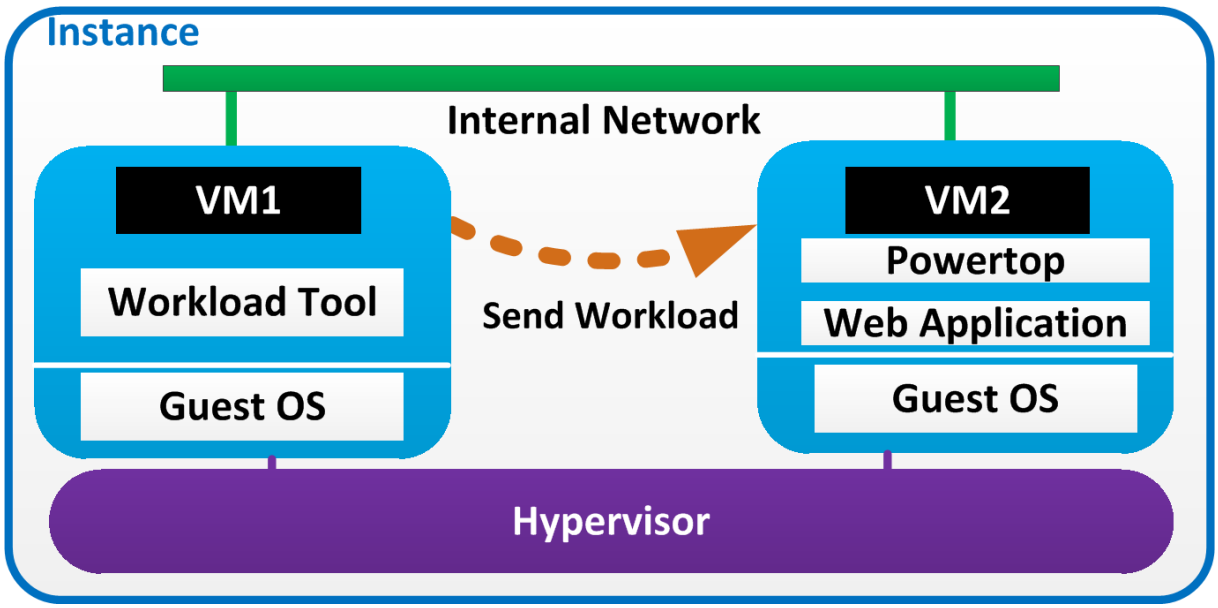


Figure 3.5: To measure energy of a virtualised system in a Cloud system.

Figure 3.5 shows the mechanism to measure energy in a Cloud system. In measuring energy for a Cloud system, when the software application to measure energy is run for the first time, it reads the Power Reference File and then captures the virtual environment information such as the number of virtual processors and memory capacity. After that, the software application to measure energy will renew its power reference file and become ready to measure the energy consumption per process. In this work, we use the virtual machine within the Elastichost [67] Commercial Cloud computing environment because it uses KVM as its hypervisor, which is similar to the one in the laptop. A snapshot of the measurement is shown in Figure 3.6, and an indication of the virtualised environment can be found in the virtio0-request [131] process. In addition, in our approach the Kernel updates will not affect the measurement as long as the software application for measurement [200] is not updated.

```
 102 mW  ,   2.4 ms/s, , , , , Process, inet_gethost 4 ,
49.1 mW  ,   1.1 ms/s, , , , , Process, [migration/0],
37.2 mW  ,   0.9 ms/s, , , , , Process, [kworker/0:2],
31.8 mW  ,   0.7 ms/s, , , , , Interrupt, [9] RCU(softirq),
31.0 mW  ,   0.7 ms/s, , , , , kWork, e1000_watchdog,
21.0 mW  , 489.9 us/s, , , , , Interrupt, [1] timer(softirq),
15.0 mW  , 349.8 us/s, , , , , Process, [ksoftirqd/0],
9.47 mW  , 221.1 us/s, , , , , Interrupt, [41] virtio0-requests,
5.10 mW  , 119.1 us/s, , , , , Interrupt, [11] eth0,
```

Figure 3.6: Virtual I/O - power per process in a Linux virtualised environment.

The measurement in the Cloud commercial provider uses a similar workload scenario to the one implemented in the Laptop environment, and uses Powertop [200] to capture energy usage. All the measurements and the workloads are run in Ubuntu Server 12.10_x64 which is provided by the Cloud provider as a virtual machine image.

3.4 Extraction of Data from the Measurement Results

Our measurement of energy produces ten files for each folder with a total of 100 folders, where each file has at least 110 lines of text. Therefore, the measurement for one configuration produces 10×100=1000 files, where each file has 110 lines of text, leading to 110×1000=110.000 lines of text per configuration. In this work, we measure 12 configurations for a small product line model.

This section discusses steps to extract the data from the energy measurement results. The measurement of energy, as explained in Section 3.3, produces a large number of files. This requires us to extract the data as needed for our next research on the prediction of energy, such as the CPU power usage that corresponds to a set of workloads. These steps are mostly written in the AWK programming language [94] within the Ubuntu environment. The following subsection will present a technique to obtain data from the measurement results' files in order to prepare a data set for the next process.

3.4.1 Preparing the Data

We develop a technique to obtain data from the measurement results for analysis via a UNIX shell script. The script makes use of a combination of Awk [94], an UNIX-based interpreted programming language for text processing with pattern-matching, and UNIX Bash command environment [181].

In preparing the data, as shown in Algorithm 1, we begin by identifying the folders of the measurement results regarding the complete data set. The data that are produced by the measurement are structured as follow: for each configuration, there are 100 folders for the workload that was set up to have a maximum of 100 users per second, as depicted in Figure 3.7. For each workload, there are 10 files for each measurement iteration. The median data is then calculated for each workload folder. After that, the median results

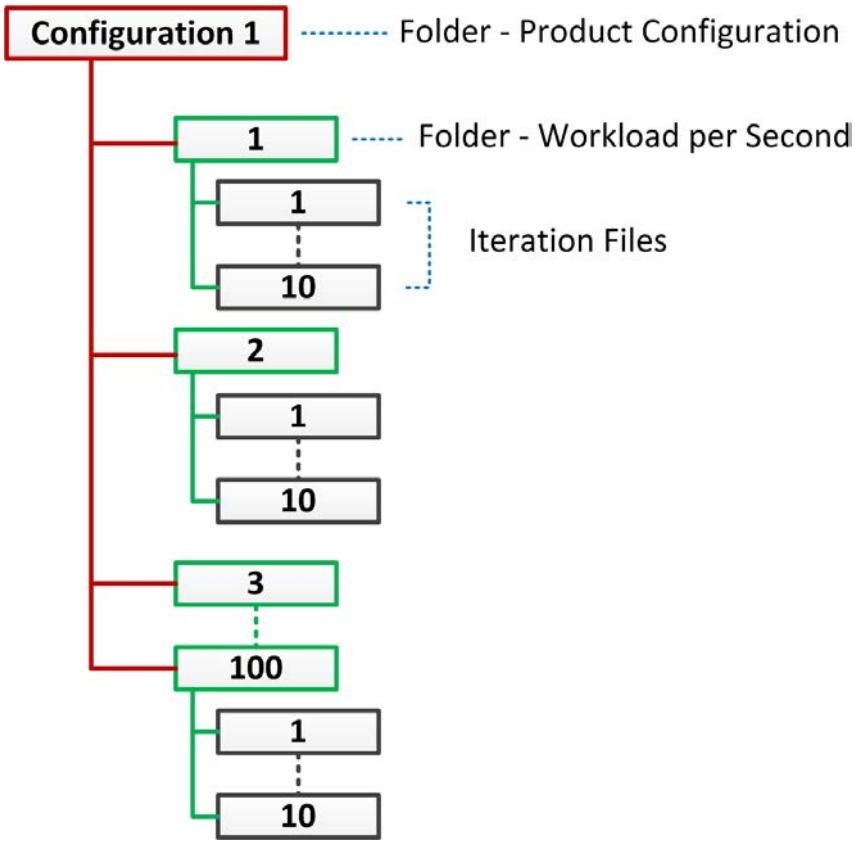


Figure 3.7: Structure of folders and files of the measurement result.

for each workload are appended into a file, which collects all the measurement results for a single scenario of a configuration.

Steps to elicit data from the measurement results are divided into four stages, as follows:

1. Scanning text position in the files and capturing lines of data that match the pattern.

2. Eliminating values in the lines of data which are not needed (this is based on the requirements of the machine learning process).

3. Transforming all the data values into a single standard. For example, we transform data from milliWatt or microWatt into Watt to cover all values of energy usage.

4. Calculating the median data from all folders using the step 3 results.

Data: Measurement Results

Result: Data Total Power Usage and CPU Power Usage

Read and List the Directory;

Create a file text;

Change the directory into the measurement directory;

while *not at end of directory* **do**

    Change the directory into the measurement directory;

    while *not at end of directory* **do**

        search Total Power Usage and CPU Power Usage;

        if *target not exist* **then**

            Skip the directory;

        else

            Captures the text result;

            Calculate the median result;

            Append the calculation result into a file;

        end

        Change directory up;

    end

    Change directory up;

end

Algorithm 1: Iteration to retrieve data from the measurement result files.

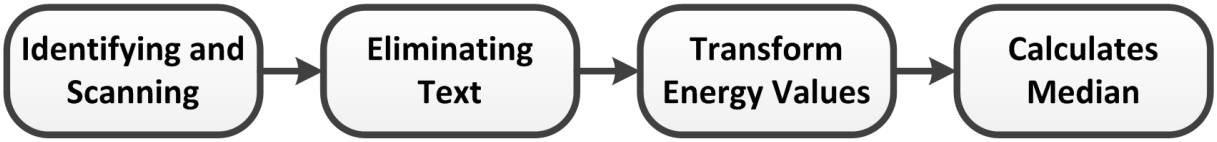


Figure 3.8: Steps to elicit data.

3.4.1.1 Obtain CPU Information

Figure 3.8 shows the steps to obtain the CPU energy values from the measurement data. We begin by identifying and scanning the whole text in the files. The text that is not needed are subsequently eliminated. The CPU energy value is then transformed to have a similar range of values, such as milliWatt into Watt. Finally, the median data are calculated. In conclusion, we present the details to obtain the CPU power information from the measurement results.

```
PowerTOP Version;v2.5
Kernel Version;Linux version 3.13.0-55-generic (buildd@kapok) (gcc version 4.8.2
 (Ubuntu 4.8.2-19ubuntu1) ) #92-Ubuntu SMP Sun Jun 14 18:32:20 UTC 2015
System Name;Oracle Corporation VirtualBox 1.2
CPU Information;1x  Intel(R) Core(TM) i5 CPU       M 520  @ 2.40GHz
OS Information;Ubuntu 14.04.2 LTS

**Power Consumption Summary**
51.8 wakeups/second, 0.0 GPU ops/second, 0.0 VFS ops/sec, 0.0 GFX wakes/sec and
7.9% CPU use

Power est.;Usage;Events/s;Category;Description
  12.2 W  ;  4.0 pkts/s;;Device;Network interface: eth0 (e1000)
  1.59 W  ;  3.7%;  1.0;Process;/usr/sbin/apache2 -k start
  818 mW  ;  1.9%;  2.0;Process;java
  430 mW  ;  1.0%;  6.0;Process;/usr/sbin/mysqld
```

Figure 3.9: Excerpt of Powertop measurement data - individual process power usage of MOODLE with workload.

First Step – captures the information of total and CPU power usage from the measurement results. Powertop generates files from an energy measurement activity, where each file consist of information about system architecture, processes and CPU power usage. As shown in Figure 3.9, in the top part of the measurement file, Powertop captures the information of processes and their energy usage within the system, such as the "Apache2" HTTP server which consumes 1.59 Watts running in a Virtualbox environment. Another part of the file, as depicted in Figure 3.10, presents the estimation of total power usage and CPU power usage per core. To obtain this information, the script captures the estimation of CPU and total power usage in a VM that matches a line text, as shown in Listing 3.1 line 7 and 8.

```
**Device Power Report**
System baseline power is estimated at 15.6 W

Power est.;Usage;Device name
          ;  4.0 pkts/s;Network interface: eth0 (e1000)
  3.38 W  ;  7.9%;CPU misc
    0 mW  ;  7.9%;CPU core
    0 mW  ;  7.9%;DRAM
          ;100.0%;PCI Device: Intel Corporation 440FX - 82441FX PMC [Natoma]
          ;100.0%;PCI Device: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
          ;100.0%;PCI Device: InnoTek Systemberatung GmbH VirtualBox Graphics Adapter
          ;100.0%;PCI Device: Intel Corporation 82540EM Gigabit Ethernet Controller
          ;100.0%;PCI Device: InnoTek Systemberatung GmbH VirtualBox Guest Service
          ;100.0%;PCI Device: Intel Corporation 82371AB/EB/MB PIIX4 ACPI
```

Figure 3.10: Part of Powertop measurement data - total energy estimation and CPU energy usages of MOODLE with a workload of ten users per second.

Listing 3.1: Script to match a line and capture the text.

```
1  #!/bin/bash
2  for file in *.csv
3  do
4   for ((i=0; i<1;i++))
5   do
6    name=${file}
7    awk '/System baseline power is estimated/{print $7""$8}' $file
8    awk '/;CPU misc/{print $1""$2""$3""$4}' $file
9    done
10 done
```

Second Step – eliminates text from the measurement result files by matching the pattern. As shown in Listing 3.2 line 3, all alphabetic characters are eliminated using a pattern match of [A–Za–z].

Listing 3.2: Script to eliminate text.

```
1  #!/bin/bash
2  awk '{gsub ( "[;%/]","" ); print $0}' cpux.txt > cpux1.txt
3  awk '{gsub ( "[A-Za-z]","" ); print $0}' cpux1.txt > cpux2.txt
```

Third Step – convert the measurement values into the same standard for the next process. While capturing the energy usage, Powertop may print different values into the file report, such as Watt, milliWatt, and microWatt. In doing so, the energy value is transformed to a single standard (Watt) to cover all measurement results. As shown in Listing 3.3 line 2 and line 3, milliwatt (mW) and microWatt (μW) are transformed into Watt (W).

Listing 3.3: Script to transform values.

```
1  #!/bin/bash
2  awk'{if ($2=="mW")$1=$1*0.001; print $0}' cpu1.txt > cpux1.txt
3  awk'{if ($2=="uW")$1=$1*0.000001; print $0}' cpux1.txt > cpuxa1.txt
```

Fourth Step – calculates the median data for our next research. As shown in Listing 3.4, the median data is calculated for each folder. When there are 100 folders for a product configuration, we will obtain 100 tuples of data.

Listing 3.4: Script to calculate median data.

```
1  awk '{for(i=1; i<=NF; i++)
2   a[i][NR]=$i}
3   END {for(i=1;i<=NF;i++) {
4   asort(a[i])
5   printf("%.1f",NR%2? a[i][(NR+1)/2]:(a[i][NR/2]+a[i][NR/2+1])/2)}
6   print ""}' cpu-ok.txt > median.txt
```

The above steps produce median data for each folder that match to the workload, which is from one to 100 users per second. It is worth noting that such steps only apply for this particular measurement technique.

3.5 A Case Study of a Three-Tier System

In this case study, we aim to report the measurement regardless of the effect of the tickless kernel, as explained in Section 2.2. All virtual environments in these measurements use KVM as the hypervisor and are set up with one core of Processor, one GB of memory and 10 GB of storage. The virtual machine images run on Ubuntu 12.10_x64. In the Cloud environment (we use the Elastichost cloud provider) the processor is the AMD Opteron Processor 6128 2GHz, and in the laptop environment, it is the Intel Core i5 M520 2.4GHz. We measure three combinations of software components that configure with different HTTP servers. In this experiment, we analyse and compare the measurement results for each configuration. It is worth noting that this experiment only measures each configuration once (no repetition).

3.5.1 Workload for Experiment

To produce a workload for the system, there is a wide range of choices, as explained in Section 3.2. Here we make use of JMeter [6]. We have conducted the measurement by producing the same load for all environments and dividing the experiment into blocks of four minutes. During the experiment, in the first four minutes we produce the load of one user per second, and the second four minutes period for two users per second, and so on. This has resulted in 240, 480, 720, 960 and 1200 users per period of 4 minutes.

As an example, Figure 3.11 plots the energy consumed by the Nginx HTTP server in the Cloud where the thick line is the average within every four minutes block of time. Furthermore, figure 3.12 (a) shows the boxplot graph that uses the same data as in Figure 3.11.

One application of our method is when a software product can run on an in-house system or Cloud. We can compare the energy consumption between them when the application runs on an in-house system such as a Laptop and Cloud, as explained in the next section.

3.5.2 Analysis of the Measurement Results

The measurement of power consumption in the Cloud computing system and virtualised environment is comparable, because both systems use similar virtualised technology, the Kernel-based Virtual Machine (KVM). The power consumption is measured based on the Advance Control and Power Interface (ACPI) that exists in the Laptop to manage the battery and in the Cloud system to turn the virtual machines on or off .

There are some spikes on each workload graph, as shown in Figure 3.11, where the increase of workload triggers the CPU to consume a higher amount of energy. We speculate two possibilities regarding these measurement results. The first, the software power meter, Powertop, has a delay in capturing the values of energy. And the second, the tickless kernel effect influences the CPU, when a workload arrives at the system under

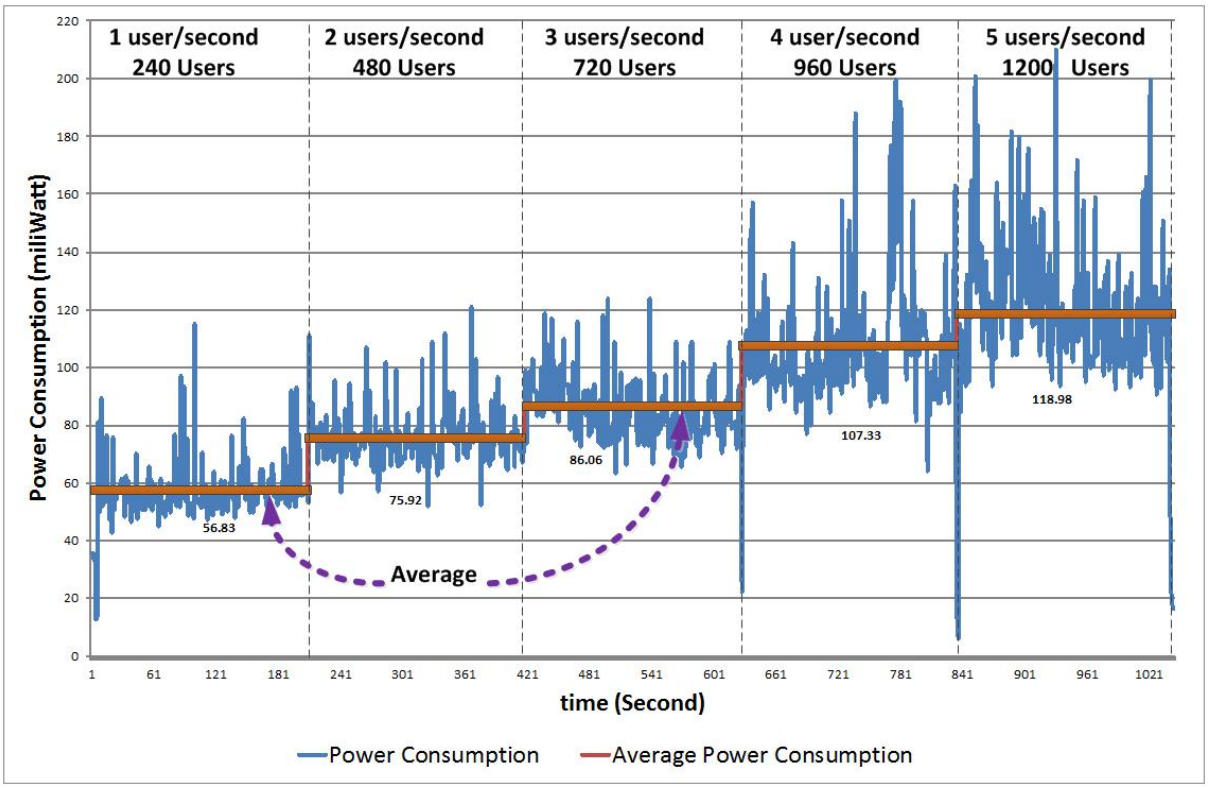


Figure 3.11: Nginx power consumption in the Cloud environment.

examination, the CPU awakes instantly, which consumes a higher amount of energy.

As depicted in Figure 3.12, when a smaller load is used, the laptop consumes less energy than the Cloud. However, when the load increases, for example at 3 users per second (720 users accessing in a 4 minutes interval), the Cloud consumes less. This trend continues as the load increases. For example, at 5 users per second (1200 users accessing the servers in a 4 minutes interval) the laptop consumes approximately 50% more energy. We speculate that the infrastructure in the Cloud is more complex than in a laptop that provides a better energy management. For example, the Cloud infrastructure is built on a shared environment that manages the CPU and memory more efficiently, compared to a laptop.

In addition, we observe that the pattern of energy consumption in the Cloud as the number of users increase becomes more stable. In contrast, in a laptop, we observe fewer spikes when the system under examination receives workloads up to two users per second. This can be seen in the outliers as depicted on Boxplot graph in Figure 3.12.

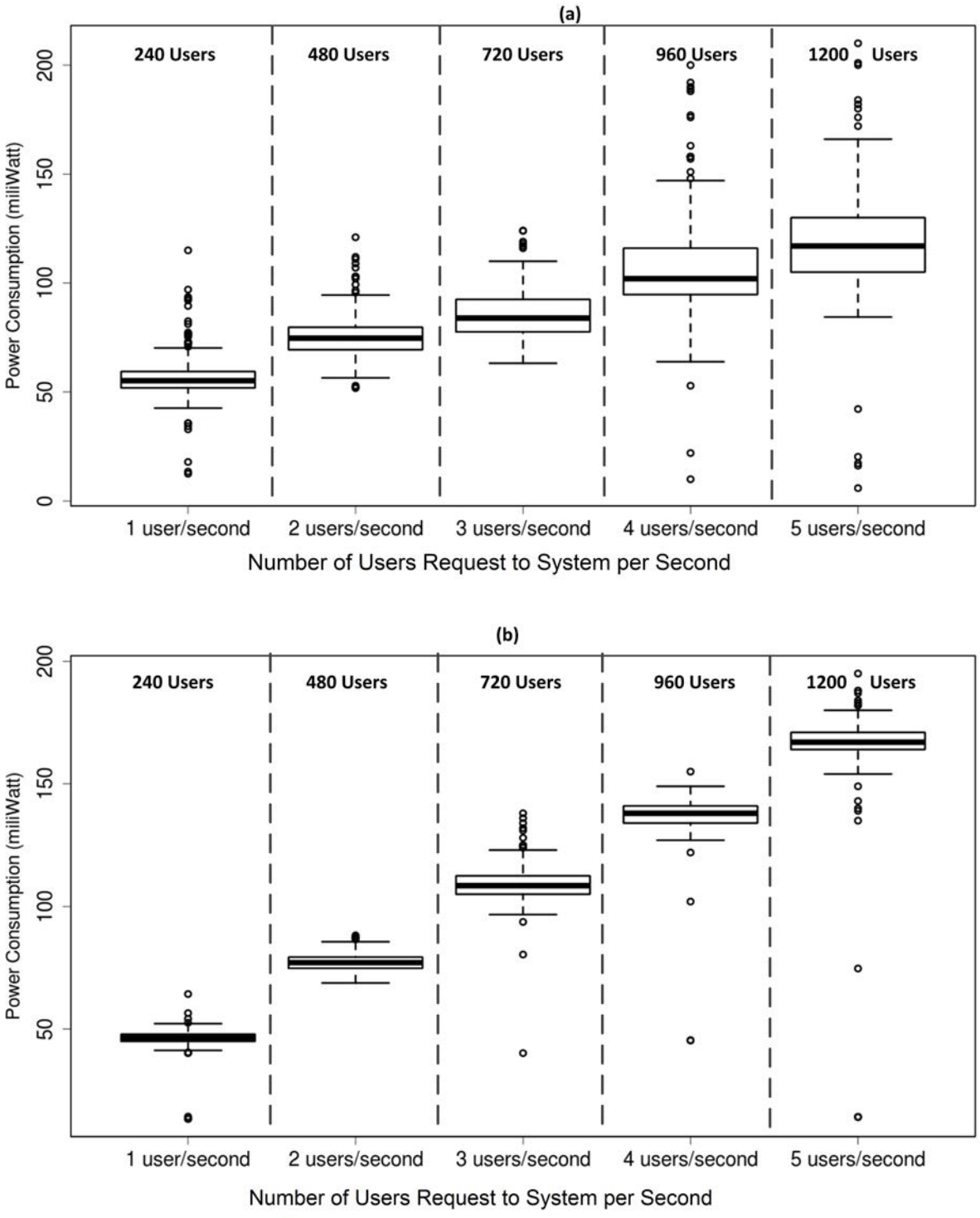


Figure 3.12: Nginx Web Server Power Consumption in two different environments - (a) Cloud System and (b) Ubuntu KVM in Laptop

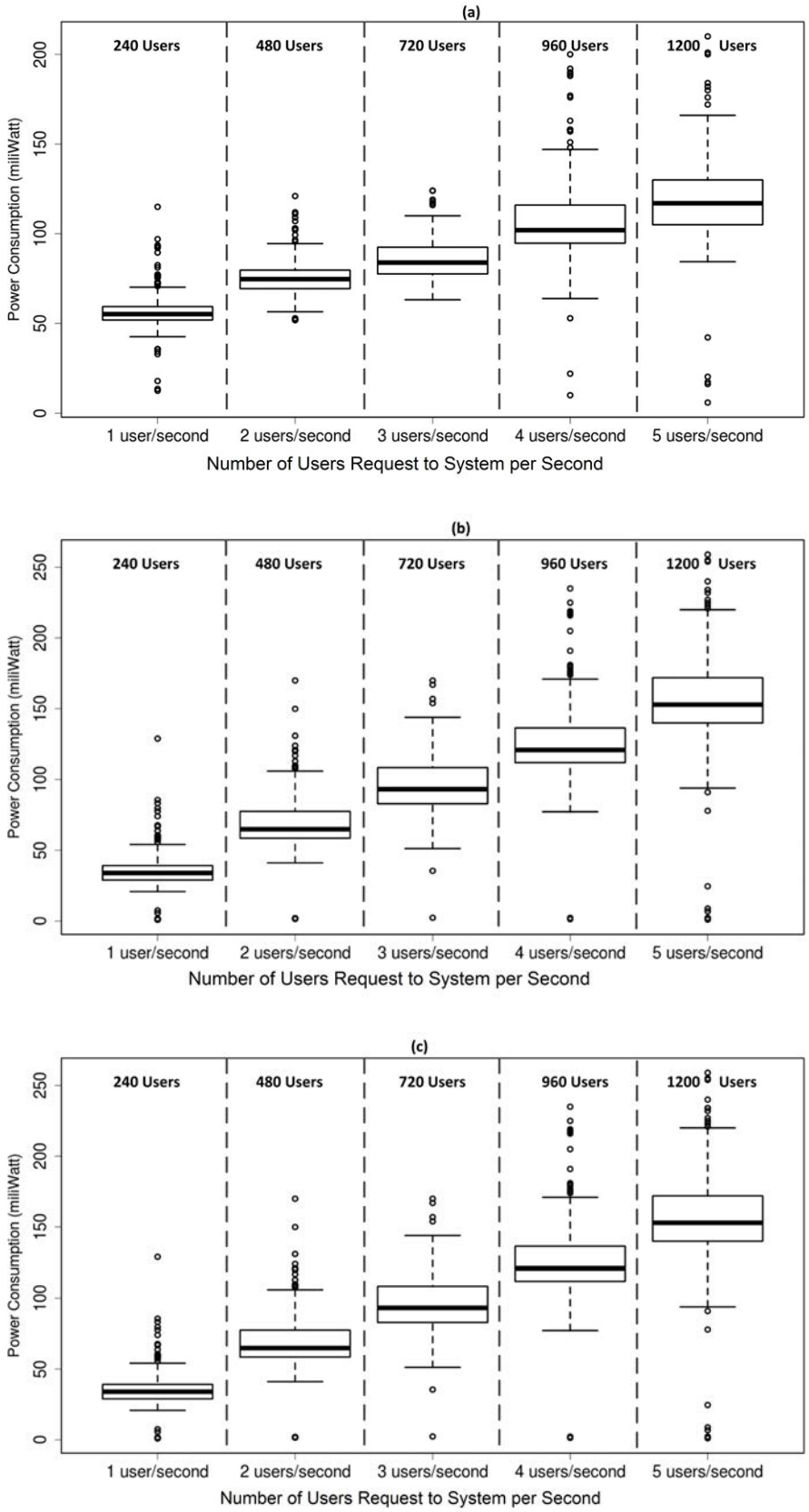


Figure 3.13: HTTP server power consumption in a Cloud system - Nginx (a), Apache (b) and Lighttpd (c).

3.5.3 Comparison of HTTP Server Energy Consumption

The configuration of the application in the Cloud system should also include the cost of operation as an important aspect. For example, the Cloud system designer must know the cost of running HTTP servers, deploying different combinations of software components, and workload behaviours from the energy consumption point of view. In this experiment, we measured the power consumption of the HTTP server that is listed in the top ten HTTP servers [51], which includes "Apache", "Nginx", and "Lighttpd", and we measured them within a Commercial Cloud system. As shown in Figure 3.13 and Table 3.1, the measurement results show us that when 1 user per second arrives at the HTTP server, "Apache" and "Lighttpd" are more efficient on average, consuming less than 50 milliWatt. However, when 5 users per second arrive at the system, "Apache" and "Lighttpd" on average consumed more than 150 milliWatt. Meanwhile, "Nginx" power consumption for the arrival of 5 users per second is 118.98 milliWatt. "Lighttpd" is the lowest in consuming power; when less than 3 users arrive at the system per second that is less than 71 milliWatt. This amount exceeds "Nginx", when three or more users arrived at the system. As shown in Figure 3.13 (a) and (c), "Nginx" and "Lighttpd" for 3 users loads have similar maximum outliers; but, "Nginx" has a smaller minimum outlier compared to "Lighttpd". It shows us that "Lighttpd" may not have a constant service when 3 users arrive at the system, as when more users arrive, we speculate, it will accumulate the unfinished service and increase the power consumption.

In the measurement within the Cloud system, the average rise of power consumption, such as the increment of power usage from one user per second to two users per second of load, tells us how the HTTP server will consume energy when more users arrive on the system. From the measurement results, the highest average of power usage increment is "Apache", which consumed on average 31,63 miliWatt, followed by "Lighttpd" which consumed 29,57 miliWatt. "Nginx" has the lowest power usage increment per user per second, 50% lower than the "Apache" power increment expenditure.

Table 3.1: Average power consumption using different loads in a Laptop and Elastichost Cloud.

OS & App.	1 User*	2 User*	3 User*	4 User*	5 User*
virtualised System in Laptop					
Apache	44	84.49	122.79	161.15	198.54
Nginx	46.09	77.25	109.22	136.70	165.14
Lighttpd	37.03	73.76	109.27	145.33	181.42
Elastichost Cloud					
Apache	48.19	84.77	109.40	136.83	174.72
Nginx	56.83	75.92	86.06	107.33	118.98
Lighttpd	36.48	70.27	96.98	127.08	154.77

** All measurement results in miliWatt*

To conclude, the same workload and combination of components in different virtualised environments consumes different amounts of energy. Furthermore, each combination of components has different behaviours when a similar workload arrives at the system under measurement. In comparing the virtualised environment energy usage, the Cloud system has better performance than a virtualised system in the laptop when dealing with a higher workload.

3.6 A Case Study to Reduce the Tickless-Kernel Effect in the Measurement of Energy

This section reports a case study that aims to reduce the tickless kernel effect [116] when measuring the consumption of energy in the virtualised environment. In this case study, we measure energy iteratively, this is because the tickless kernel moves the idle CPU into a sleeping state, which restricts the number of incoming requests [116] into a virtualised system. The iteration avoids such limitation by executing tasks continuously, so that the CPU will always be in the normal state. This measurement send a workload that increases gradually from 1–100 users/second, and repeats the process of measurement ten times, as explained in Section 3.3.1.1. For extracting the measurement results, this case study uses a technique that is described in Section 3.4, by calculating the median data for each

group of workload.

Table 3.2: List of combinations of packages for small feature model.

No	Configurations
1	nginx, mysql, php5-fpm, php5-cli
2	apache2, mysql, php5-cgi
3	apache2, mysql, php5-fpm
4	apache2, mysql, python
5	apache2, mysql, php5-fpm, pear
6	apache2, postgresql, python, lib-apache
7	apache2, postgresql, sqlite, python, python-django, lib-apache
8	apache2, tomcat7, mysql, openjdk-7
9	apache2, sqlite, php5-fpm
10	apache2, mysql, ruby, python, rails, rubygem, ruby-on-rails
11	nginx, postgresql, python, python-django
12	apache2, mysql, php5

This case study uses Virtualbox [219] as a hypervisor and makes use of a virtual manager tool, Vagrant [56], to automate the measurement process. The target of measurement consists of 12 combinations of software components, as depicted in Table 3.2, that are deployed in distinct virtual machines, running on top of the Ubuntu server operating system, where the source of application packages are retrieved from the Ubuntu package repository, as explained in Chapter 4. In this measurement of energy, the hardware environment uses a Laptop with Intel Core i5 M520 2.4Ghz. The host machine runs on the Ubuntu 12.10_x64 operating system and each virtual machine instance is setup with Ubuntu 14.04_x64 and is configured with 1 core CPU, 1 GB memory and 10 GB of storage.

3.6.1 Workload for Experiment

This case study uses JMeter [6] to produce a workload for the system under measurement that increases from 1 to 100 users per second, consecutively, aiming to create a smooth increase of workload, in order to execute tasks that produce a smooth increment of energy usage within the target of measurement.

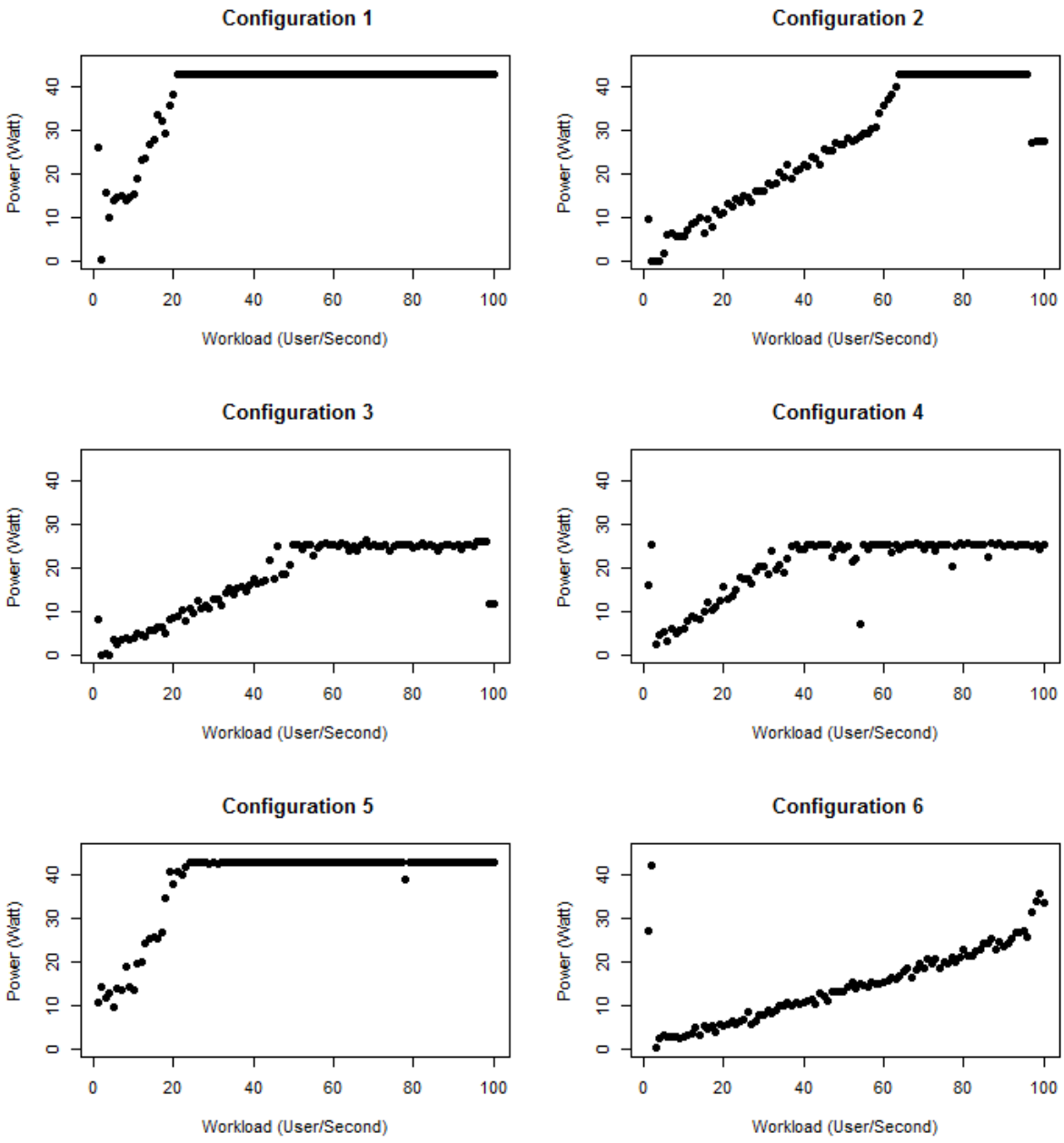


Figure 3.14: Measurement result for configuration 1 – 6.

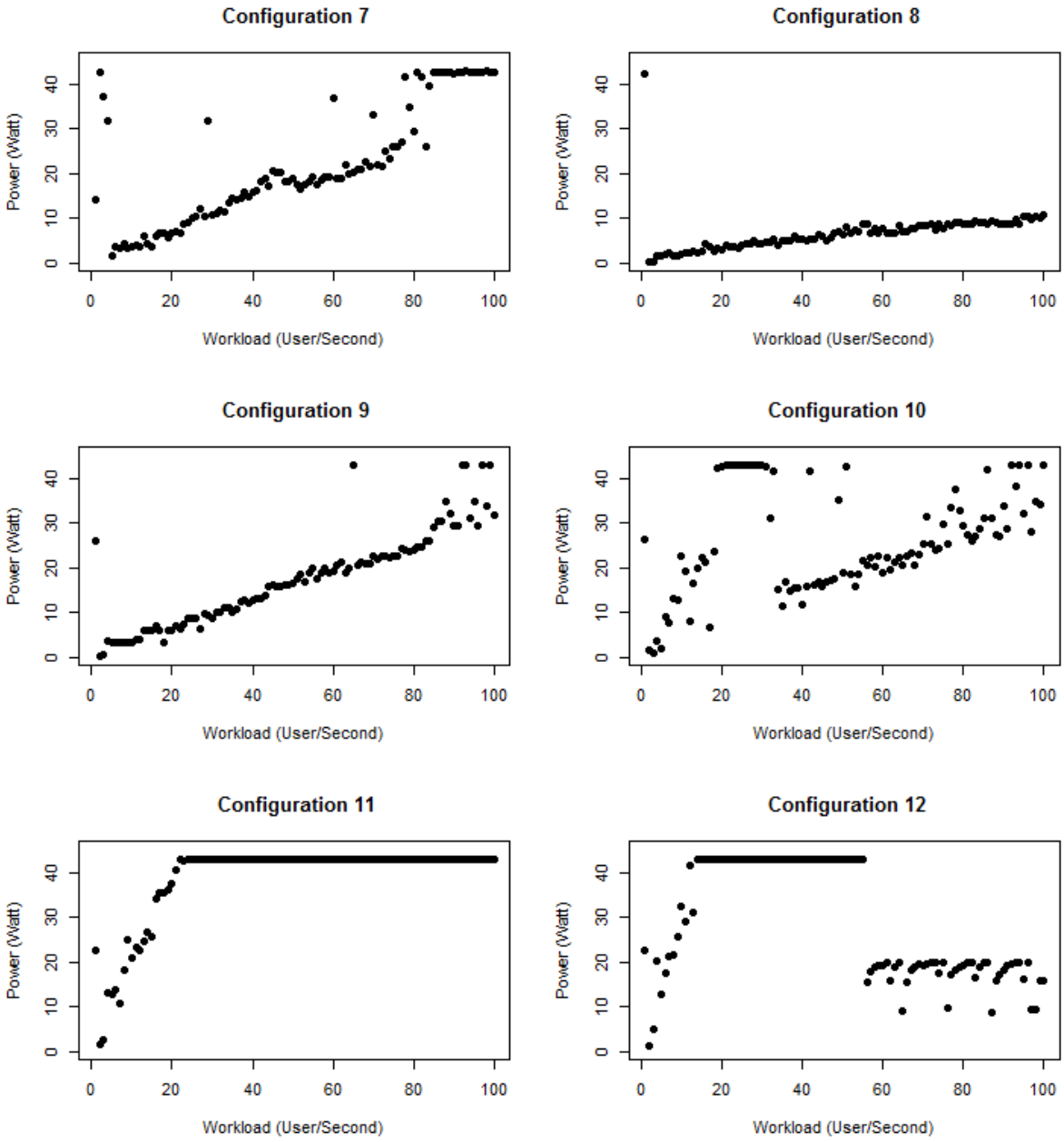


Figure 3.15: Measurement result for configuration 7 – 12.

3.6.2 Analysis of the Measurement Result

In this case study, the measurement of energy in the different combinations of software components are comparable, because all the systems are running on the same hypervisor and operating system. The energy consumption is measured based on the electric current drain from the Laptop battery that is captured by the Advance Control and Power Interface (ACPI) [152].

As shown in Figure 3.14 and 3.15, when a lower workload arrives, the configurations 2, 3, 5, 8 and 9, mostly have apache2 and mysql in their configurations, have fewer fluctuations than others. Configuration 3 (configured with apache2, mysql and php5-fpm) and 4 (configured with apache2, mysql and python) have their maximum cpu power consumption less than 30 Watt. We speculate that the CPU activities for both configuration 3 and 4 do not increase, when the workloads are more than 30 users per second, as the demand of processes does not increase significantly. Furthermore, when the workload reaches more than 20 users per second, configurations 1, 2, 5, 6, 8, 9 and 11 have lower fluctuations than other configurations. Whereas for configurations 1, 5 and 11, these fluctuations exist because of the measurement has reached the maximum power the system can handle. However, the configurations 3, 4, 7, 10 and 12 produce fluctuations in their measurement results, since the workload arrives at the system under measurement, where configuration 10, that has apache2, mysql, ruby, rails and python as its components, and configuration 12, that has apache2, mysql and php5, have more fluctuations than other configurations. In configuration 10, the CPU can handle the workload of less than 40 users per second, and configuration 12 of less than 60 users per second. We speculate that configurations 10 and 12 execute tasks intensively, which call the operating system using a system call - in a random pattern that is different to other configurations. Furthermore, the CPU may also have its maximum capacity to handle the demand of processes. As a consequence, some running processes are terminated by the operating system in order to avoid overload within the running system. However, some running processes in configuration 10 and 12,

as shown in Figure 3.15, still try to continue their execution that shows a similar pattern as depicted in configuration 12, when more than 50 users per second arrive at the system under examination.

In this case study, we observe that the combination of software components that include apache2 and python (when the measurement does not reach the maximum capacity of the system) produce more fluctuations than other configurations. Overall, configuration 6 (configured with apache2, postgresql, python, lib-apache) and 8 (configured with apache2, tomcat7, mysql, openjdk-7) outperform all other configurations and act differently as their results are almost linear. Moreover, most of the measurement results in figure 3.14 and 3.15 show the fluctuations when the first workload arrived at the system. We speculate that the first arrival of the workload triggers the CPU to force its maximum capacity to handle the requests.

To conclude, by sending a repeated workload, we can reduce the tickless kernel effect on the measurement of energy significantly. In this case study, from 12 configurations, only configuration 10 (configured with apache2, mysql, ruby, python, rails, rubygem, ruby-on-rails) and configuration 12 (configured with apache2, mysql, php5) produce almost non-linear graphs. This result supports Suresh et al. [116] that in a virtualised environment the idle CPU will limit the number of possible incoming requests. When compared with the results in Sections 3.4 and 3.5, especially figures 3.11, 3.14 (configuration 1) and 3.15 (configuration 11) that use nginx as a HTTP server, the measurement of energy in the virtualised environment with the repeated workload outperforms other measurements, in terms of the linearity of the results. This case study demonstrates that our approach to measuring energy is not restricted by the tickless kernel effect, and produces results of the consumption of energy within a different combination of software components in a virtualised environment.

3.7 Chapter Summary

Chapter 3 presents a technique to measure energy with a workload within the virtualised environment. In particular, this chapter introduces a technique to generate a workload, as explained in Section 3.2, in order to execute tasks within the target of measurement. This chapter has also presented techniques to measure energy in two different virtualised environments; in a Cloud and in a virtualised system in a laptop. Section 3.4 presented steps to prepare a data set for the machine learning process that includes filtering out the measurement results to obtain the data needed. Sections 3.5 and 3.6 evaluate our measurement technique to measure energy usage in the virtualised environments that focuses on the tickless kernel effect. The results of this experiment show that our measurement technique is suitable for measuring energy in the virtualised system.

CHAPTER 4

AUTOMATIC CREATION OF FEATURE MODELS FROM A LARGE SOFTWARE REPOSITORY: A CASE STUDY

Chapter 3 described how to measure the consumption of energy from combinations of software components in virtualised environments. To apply a Software Product Line, we need a feature model to represent all the products [121, 179, 54]. For a small system such as a web based system [80], we can build the feature model manually by exploring all of the dependencies between components. However, for a large system there are a large number of component dependencies, for which it is infeasible to use the same technique as for the small system.

This chapter presents a technique to automate the creation of a feature model for a Software Product Line (SPL) development from a software repository, with the help of graph approaches. The aim of using the graph approach is to reduce the complexity, which emerges as a result of package dependencies such as when we merge two or more configurations from sets of features. Furthermore, this technique also reduces the steps and provides more flexibility compared with the previous technique [81] to create constraints for a feature model. Such a technique allows us to resize the composition, such as including only relevant packages of a feature model automatically. In this chapter, we also present the graph applications to merge and modify the feature models that are retrieved from a software repository.

Section 4.1 provides a description of our approach to retrieving and to building a feature model from a software repository. Section 4.2 presents the software repository metadata as the main source of obtaining the information about dependencies between components. Section 4.3 offers a technique to retrieve package dependency from the Ubuntu package repository. Section 4.4 introduces the steps required to extract the package repository into a feature model, which covers package dependency and constraints. Section 4.5 suggests a technique, which uses a running example, to transform the package dependency graph into a constraint structure for a feature model.

4.1 Description of Our Approach

This section provides a description of our approach to create a feature model from the software repository that requires several components so as to achieve its goals. Firstly, there are metadata within a software repository to describe the contextual condition of the sources. Secondly, the repository has transparent information and its content is accessible for viewing. Thirdly, each package can be retrieved as a single component.

Most large software repositories use metadata as their information library to provide a description of what the existing resources are and how they can be useful for a deployment [178, 42, 10, 211]. This metadata mostly consists of source name, description, working environment, file size and version. Furthermore, the metadata for a software repository is slightly similar to a file property, where there is some description of component sources and its size

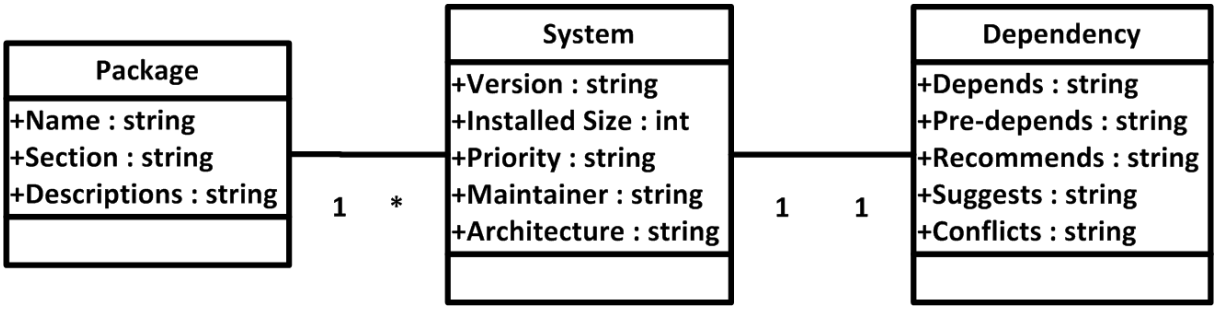


Figure 4.1: Excerpt Metadata of Ubuntu package repository.

In large software repositories, such as Git [42], Ubuntu package repository [211] and SVN [178], metadata are utilised to manage the information of files associated with software components. In particular, each software repository may have a different structure of metadata. For example, the Debian/Ubuntu package repository uses a reference file to describe information about dependencies between components, versions and system environments. The metadata of a package in a Ubuntu package repository, as shown in Figure 4.1, can have more than one version, where each version has its own package dependency. By contrast, Git uses a database to provide a track of a committed project as a snapshot of the project over time.

In this work, we can only handle a Debian/Ubuntu package repository, where the retrieval result is in a text graph file format, which is in *.dot* Graphviz [68] format. Our graph result can be transformed into different graph formats such as GraphML [202] and pajek [15], using graph tools, e.g. Gephi [14] and Igraph [53].

4.2 Metadata in Software Repository

This section describes the metadata in a software repository in the interest of creating a feature model from a repository. A metadata within a software repository consists of the name of the component, version, file size and dependency, as shown in Table 4.1. A typical component in a repository may have more than one version and a different size of files. Furthermore, a dependency presents something that is required when installing a component in order to have a system run accordingly. For example, the package apache2-bin requires libpcre3 to run a Perl script in an Ubuntu system.

The information about dependency varies between software repositories. For example, Debian and Ubuntu make use of a script file as the metadata, and categorise the dependency between components as 'suggest', 'depends', 'recommends' and 'conflicts', as explained in Section 2.4.2. In the RPM repository, the metadata is in an XML file that is linked to the information about the components, such as 'version', 'release' and 'de-

Table 4.1: Excerpt of metadata in several software repositories.

Repository	Name	Version	Dependency	Category	Size	Description
Debian [55]	yes	yes	yes	yes	yes	yes
Ubuntu [211]	yes	yes	yes	yes	yes	yes
FreeBSD [79]	yes	yes	yes	yes	yes	yes
RPM [169]	yes	yes	yes	yes	yes	yes
Git [89]	yes	yes	yes		yes	yes
eCos [65]	yes	yes	yes	yes	yes	yes

pendency', and list of the files in the repository. Slightly similar to RPM, the FreeBSD metadata is in the form of two files that are compact-manifest and manifest to divide the information about components and its file locations. The eCOS repository, which is an embedded Linux-based system, manages the dependency within the configuration of a pre-compile code that a required component should be listed in the configuration code.

We believe, based on the metadata, one can build a graph that represents the dependencies between components, such as require or exclude, in order to build a feature model from a software repository. For example, we can build the feature dependency for a feature model using metadata by creating a graph that traces the dependencies between components in the software repository. One of the challenges of creating the graph dependencies from the software repository is that each repository has its own rules which may have differences from each another. In doing so, we need further exploration in order to get a graph's dependencies.

In the subsequent section, we will present a technique to extract component dependencies from a software repository in creating a feature model.

4.3 Retrieval of Package Dependency into a Graph Model

This section presents a technique to retrieve package dependencies from the software repository in order to build a graph model. This model is in the form of a directed acyclic

graph, which provides information about related components that are included into a set of component configurations.

In this work, we use the Ubuntu package repository as the source of our activity. As explained in Section 2.4.1, the Ubuntu package repository provides information on package dependencies in their file description. The Ubuntu package dependencies can be retrieved and built into a directed graph. The graph model enables us to analyse and check the package dependencies, such as which packages influence configuration and the conflicts existing between packages. Another benefit of having a package dependency as a graph model is that we can compare and reproduce the package configuration of different release versions. For example, we can analyse why a configuration for the package Wordpress on release Ubuntu 14.04 has better performance than on release 15.04. In addition, we can analyse the evolution of a package such as Wordpress through its common and variant packages.

A graph model of the retrieval result is represented as .dot files that serve as input for *DOT* language package of the Graphviz [68]. For example, a result of the debtree – a Debian/Ubuntu utility to generate a graph from a software repository –, as shown in List 4.1, can be customised to reduce or to enlarge the graph size. One of the techniques to resize the graph model is to limit the depth of the levels of the package dependencies. This technique reduces the dependency by removing the branch of package dependency where the choice of branch to be removed is based on the commonality of the components (the packages that are used by most of the configuration, such as library components). It is worth noting that this technique aims to create a feature model, and it is not remove the software components. For example, "wordpress" -> "libjs-scriptaculous", "wordpress" depends to "libjs-scriptaculous", where the symbol " -> " means the left part has a dependency to the right part. Then, "libjs-scriptaculous" ->"libjs-prototype", "libjs-scriptaculous" depends on "libjs-prototype". To limit the size of a graph model, the depth of dependency is reduced, such as removing the "libjs-scriptaculous" dependency to the "libjs-prototype".

Listing 4.1: An excerpt dot file of package Wordpress

```
digraph "wordpress" {
        rankdir=LR;
        node [shape=box];
        "wordpress" -> "libjs-cropper" [color=blue];
        "libjs-cropper" -> "libjs-scriptaculous" [color=blue];
        "wordpress" -> "libjs-prototype" [color=blue];
        "wordpress" -> "libjs-scriptaculous" [color=blue];
        "libjs-scriptaculous" -> "libjs-prototype" [color=blue];
        "wordpress" -> "libphp-phpmailer" [color=blue];
        "libphp-phpmailer" -> "php5" [color=blue];
        "wordpress" -> "libphp-snoopy" [color=blue];
        "libphp-snoopy" -> "php5" [color=blue];
        "wordpress" -> "tinymce" [color=blue];
        "wordpress" -> "apache2" [color=blue];
        "wordpress" -> "mysql-client" [color=blue];
        "wordpress" -> "libapache2-mod-php5" [color=blue];
}
```

In a Software Product Line development, combinations of features are merged and customised to produce new configurations, where commonalities and variabilities make their effort. We merged packages to model product configurations for a feature model development. For example, to build a Blog system, we combined packages of HTTP server, Database management system and Web application. However, some redundancy may exist from the merging process. In doing so, we can eliminate duplicate edges of package dependencies using Proximity Stress Model (PRISM) [83].

4.4 Proposed Method for Automatic Extraction of the Feature Model

This section presents steps to create a feature model from a software repository. We believe this technique has the potential to be used for different repositories that require certain conditions, such as a metadata with information of a component's name, version and dependency – such as require and exclude –, and the file of the component itself. Figure 4.2 shows an approach to creating a feature model from a software repository where the metadata provides information to build the graph of components dependencies. This graph then transforms into a feature model using a technique that will be explained in

the next subsection.

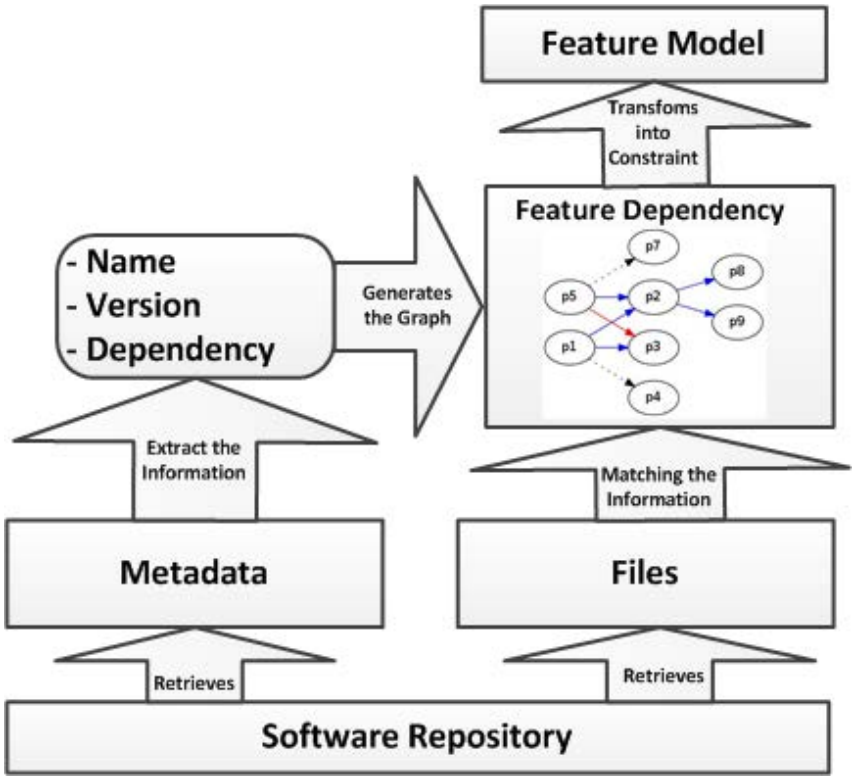


Figure 4.2: Excerpt of build a feature model from a software repository.

In a feature model, the cross-tree relationship represents the feature dependencies and the tree hierarchy represents the parent-child features, as explained in Section 2.4.2. In a large software repository, such as the Ubuntu/Debian package, the repository consists of thousands or more of packages. The package dependencies are described in the control script file. As a result, when a modification occurs within a set of packages, this will influence their dependencies. Furthermore, these steps include a technique to resize the generated feature model using an Ubuntu utility, and transform the graph results into constraints of a feature model.

The package within a repository evolves with time because of product development. Therefore, after creating a feature model from a repository, if the package repository evolves, we need to check whether the feature model is still consistent with the new version of the repository, i.e., if it is still valid. The process to check consistency observed in previous work [3] is complex, possibly requiring a change in the behaviour of the tools,

in order to check consistency when there is a change in the package repository. In addition, we can adjust the size of the feature model generated from the repository by adding some constraints. For example, we can exclude the conflict packages by adding this option in a script that runs by the Debtree, before retrieving the package dependency from a Ubuntu package repository. This option of constraints is based on the Ubuntu package rules, as explained in Section 2.4.2. This is useful because it allows management of the size of the feature model to include only relevant packages. Previous work [81, 58] did not consider this issue.

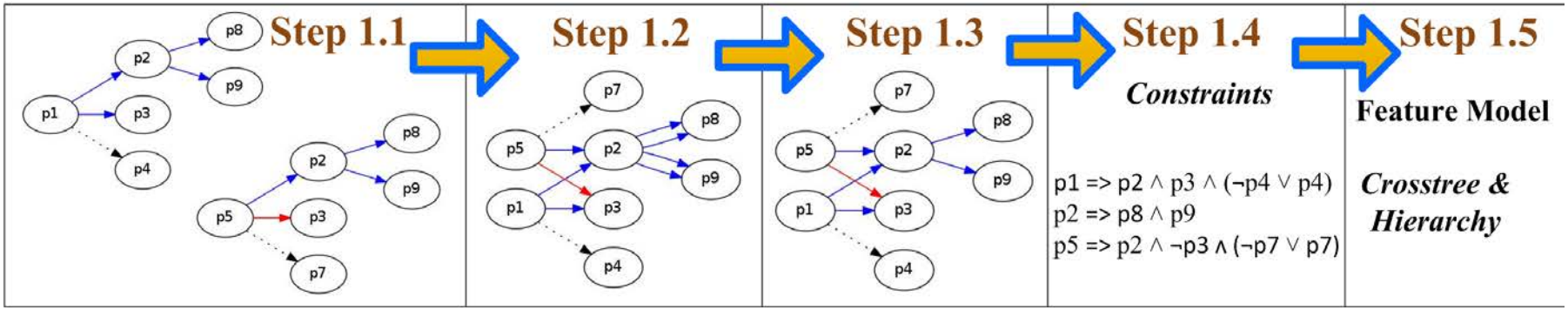


Figure 4.3: Steps 1–5 to build the feature model from the Debian package repository. In Step 4, constraints are in the form of the Propositional Logic

4.4.1 Extracting Package Metadata from a Repository to a Feature Model

This subsection explains our method to find the variability from the Ubuntu/Debian package repository and to build a feature model, as depicted in Figure 5.5. Our method consists of the following steps:

Step 1 – Retrieving Data: In this step, we aim to create a graph that shows the package dependencies which are retrieved from the Ubuntu package repository. We make use of the Debian utility, *debtree* [1], to extract the information from the package file description. The result of this retrieval activity is transformed into a directed acyclic graph. The edges of this graph have different colours to distinguish rule dependencies, as explained in Section 2.4.2. This graph is saved in a *DOT* Graphviz format [68], which is a text file structure.

Step 2 – Merging Configuration: We merge a given set of packages to configure a new software product. This step creates a graph containing all the required packages and their dependencies, which merge two or more sets of graphs from Step 1. For example, to build a feature model of web services based on Debian Linux, we merge all the packages required for web services development, such as Web Servers and Database Management systems. We use the Proximity Stress Model (PRISM) [83] to perform the merging to avoid overlaps, while retaining the structural information inherent in a layout using little additions area. The result from this activity is a combination of graphs that merge similar nodes (packages), but may have redundancy on their edges.

Step 3 – Removing Duplicated Edges: This step aims to remove the redundant edges of Step 2. We remove the duplicate edges to have a unique relationship between packages. In this step, each line of graph syntax is compared one-to-another to find any duplicate statements. When one is found, the second statement is eliminated from the file. Moreover, by converting the graph format from the results of this step into Pajek [15] and GraphML [202], we gain more information of its nature from the graph, such as

its diameter, whether it is edge-in or -out and its isomorphism is analysed using the graph tool, Igraph [53]. We plan to extend the results of this step using a combination of graph and machine learning approaches as part of our future work.

Step 4 – Creating Constraint Rules: Steps 1–3 will lead to a directed acyclic graph with merged nodes and edges. Each node of this graph corresponds to a leaf in the feature model, whereas the edges represent the features' dependencies. Therefore, this graph is used to create the constraints describing the cross-tree relationship of the feature model. Further detail of this step is explained in Section 4.5.

Step 5 – Finalising the Feature Model: This step constructs the hierarchy and cross-tree relationship of a feature model. The hierarchy relationship builds a tree of parent-child from the categories of all the packages that are retrieved from the repository. The cross-tree relationship is taken from the results of Step 4. The combination of the hierarchy and the constraints forms the feature model.

After the feature model is created, we can use FeatureIDE [206] to visualise the feature model. We can also analyse the feature model to obtain several practically important pieces of information, such as the key features and the complexity of the dependencies. Section 4.4.2 explain how to acquire this information.

4.4.2 Key Features and Complexity of Feature Dependency

We can analyse whether a feature is a *a key feature* in the model. For example, if we are creating a feature model of a web service, the web server possibly has more dependencies than others, i.e., it may be a key feature. Our approach can be used to identify key features based on the number of edges-in and -out from its corresponding node in the graph representing the cross-tree relationship. It is also possible to measure the graph diameter to ascertain the complexity of the feature dependency. Based on this information, one may wish, for instance, to filter out some packages from the feature model.

The information about key features may reduce the cost of analysis in an SPL development. It gives insight into which features will influence the product configuration

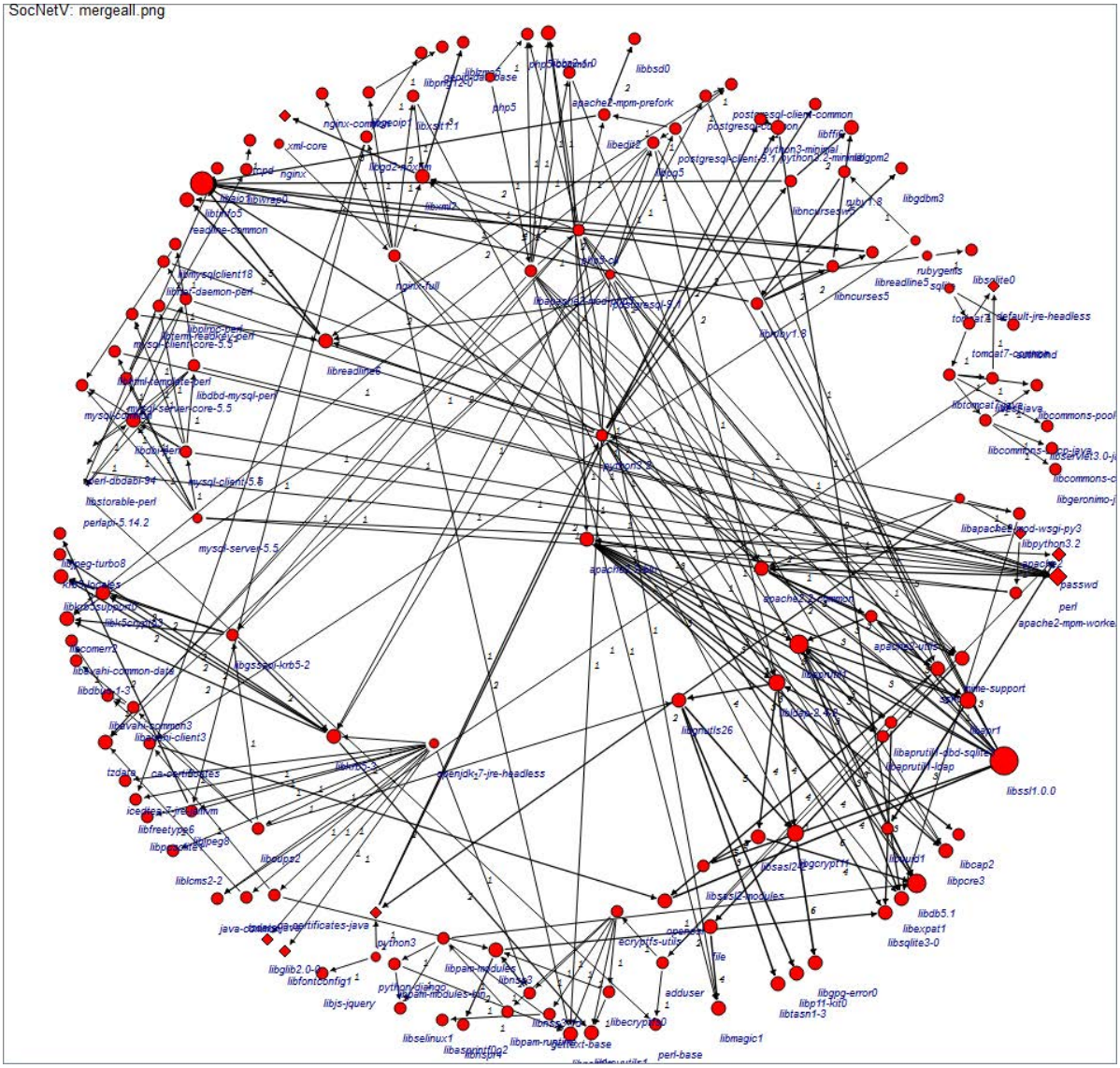


Figure 4.4: Merge of packages - Edge-In [The bigger the object, the higher number of packages depend on it]. This graph is generated using SocNetV [120]

process as *Pre-Configuration* knowledge. Furthermore, using graph approaches, such as edge-in or -out, the key features can be observed at the early stages of development. As shown in Figure 4.4, package libssl1.0.0, which is a package to serve the security of the online system as the Secure Socket Layer (SSL), has a bigger size than any node in the graph (node in red) and many edges (line with arrow and in black) pointing in to this package, which reveals that this node is required by many packages in order to have a valid software configuration for the Ubuntu system.

4.4.3 Comparing a Feature Model and the Current Information within the Ubuntu Package Repository

In the Debian/Ubuntu package repository, packages evolve to improve their performance or to remove bugs within the time line. Since a feature model is created from the repository, it will only receive the information of the latest updates. As a result, when there is a change in their dependencies, because of a development result, the next feature model becomes invalid.

Checking the consistency between a given feature model and the newest version of the package repository is straightforward when using our method. We can generate a new cross-tree relationship automatically, based on the latest version of the repository and compare it to the cross-tree relationship of the existing feature model. This can be done based on a textual comparison between the constraints from the existing feature model and the new one extracted from the repository.

As an example, as shown in Listing 4.2 line 2 and Listing 4.3 line 2, "php5-cgi" package installed for "wordpress" in Ubuntu 12.02 and 14.04 has differences on "libdb5.1" and "libdb5.3". A simple linux command such as *diff* can be used as a consistency check. The diff command compares the old and new text of graph files from the repository retrieval results, as depicted in Listing 4.4 line 2. Moreover, the graph comparison check also supports the consistency check with more cost on building the graph.

Listing 4.2: Ubuntu 12.04 – Precise.

```
1  "alt7":"php5-cgi" -> "libbz2-1.0" [color=blue];
2  "alt7":"php5-cgi" -> "libdb5.1" [color=blue];
3  "alt7":"php5-cgi" -> "libpcre3" [color=blue];
4  "alt7":"php5-cgi" -> "libssl1.0.0" [color=blue];
5  "alt7":"php5-cgi" -> "libxml2" [color=blue];
6  "alt7":"php5-cgi" -> "mime-support" [color=blue];
7  "alt7":"php5-cgi" -> "php5-common" [color=blue];
8  "alt7":"php5-cgi" -> "libmagic1" [color=blue];
9  "alt7":"php5-cgi" -> "ucf" [color=blue];
10 "alt7":"php5-cgi" -> "tzdata" [color=blue];
```

Listing 4.3: Ubuntu 14.04 – Trusty.

```
1  "alt7":"php5-cgi" -> "libbz2-1.0" [color=blue];
2  "alt7":"php5-cgi" -> "libdb5.3" [color=blue];
3  "alt7":"php5-cgi" -> "libpcre3" [color=blue];
4  "alt7":"php5-cgi" -> "libssl1.0.0" [color=blue];
5  "alt7":"php5-cgi" -> "libxml2" [color=blue];
6  "alt7":"php5-cgi" -> "mime-support" [color=blue];
7  "alt7":"php5-cgi" -> "php5-common" [color=blue];
8  "alt7":"php5-cgi" -> "libmagic1" [color=blue];
9  "alt7":"php5-cgi" -> "ucf" [color=blue];
10 "alt7":"php5-cgi" -> "tzdata" [color=blue];
```

Listing 4.4: Diff results.

```
1 diff trusty.txt precise.txt
2 2c2
3 < "alt7":"php5-cgi" -> "libdb5.3" [color=blue];
4 ---
5 > "alt7":"php5-cgi" -> "libdb5.1" [color=blue];
```

Using the above technique, the consistency between releases of the Debian/Ubuntu package dependencies and a feature model can be checked to find any changes. Consistency checks for another large software repository will need further research.

Next, we present a technique to transform the packages' dependencies that are retrieved from the Ubuntu package repository into a cross-tree relationship (constraints of a feature model), which adopts the Debian/Ubuntu package rules [106], as explained in Section 2.4.2.

4.5 Transforming Package Dependency into a Cross-Tree Relationship

This section presents a method to transform the package dependency from the Debian/Ubuntu package repository to a cross-tree relationship in a feature model. The package dependencies are transformed from a graph format file into a Propositional Logic format, that is suitable as input for the constraint of the feature model tool [206]. The transformation uses the results of the package retrieval from the Ubuntu package repository. Thus, it is restricted a human intervention transformation.

The retrieval results, as explained in section 4.4, produce a graph file with a format

based on *DOT* Graphviz graph text file. When transforming the package graph file, we can observe that the relationship between packages has a variety of structures. So, each type of dependency should have an equivalent symbol, whereas the colours of edges provide an insight into the relationship between features.

We transform the package dependencies from the debtree results into constraints as follows,

Pre-Depends Rule

The graph below represents the rule that the wordpress package should have package dpkg previously installed in order to be installed. If package dpkg is not installed beforehand, then package wordpress will be uninstallable. It is worth noticing that this rule must be presented at the early line of script of the constraint.

The Graphviz DOT format for this graph is:

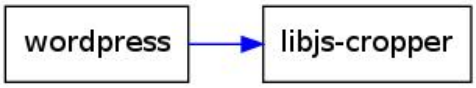
```
1  "wordpress"->"dpkg"[color=purple,style=bold];
```

The corresponding constraint in propositional logic is:

wordpress $\Rightarrow$ dpkg

Depends Rule

The graph below represents the rule that package Wordpress must include libjs-cropper to have a valid configuration.



The Graphviz DOT format for this graph is:

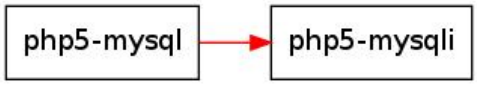
```
1  "wordpress"->"libjs-cropper"[color=blue];
```

The corresponding constraint in propositional logic is:

wordpress $\Rightarrow$ libjs-cropper

Conflicts Rule

The graph below represents the rule that package php5-mysql will conflict with package php5-mysqli.



The Graphviz DOT format for this graph is:

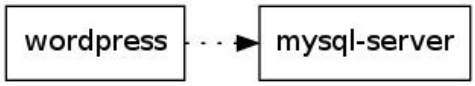
```
1  "php5-mysql"->"php5-mysqli"[color=red];
```

The corresponding constraint in propositional logic is:

php5-mysql $\Rightarrow$ $\neg$php5-mysqli

Suggests Rule

The graph below represents the rule that wordpress is suggested to include mysql-server. Including mysql-server may enhance the usefulness of wordpress; however, not including mysql-server would not cause any problem.



The Graphviz DOT format for this graph is:

```
1  "wordpress"->"mysql-server"[style=dotty];
```

The corresponding constraint in propositional logic is:

wordpress $\Rightarrow$ ($\neg$mysql-server $\vee$ mysql-server)

Recommends Rule

The graph below represents the rule that wordpress may include wordpress-l10n to satisfy the installation. For example, when dealing with a Chinese language application, a Chinese language support package is recommended to be installed. If it is not installed, wordpress would still work, but would not satisfy the Chinese language user requirement.

The Graphviz DOT format for this graph is:

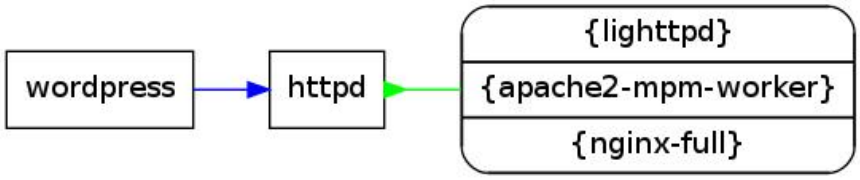
```
1  "wordpress"->"wordpress-l10n"
```

The corresponding constraint in propositional logic is:

wordpress $\Rightarrow$ $\neg$wordpress-l10n $\vee$ wordpress-l10n

Provides Rule

The graph below represents the rule that the wordpress package depends on a virtual package[1] httpd, which has three options of packages.



The Graphviz DOT format for this graph is:

```
1  "wordpress" -> "alt1" [color=blue];
2  "alt1":"httpd" -> virt1 [dir=back,
3  arrowtail=inv,color=green];
4  alt1 [
5          shape = "record"
6          label = "<httpd> httpd"
7           ]
8  virt1 [ shape = "record"
9           style = "rounded"
10          label = "<lighttpd> \{lighttpd\} |
11          <apache2-mpm-worker> \{apache2-mpm-worker\}
12          | <nginx-full> \{nginx-full\}" ]
```

The corresponding constraint in propositional logic is:

wordpress $\Rightarrow$ httpd

httpd $\Rightarrow$ (lighttpd $\vee$ apache2-mpm-worker $\vee$ nginx-full)

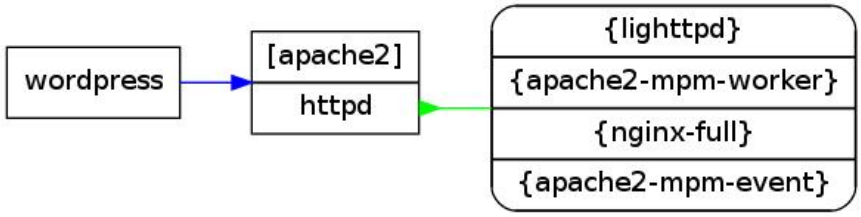
It is worth noting that the order of the propositional logic statements is important, as former statements affect later statements.

Extended 1

In the Debian package repository, a virtual package may depend on another virtual package. In the example below, a package wordpress has options to include apache2 or httpd,

[1] A virtual package represents a group of packages with similar functionality.

where httpd is a virtual package with options of some other packages.



Graphviz DOT format for this graph is:

```
"wordpress" -> "alt1" [color=blue];
     "alt1":"httpd" -> virt1 [dir=back,
      arrowtail=inv,color=green];
alt1 [
       shape = "record"
       label = "<apache2> [apache2] | <httpd> httpd" ]
virt1 [
       shape = "record"
       style = "rounded"
       label = "<lighttpd> \{lighttpd\} |
       <apache2-mpm-worker> \{apache2-mpm-worker\} |
       <nginx-full> \{nginx-full\} |
       <apache2-mpm-event> \{apache2-mpm-event\}" ]
```

The corresponding constraint in propositional logic is:

wordpress $\Rightarrow$ apache2 $\vee$ httpd

httpd $\Rightarrow$ (apache2-mpm-prefork $\vee$ apache2-mpm-itk $\vee$ lighttpd $\vee$ nginx-full)

4.5.1 An Example of Transformation from Package Dependency to Constraint

An example of a transformation from package dependencies to a constraint of the large feature model, as depicted in Figure 5.8, using the rules explained in the previous section is as follows: Package “php-pear” has option to include “mysql-server” or “postgresql-server”. Package “php-pear” depends on packages “php5-fpm”, “php5-cgi” and “php5-gd”. However, package “php5-cgi” has a conflict with package “php5-fpm”. As a result, package “php-pear” can only be installed with “php5-fpm” or “php5-cgi”, as illustrated in figure 4.5.

The steps of transformation are:

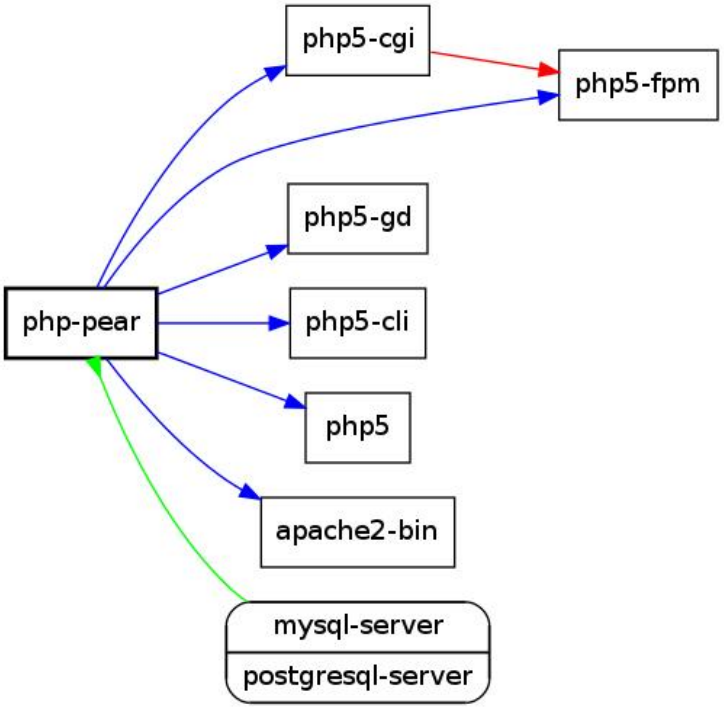


Figure 4.5: Example of package dependencies used to create a constraint rule in a feature model.

(1) php5-cgi $\Rightarrow \neg$ php5-fpm (Conflicts Rule)

(2) php-pear $\Rightarrow$ php5-fpm (Depends Rule)

(3) php-pear $\Rightarrow$ php5-cgi (Depends Rule)

(4) php-pear $\Rightarrow$ php5-gd (Depends Rule)

(5) php-pear $\Rightarrow$ php5-cli (Depends Rule)

(6) php-pear $\Rightarrow$ apache2-bin (Depends Rule)

(7) php-pear $\Rightarrow$ php5 (Depends Rule)

(8) php-pear $\Rightarrow$ (mysql-server $\vee$ postgresql-server) (Provides Rule)

Combines (1)(2)(3):

(9) php-pear $\Rightarrow$ php5-fpm $\vee$ php5-cgi

Combines (9)(4)(5)(6)(7):

(10) php-pear $\Rightarrow$ php5-fpm $\vee$ php5-cgi $\wedge$ php5-gd $\wedge$ php5-cli $\wedge$ apache2-bin $\wedge$ php5

Combines (10) and (8):

(11) php-pear $\Rightarrow$ php5-fpm $\vee$ php5-cgi $\wedge$ php5-gd $\wedge$ php5-cli $\wedge$ apache2-bin $\wedge$ php5 $\wedge$ (mysql-server $\vee$ postgresql-server)

The combination (11) is a constraint for the large feature model.

4.6 Chapter Summary

In this chapter, we presented a technique to create a feature model automatically from the Ubuntu package repository. This technique has used the graph approaches to add feature and modify the relationships between features which have shown their effectiveness in order to build a feature model from a large software repository. As presented in Section 4.4.1, this chapter has described steps to extract the graph information into a feature model. The package dependency describes the features cross-tree relationship and the package categories present the features hierarchy relationship. Section 4.5 presented a transformation from a graph syntax into a propositional logic format to match the feature model constraints. This transformation was built using the Debian/Ubuntu package dependency rules.

CHAPTER 5

A SOFTWARE PRODUCT LINE WITH ENERGY MANAGEMENT FOR A VIRTUALISED ENVIRONMENT

As explained in Chapter 1, this work proposes a technique using an amalgamation of Software Product Line (SPL) and an approximation of energy usage to reduce energy in a virtualised environment. Such an approximation technique is essential to find out the combination of features that consume less energy associated with a workload. In this work, for a small number of features, we can measure energy corresponding to all the possible combinations, but it is infeasible to measure the consumption of energy for all possible combinations that include a large number of configurations of features. As a result, this chapter describes a machine learning ensemble - which combines multiple learning algorithms to obtain better predictions - to make an estimation of energy usage for a large feature model.

Our work is achieved through the following steps: Section 5.1 presents machine learning techniques to create a prediction model for a Software Product Line development using a synthesis of workload, product configuration and energy consumption. These techniques make use of the CPU power information from the energy measurement results, as the prediction target. Section 5.2 introduces a technique to create a machine learning model from a feature model where features are retrieved from a software repository. This chapter then introduces Energy Prediction Trees (EPTs) in Section 5.3. EPTs aim to

make the results of a prediction more interpretable. In Section 5.4, we conduct a comparative study between EPTs and several other prediction techniques from the performance point of view. The comparison is based on the case study that is created from the Ubuntu package repository. Finally, Section 5.5 summarizes this chapter.

5.1 Machine Learning for an SPL with Energy

This section presents a technique using an approximation approach to select the combination of features from a feature model that uses less energy.

As explained in Chapter 2, there are a large number of possible components and variations of features within a Software Product Line with energy that is retrieved from the software repository. This, in turn, needs a method to predict the CPU power consumption of a given workload and configuration of features. Building a model to make such predictions can be viewed as a machine learning regression problem, i.e., the problem of learning how to predict a numeric target variable (CPU power) given a set of input variables (workload and configuration). Machine learning methods build predictive models based on a training set composed of examples of input values and their corresponding target outputs.

Several different machine learning methods exist for regression problems [23]. If the relationship between the input and target variables is linear, linear regression methods can be used. These methods usually mathematically and deterministically estimate the coefficients of linear equations so as to minimise the error associated with the estimations on the training set. One of their advantages is that they produce models that are easy to interpret and visualise. However, they are restricted to linear models.

If the relationship between the input and target variables is non-linear, the Multi-Layer Perceptron (MLP) is a very popular model [23]. MLPs are composed of several nodes arranged in an input layer, one or more hidden layers, and one output layer, as shown in Figure 5.1. Each node computes a function of the weighted sum of its inputs. Learning

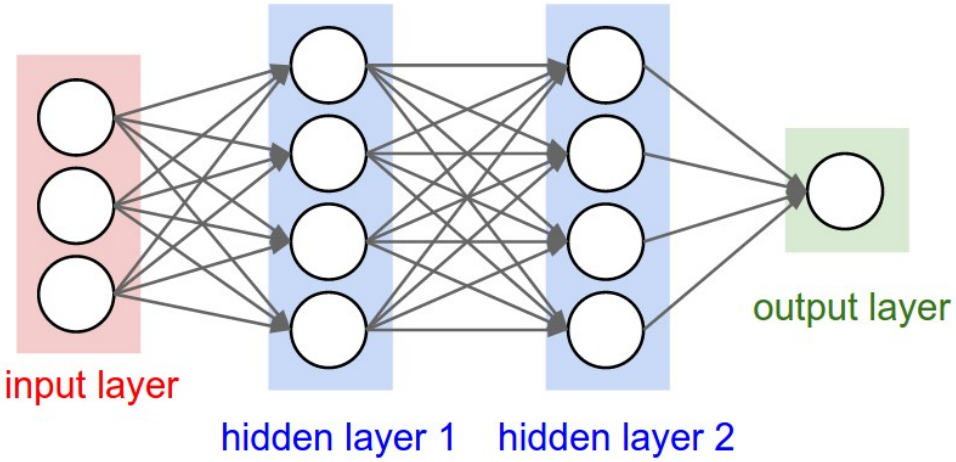


Figure 5.1: Example of structure of MLPs with three input and one target variable.

consists of determining the best weights so as to minimise the error on the training set. One of the most well-known algorithms for learning the weights is called Backpropagation [23]. A key advantage of MLPs is that they can approximate any continuous function. However, they are difficult to interpret by humans and tend to be sensitive to parameter choices, making them more difficult to use by practitioners [87].

Regression Trees (RTs) are also widely used with non-linear models. They divide the input space into a tree structure based on the values of the input variables. Learning consists of deciding which variables and values to split up at each level of the tree. When dealing with regression problems, splits are usually created so as to minimise the variance of the target values of the training examples allocated to each child node. The human interpretability of RTs is often seen as one of its key advantages over models such as MLPs. However, RTs produce predictions with discontinuities at the split boundaries, which can be an issue when aiming to model smooth functions, such as the measurement results explained in Section 3.2.1. It is worth noting that, in our measurement of energy, we generate a workload in a linear smooth way, where the load is sent in an increasing sequence order from 1 – 100 user requests per second. Therefore, the workload can be monitored as a linear increase of resource usage within the target of measurement.

In order to improve predictive accuracy, ensembles of models can be created by using methods such as Bagging [29]. By combining different models into an ensemble, it is

expected that the wrong predictions given by some of the models are compensated for by the correct predictions given by the others, increasing the predictive accuracy of the ensemble as a whole. Bagging uses the fact that models such as MLPs and RTs can be considerably different when created, based on different training sets. Bagging builds different models of a given type (e.g., different MLPs) based on different bootstrap samples of the training set. A prediction given by the ensemble is the simple average of the predictions given by each model. However, as ensembles are composed of several different models, they are more difficult to interpret.

In this work, we compare the performance of Linear Regression, MLP, RT and bagging. We aim to find which machine learning approaches perform better to predict the consumption of energy from different combinations of features.

5.1.1 CPU Power Consumption Prediction

As explained in Chapter 1, we use machine learning for predicting the CPU power consumption of different workloads and configurations. According to Andrew [163], when applying machine learning for a given problem for the first time, at least, these questions should be asked :

(Q1) Can we consider the performance of the methods acceptable for the problem in hand?.

(Q2) Which machine learning method is best to use for this problem?, and

(Q3) Can we get any insight into the problem, based on the models created?

This section presents a comparative study of the performance of models that is explained in the previous section, aiming at answering the first two questions above. The interpretable models are also analysed to gain insight into the problem, answering the third question. Section 5.1.2 reports the setup of the prediction process and section 5.1.3 reports the analysis done to answer the questions above.

5.1.2 Experimental Setup

In this work, the data set is obtained from an experiment that measures the consumption of energy of a variant php-based web system [162]. We use WEKA [102], a machine learning tool, to help us predict energy usage from a feature model of a Software Product Line. WEKA's implementation of linear regression, MLP, RT (REPTree), bagging using MLPs and bagging using RTs was used with its default parameters in the experiments [102]. Using the default parameters is a reasonable choice for a first analysis, as practitioners would frequently leave parameters untuned unless they are experts in machine learning.

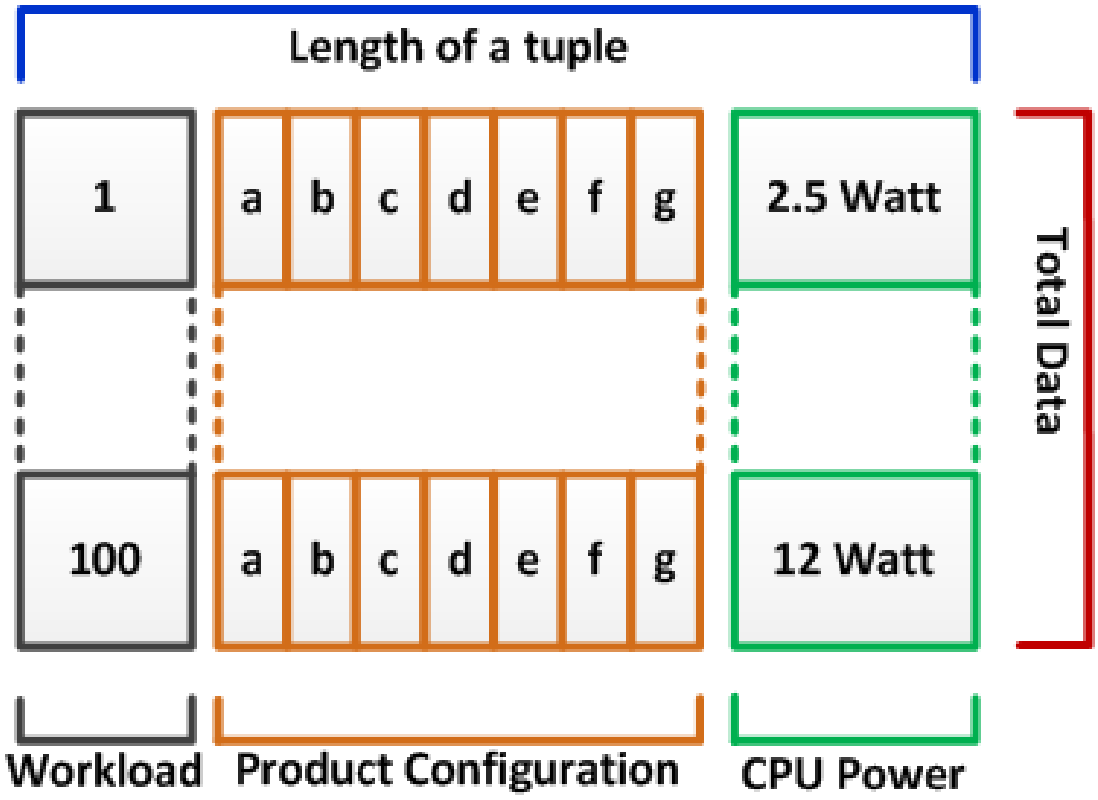


Figure 5.2: Example of a data set for machine learning setup.

Figure 5.2 describes the data set, which consists of workload, combination of features and CPU power. This data set combines the feature model and the results of energy measurement. As depicted in Figure 5.2, this data set consists of the following: one input variable is used to describe the workload, seven binary variables (a – g) are used to describe the configuration of features, and the target variable is the CPU power. The combination

of features is depicted in Table 5.1 showing six configurations. The feature model for this experiment can produce more configurations, as shown in Figure 5.3, but we create constraints to limit it to only six configurations, in the direction of reducing the complexity of the feature model. In order to limit the possible configurations from a feature model, we create a cross-tree relationship. For example, when we include the feature lighttpd into a product configuration, it must be configured with php-cgi or php-fpm (php-cgi $\vee$ php-fpm), and exclude php-native ($\neg$php-native). When nginx is selected, then php-fpm must be included, excluding both php-cgi and php-native ($\neg$php-cgi $\wedge$ $\neg$php-native) from the configuration.

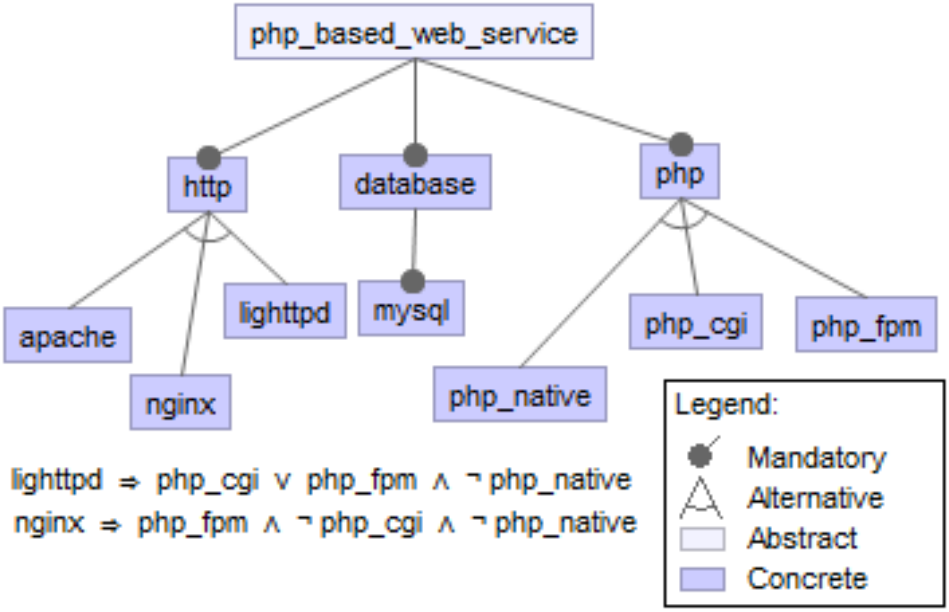


Figure 5.3: Feature model of php-based web system.

The performance measures used in this study were Mean Absolute Error (MAE), Root Mean Square Error (RMSE) and Median Magnitude of the Relative Error (MdMRE), as explained in Section 2.5.3.

Our approach can be scaled up to include many more features. One of the methods is by retrieving a large scale software repository, such as the Ubuntu package repository, where the package dependencies within it can be transformed into a feature model, as explained in Chapter 4. Furthermore, the combination of the measurement of energy

Table 5.1: Configuration variables in the data set of php-based web system.

a	b	c	d	e	f	g	Configuration
0	0	1	1	0	0	1	apache + php-cgi + mysql
0	0	1	0	1	0	1	apache + php-fpm + mysql
0	0	1	0	0	1	1	apache + php-native + mysql
1	0	0	1	0	0	1	lighttpd + php-cgi + mysql
1	0	0	0	1	0	1	lighttpd + php-fpm + mysql
0	1	0	0	1	0	1	nginx + php-fpm + mysql

Boolean indicating whether a feature included (1) or not (0) into the configuration.

values, the feature configurations and workloads create the data set for our approximation approach.

5.1.3 Experimental Results

Table 5.2 shows the results of the average test performance. Given that the average and standard deviation of the target CPU power levels in the full data set were 18.56 and 7.33, we can consider the average MAE and RMSE to be generally small for all methods. This is further confirmed by the very low MdMRE. For instance, the MdMRE for bagging+RT was only 2.22% of the actual CPU power levels. This answers Q1 concerning the performance of the acceptable methods, as mentioned in Section 5.1.1, showing that the methods investigated in this study present a good performance for the problem of predicting CPU power. It is worth noting that we used the default parameters of the methods for these experiments. If the parameters are tuned, even better results may be obtained.

Table 5.2: Average performance (+- standard deviation) of machine learning methods for predicting CPU power.

Method	MAE	RMSE	MdMRE
Linear regression	0.79+-0.22	1.74+-1.01	2.43+-0.52
MLP	1.17+-0.56	2.07+-1.03	4.41+-2.81
RT	0.72+-0.18	1.28+-0.60	2.64+-0.43
Bagging+MLP	0.89+-0.25	1.86+-1.00	2.66+-0.85
Bagging+RT	0.61+-0.14	1.15+-0.52	2.22+-0.38

Table 5.2 shows that the MAE and RMSE of each method was compared to the highest ranked approach (bagging+RT), based on the Wilcoxon Sign-Rank Test with Holm-Bonferroni corrections at the overall level of significance of 0.05. The p-value is a random variable derived from the distribution of the test statistic used to analyse a data set and to test a null hypothesis [100], where p-value < 0.005 indicates very strong evidence against the null hypothesis in favour of the alternative. In this experiment, all p-values were smaller than $2.5 \cdot 10^{-5}$. Therefore, we can conclude that bagging+RT obtained significantly better MAE and RMSE than the other methods. It is thus recommended over the other methods for predicting CPU power, if the main aim is to achieve high performance. This answers part of Q2.

It is important to notice that the magnitude of the differences in performance between bagging+RT and the other methods is not large, despite being statistically significant. For instance, the difference in MAE between bagging+RT and RT, which is the second ranked approach in terms of MAE and RMSE, is only 0.11 units of power. Therefore, a practitioner might prefer to use RT instead of bagging+RT in order to get more insight into the problem. This provides the remainder of the answer to Q2, finding a suitable machine learning method for our problem of the consumption of energy by different combinations of features.

In order to answer Q3, we thus provide a closer look into the RT model. As ten times ten fold cross-validations generate 100 different models, we report here insights gained based on a single RT model built using the full data set as training data. This RT model used the input variable workload as the sole variable to create splits from the first to the fourth levels of the tree. This means that the workload was the main factor affecting CPU power, which is reasonable.

As the RT divided the input space into several different workload sections, it was quite large. In this section, we present only part of the fourth to eigth levels of the RT in Figure 5.4 as an example, and put the whole RT in Appendix A.

Depending on the workload value, no further splits were performed by the RT based

on configuration. This is the case, for example, for the workload in the interval $[93.5, 96.5)$ shown in Figure 5.4. This gives the insight that, depending on the workload, different configurations are unlikely to lead to a significant impact on CPU power consumption. However, for certain workload values, configuration becomes more important. The configuration variable c was the most importance one, having been used to make several splits, as depicted in figure 5.4. This gives the insight that the choice between configuration $c = 0$ and $c = 1$ (i.e., using or not using Apache) is the one that will usually affect CPU power usage the most.

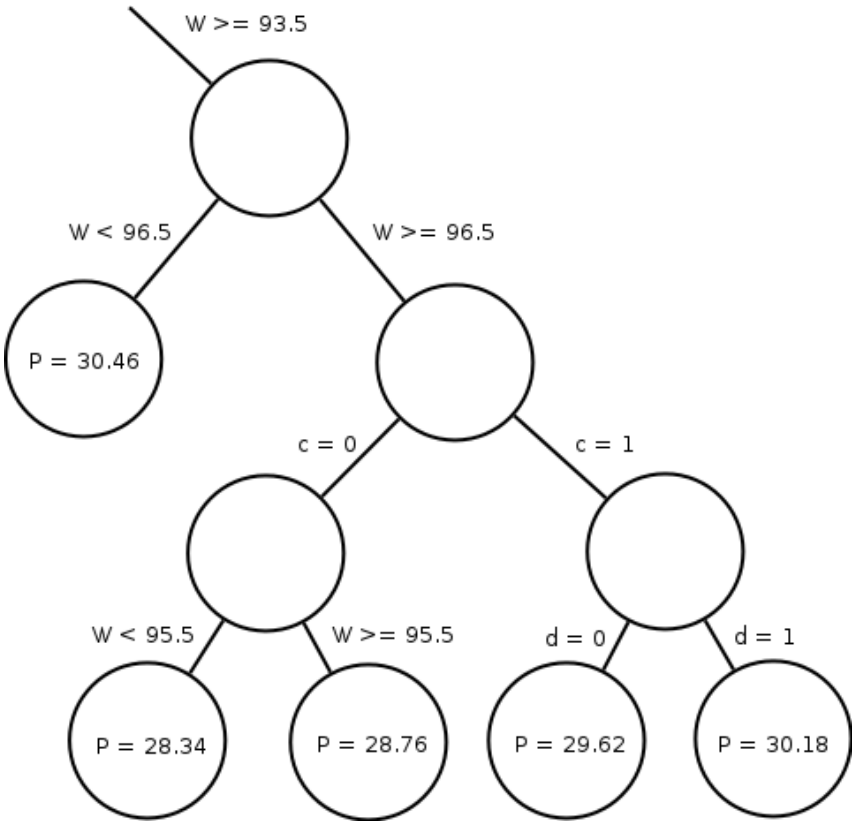


Figure 5.4: Section of the RT model created using the whole data set as training data. W stands for workload and P stands for CPU Power. c stands for Apache and d stands for PHP-CGI.

The linear regression model could also be used to get some insights into the problem. However, as it is constrained to a linear function, it is likely to be less accurate compared to reality. For instance, the relationship between workload and CPU power for a fixed configuration is roughly linear, but there are some non-linear variations in CPU power.

Linear regression will ignore potential non-linear variations in CPU power that could be caused by certain workload levels or configurations.

The linear regression model created using the whole data set as training data is the following:

$$\text{CPU power} = 0.244 * \text{Workload} + 0.6696 * c + 5.8008$$

We can see that the only configuration variable considered important by the linear regression model was c. This model gives the insight that, approximately, a configuration using $c = 0$ (i.e., not using Apache) tends to use less CPU power, and other configuration choices tend not to affect CPU power as much as c. If one is trying to improve a given service, one may wish to invest mainly in improving the potential to provide configurations $c = 0$ (i.e., configurations that avoid the use of Apache). Examples of such configurations are nginx + php-fpm + mysql and lighttpd + php-cgi + mysql.

5.2 Creating a Data Set for Machine Learning Model from a Software Repository

This section introduces a technique to create a data set for the machine learning process for a Software Product Line with energy from a large software repository, such as the Ubuntu package repository. As explained in Chapter 4, *debtree*, an open source tool to generate feature dependencies from the Debian /Ubuntu repository, is used extensively. Using this technique, we can resize the feature model according to our needs, i.e. excluding conflict packages.

In order to provide information on the energy-efficiency of different configurations, we need to, first, extract a feature model from the software repository being used and second determine the energy consumption associated with different configurations from this feature model. The process of extracting a feature model from a large software

repository is very time consuming, even if the intended application is associated with a small number of different possible configurations. As explained in Chapter 1, it is desirable to have an automated method to extract feature models when dealing with large repositories. When dealing with a very small feature model, it may be possible to measure the energy consumption of each possible configuration in order to find out which of them is more energy efficient. However, more complex applications can result in a large number of possible configurations, making infeasible not only the manual extraction of the feature model, but also the measurement of the energy consumption of all possible configurations. Machine learning approaches investigated in the previous section on predicting energy consumption would lead to very large models [162], from which it is difficult to gain insights into what configuration to choose.

In this work, we deal with the problems explained above by proposing an interpretable and scalable machine learning model design for providing insights into which configuration to choose. The next subsection will describe steps to create the Machine Learning data set from a feature model that is built from a large software repository.

5.2.1 Steps to Building a Machine Learning Model from a Large Software Repository

This section explains the steps to build a machine learning model for energy prediction from a large software repository. The steps included in retrieving packages to build a feature model from the repository have already been explained in Chapter 4 (using Debian utility to retrieve the package dependencies).

The approach is depicted in Figure 5.5. Our approach to building a Machine Learning model consists of the following steps:

- **Step 2 – Build a Machine Learning (ML) Model for Predicting Energy Consumption:** three steps are required to build an energy prediction model in order to avoid measuring energy consumption of all possible configurations and

workloads. They develop from **Step 1** that creates a feature model from a large software repository – as explained in Chapter 4.

- **Step 2.1 – Create input variables for ML model:** features from the feature model are used as binary input variables of the ML model in order to represent a configuration. The binary value 1 means a feature is included in a configuration, and 0 not included. Each input variable indicates whether an existing package is used in the configuration or not . An additional numeric input variable represents a given workload, with higher values indicating a higher workload.

- **Step 2.2 – Generate data set:** a subset of all possible configurations is selected in order to create a data set for building or evaluating the ML model. The configurations that make up the data set can be selected randomly, or partially randomly, by ensuring that each package is included in at least one configuration and popular configurations are included, so that there is a representative of well-known configurations. For example, to generate a data set for a Blog system, the "Wordpress" configurations should be included. Different workloads are then simulated as explained in Section 3.2. The workloads are fed into the system using the selected configurations, and their corresponding energy consumption is measured using Powertop, as described in Chapter 3 [161]. Each tuple of the configuration, workload and energy measurement forms one example in the data set. It is worth noting that the energy measurement performed here is platform specific, e.g., a measurement taken on VirtualBox x64 may be different from a measurement taken on VMWare X86_64. Therefore, the data set should be created with a specific platform in mind.

- **Step 2.3 – Build ML model:** the data set created in step 2.2 is then fed to a ML algorithm in order to build a model able to predict the energy consumption associated with a given configuration and workload. Details on how to build the ML model proposed in this work are given in Section 5.3. This ML model is specific to the platform used for generating the data set, e.g., if the data set created was

based on energy measurements on VirtualBox_x64, the ML model should only be used for making predictions for this platform.

These steps are bonded to the procedure that creates a feature model from a large software repository, as explained in Chapter 4. Any changes of product configurations will affect the prediction of energy usage.

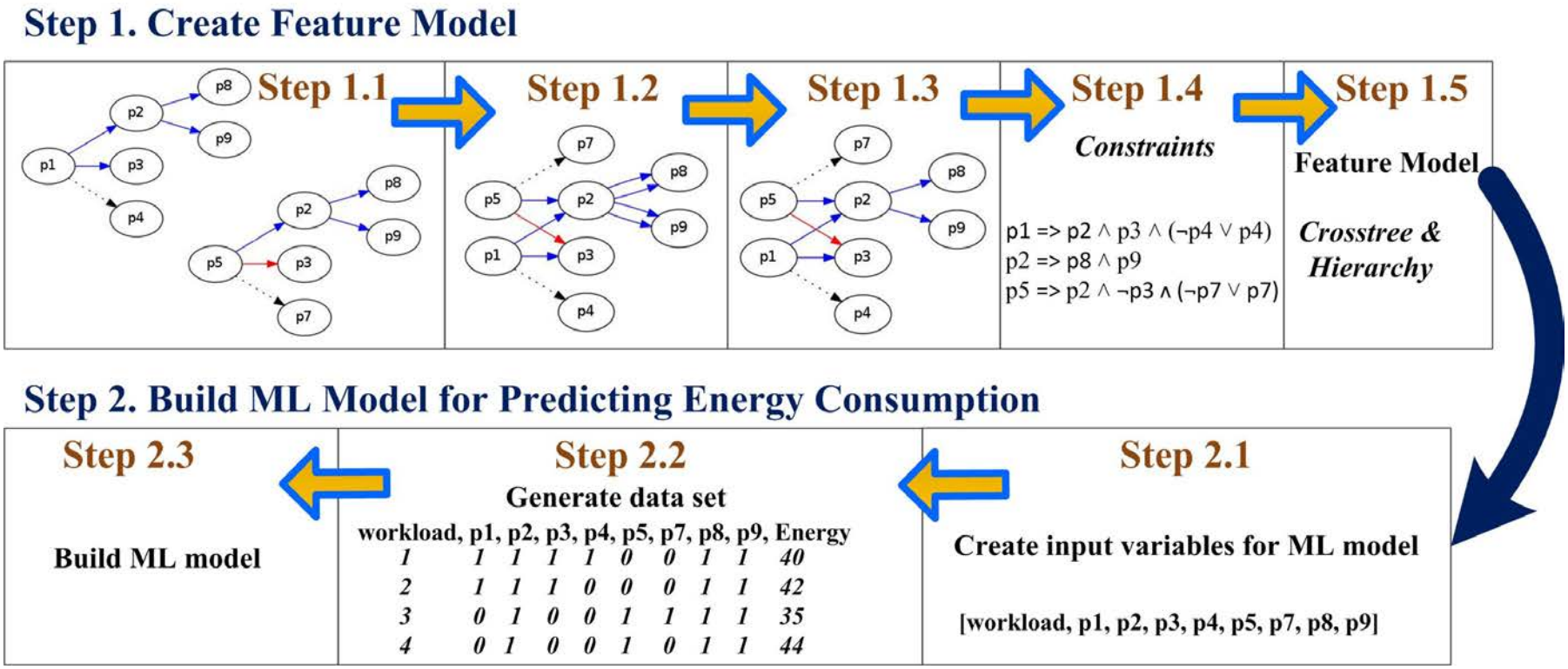


Figure 5.5: Step 1–2 to build feature model and ML prediction model for predicting energy consumption from the Debian package repository. Step 1 already explained in Chapter 4.

5.3 Energy Prediction Trees

This section presents an approach to approximating the energy usage of a feature model built from a large software repository. This approach extends the regression tree model to give a better prediction result.

The feature model generated using the method proposed in Chapter 4 can be used to determine all valid configurations. However, measuring the energy consumption of all valid configurations is infeasible when the number of configurations is high. For example, the large feature model, as shown in Figure 5.8, has 4519 possible configurations. In that case, as explained in Section 1.2, we need an accurate and interpretable model for predicting the energy consumption of configurations and workloads without the need for measuring the energy consumption of all valid configurations. This section explains our proposed Machine Learning (ML) model design for predicting energy consumption based on a subset of the configurations from the feature model. This method combines the advantages of Regression Trees (RTs) and linear regression, and is expected to be both accurate and interpretable in this domain.

As explained in Section 5.1, the ML methods Bagging+RTs and RTs are the most accurate methods in the literature for predicting CPU power (target variable) based on configuration and workload (input variables) [162]. However, Bagging+RTs are inherently not easily interpretable because they consist of several RTs and Single RTs are hard to interpret because their size becomes very large in this domain. The probable reason for their large sizes is that CPU power consumption increases smoothly with the input variable *workload*. This causes RTs to create several node splits based on workload, leading to very large trees. Figure 5.4 in Section 5.1.3 can be used to illustrate this problem. From the 163 nodes of the RT part of which is shown in that figure, 112 were split into branches based on workload. The whole RT can be found in Appendix A. Another problem of RTs is that their discontinuous predictions are not ideal for modelling smooth functions.

In order to overcome these issues, we propose a new RT design tailored for this domain,

called Energy Prediction Trees (EPTs). The design is based on the observation that CPU power is expected to increase fairly linearly with the workload, and that different configurations can lead to different linear functions, causing CPU power to increase more or less rapidly. Figure 5.6 shows an example of this for two different configurations. We thus restrict our EPTs to make splits only on input variables representing configurations, i.e., splits on workload are not allowed. Each leaf of the EPT uses its corresponding training examples to build a linear regression model of CPU power based solely on the workload. This model is used for providing CPU power predictions based on the workload of any configuration corresponding to the leaf. This design is expected not only to reduce the size of the trees, but also to improve predictive accuracy, by avoiding discontinuous predictions through the linear regression models.

We also note that different configurations can only cope with workloads up to a maximum level, above which the extra workload is simply rejected by the CPU. For example, as shown in Figure 5.6, this maximum is just above 20 for configuration 1, whereas configuration 2 can withstand larger workloads. We speculate that configuration 2, that includes packages Apache2, Tomcat, MySQL and Java, does not making much system calls to the operating system compared with the configuration 1 that configured with Apache2, MySQL, php-fpm and Pear when an increasing amount of workloads arrives at the system. Although, the configuration 1 and 2 have commonality, both include Apache2 as an HTTP server, MySQL as a database management server and run in the same operating system - Ubuntu 14.04, their configuration with Java and php-fpm create different behaviour, in terms of CPU energy usage. We argue that the php-fpm create many more system calls to the operating system than Java, which utilises CPU to its maximum processing capacity. Moreover, the CPU power measurement for the minimum workload is frequently an outlier, due to the measurement process. We use this prior knowledge to filter the data, eliminating training examples that would otherwise hinder the linear regression learning procedure. Specifically, we filter out training examples containing a workload above the maximum for each given configuration, and training examples corre-

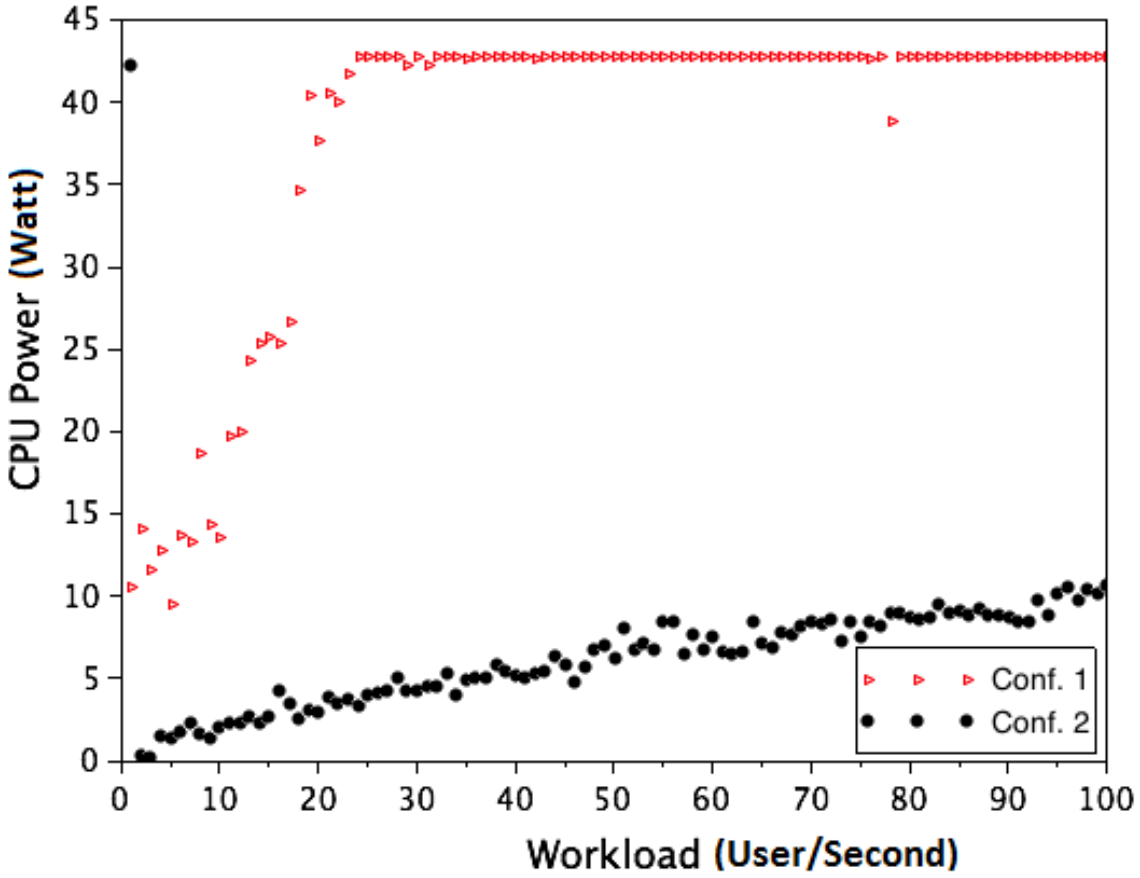


Figure 5.6: Examples of plots of CPU power versus workload for two different configurations. Configuration 1 uses Apache2, MySQL, Ubuntu 14.04, PHP-FPM and Pear. Configuration 2 uses Apache2, Tomcat, MySQL, Ubuntu 14.04 and Java.

sponding to an inaccurate CPU power measurement due to a minimum workload. Should the EPTs be required to make predictions for workloads above the maximum allowed for a given configuration, the maximum CPU power actually measured for that configuration should be provided, rather than the CPU power predicted by the corresponding linear model.

Given that the EPT's leaves contain a linear regression model rather than a prediction of a single CPU power value, we build our EPTs using the model-based recursive partitioning procedure proposed by Zeileis et al. [226]. This procedure has a rigorous theoretical background for integrating parametric models (such as linear regression) into trees. Its basic steps are as follows:

1. To build a parametric model using all the training examples associated with the current node, i.e., using the current *sample*. In the case of EPTs, this parametric model is a simple linear regression model.

2. To check whether splitting the sample in some input variable can capture instabilities in the model and thus improve the fit of the model to the data. This is done based on a modern class of instability statistical tests [225].

3. To split the node on the input variable associated with the highest instabilities, i.e., the variable that can improve the fit the most.

4. To repeat the process for each child node.

5.4 Creating and Evaluating Machine Learning Models for CPU Power Prediction

This section presents the experiments performed with the objective of evaluating our Energy Prediciton Trees (EPTs) in terms of predictive accuracy, interpretability and scalability. To this end, experiments were performed not only using EPTs, but also the Machine Learning (ML) approaches that have shown to perform best (Bagging+Regression Trees (RTs) and RTs) and that have been shown to be the most interpretable ones (Linear Regression (LR)) in the domain of CPU power prediction [162]. RTs have also been shown to perform well in the related domain of performance prediction. In addition, we have included an extra model for the experiments, namely Random Forests (RFs) [31].

RFs are ensemble learning methods that combine the idea of bagging [29] with random selection of input variables. In order to decide the splits of the decision trees, a certain number of randomly chosen input variables is selected, rather than considering all input variables for the split, as is the case for Bagging. This can help to reduce the correlation between the trees that compose the ensemble, potentially resulting in a better prediction performance than bagging+RTs [31]. Therefore, even though RFs have the same inherent

interpretability problem as bagging+RTs, it is worth investigating them in this domain to check how much improvement in predictive accuracy they can provide. To the best of our knowledge, RFs have not been used for CPU power prediction in the past.

The experiment activity is divided into subsections as follows: Section 5.4.1 explains the data sets, Section 5.4.2 explains the design of the experiments and Section 5.4.3 presents the analysis of the results.

5.4.1 Data Sets

As explained earlier in Chapter 5, we extracted both a relatively small and a large feature model from the Debian/Ubuntu package repository in order to investigate the scalability of our approach. In order to build the ML models (step 2.3 of the procedure as explained in Section 5.2.1) corresponding to each of these feature models, we first need to determine what the input variables are (step 2.1) and generate data sets (step 2.2).

We have 23 and 57 input variables corresponding to the small and large feature models, respectively. Of these, 22 and 56 variables are binary variables representing whether or not each of the 22 and 56 packages from the feature models is used. The remaining input variable is a numeric variable representing the workload. The target output is CPU power in Watts.

We created two data sets S and L, one corresponding to the relatively small and one corresponding to the larger feature model. Each example in a data set represents a given configuration (combination of packages), workload and its associated CPU power in Watts. Different examples for data sets S and L were created by selecting 12 and 29 different possible configurations, respectively, based on the feature models extracted in Chapter 4. The feature models are shown in Figure 5.7 for the small feature model and in Figure 5.8 for the large feature model. The configurations were chosen so as to ensure that each package was used in at least one configuration and that the most popular configurations, according to the web applications used in this study, were covered. Workloads were generated as explained in Chapter 3 and varied from 1 to 100 users, leading

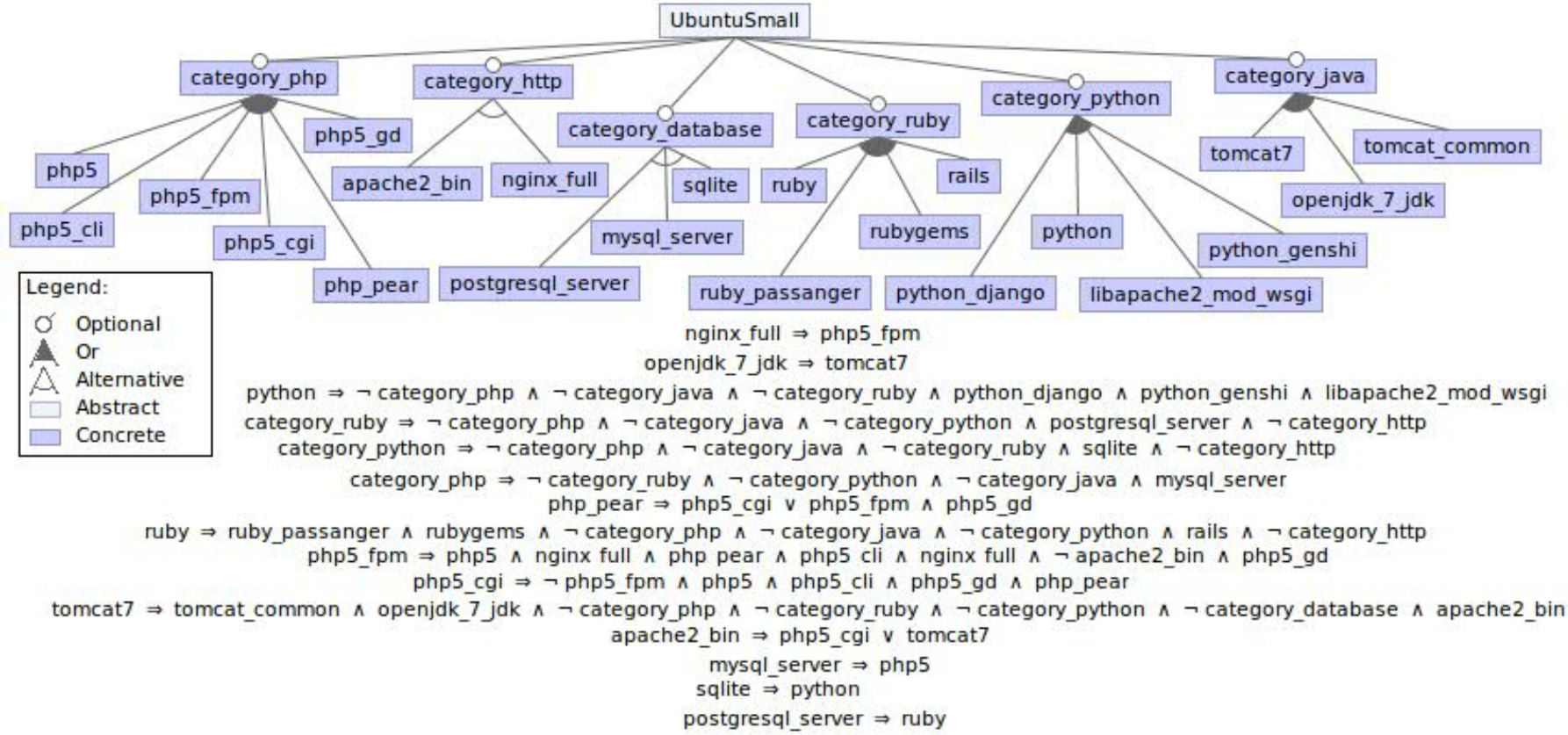


Figure 5.7: Small feature model – The small feature model has 12 possible configurations.

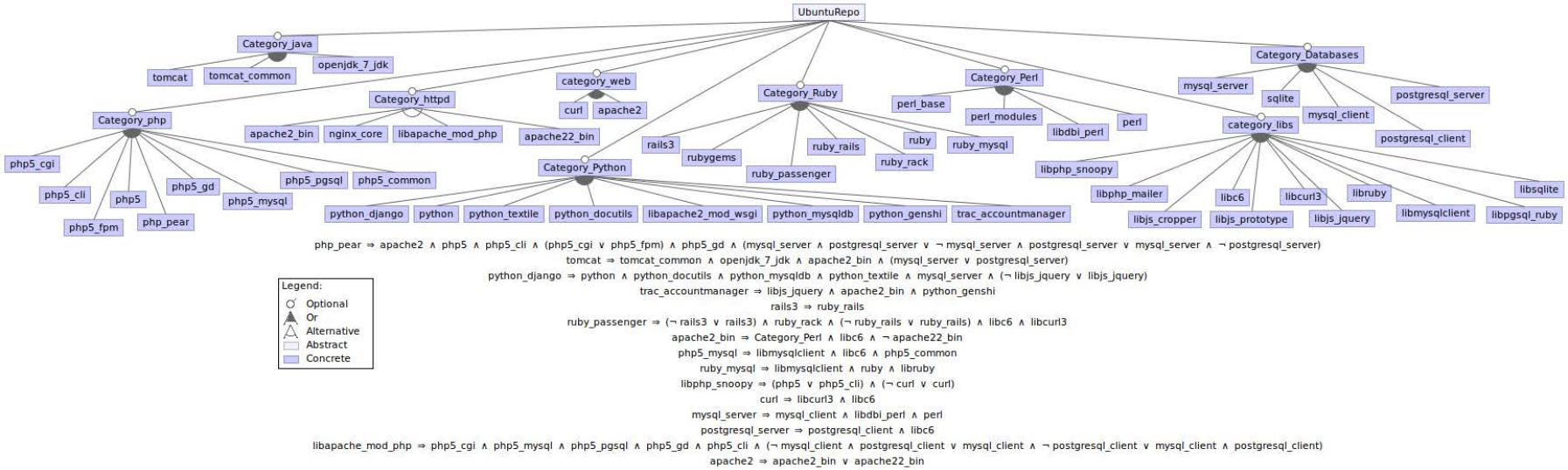


Figure 5.8: Excerpt of the large feature model – The full large feature model has 4519 possible configurations.

to a total of $12 \times 100 = 1200$ and $29 \times 100 = 2900$ examples for data sets S and L, respectively. Each workload level was simulated for 10 seconds for each configuration on the VirtualBox_x64 platform. Our energy measurement method for virtualised systems [161] was used to capture the CPU power consumption at every second. As the measurement procedure contains noise, for each given workload and configuration, the median of the 10 measurements was used as the target output. The data sets were then filtered to exclude examples with the minimum workload and with a workload above the maximum supported by each given configuration. This resulted in 667 and 1504 examples for data sets S and L, respectively.

5.4.2 Experimental Design

The evaluation procedure used in the experiments was 10 times 10 fold cross-validations, which is a standard evaluation procedure in Machine Learning (ML). This gives a total of 100 runs using different subsets of data for training and testing each ML approach. The performance measures used in this study were Mean Absolute Error (MAE), Root Mean Square Error (RMSE) and Median Magnitude of the Relative Error (MdMRE), as explained in Section 2.5.3.

The number of nodes of the RT and EPT trees was used to compare their interpretability. For Bagging+RTs and RFs, the sum of the number of nodes of all their trees was used. LR is equivalent to a single node.

Wilcoxon Signed-Rank statistical tests were also performed with Holm-Bonferroni corrections at the overall level of significance of 0.05 to support the comparison of size and performance of the ML models.

In performing a significance test in statistics, p-value helps us to determine the significance of our data estimates [91, 184]. The significance tests are used to test the validity of a claim that is made about a model that is created using a data set and a Machine Learning (ML) approach, which has its calculation based on an assumption that null hypothesis is true [60], where the claim on trial is the null hypothesis. The evidence in the trial is our

measurement data and the statistics that collaborate with it. In this experiment, we use the significance level $\alpha = 0.005$ and the p-value can be interpreted as follow [184, 177],

- A small p-value (≤ 0.05) indicates strong evidence against the null hypothesis, which means our data set and an ML model are relevant for the prediction.
- A large p-value (> 0.05) indicates weak evidence against the null hypothesis, which means our data set and an ML model may not be relevant for the prediction.

The accuracy and size of the models created using data sets S and L were compared in order to analyse their scalability.

Following previous work [162], WEKA's implementation [102] of Bagging+RTs, RTs (REPTree) and LR were used to build the ML models in the experiments with its default parameters, unless stated otherwise. WEKA's RF implementation was also used with its default parameters, except for the number of trees, which was set to 10 in order to allow a fair comparison with bagging, which uses 10 trees in its default configuration. EPTs were created, based on the model-based partitioning algorithm provided by R's *party* package [226] with its default parameters, except for the parameter *MinSplit*. This parameter refers to the minimum permitted number of training examples in a node, and corresponds to the parameter *MinNum* of REPTrees. In order to allow fair comparisons between these approaches, *MinSplit* was set to the default WEKA value of 2 used in previous work [162].

5.4.3 Experimental Analysis

Our analysis is divided into two parts. The first part consists of investigating EPTs' interpretability and predictive performance in comparison with Bagging+RT, RT, RF and LR. Table 5.3 shows the size (in number of nodes) and Table 5.4 shows the predictive performance achieved by each model. The results of the Wilcoxon Signed-Rank tests for comparison of size, MAE and RMSE against EPT are also shown. All models' sizes, MAEs and RMSEs were statistically significantly different from EPT's, except for RT's RMSE for data set S.

Table 5.3: Median size (number of nodes) of ML models across 100 runs and p-values of Wilcoxon Sign Rank Tests for a comparison with EPT

Machine Learning Model	Data Set S (Small)		Data Set L (Large)	
	Size	P-Value	Size	P-Value
LR	1	0.00000	1	0.00000
EPT	19	–	43	–
RT7 / RT5	81	0.00000	43	0.80258
RT	154	0.00000	297	0.00000
Bagging+RT	1556	0.00000	2942	0.00000
RF	5219	0.00000	11901	0.00000

Models are sorted in ascending order of size. P-values in lime (grey) indicate statistically significant difference when using Holm-Bonferroni corrections at the overall level of significance of 0.05 considering six comparisons.

LR is the easiest model to interpret, as it always consists of a single linear regression equation. However, LR was the last ranked model in terms of predictive performance, both in terms of MAE and RMSE. By observing its MdMRE, we can see that the size of the difference in performance between LS and the first ranked approach (RF) was small (8.18 for data set S and 8.84 for data set L). This means that, as LR is easy to interpret, it is a good model in the domain of CPU power prediction, representing accurately the relationship between the configuration and the workload against CPU power.

RF was the first ranked model in terms of MAE, RMSE and MdMRE for both data sets S and L. However, it contained an extremely large number of nodes (5,219 for S and 11,901 for L), making it very difficult to be interpreted by people. Bagging+RT also contained a large number of nodes in this domain (1,556 for S and 2,942 for L).

EPTs were the easiest models to interpret after LR, containing 19 nodes for S and 43 for L. They also achieved significantly better MAE and RMSE than LR for both data sets. Even though the EPT's MAEs and RMSEs were statistically significantly worse than those of the best ranked approach (RF), the margin of the difference in performance was very small (only 0.83 for data set S and 0.71 for data set L). Therefore, it is advantageous to adopt EPT as opposed to LR, Bagging+RT and RF.

RTs had a considerably larger size and significantly worse MAE and RMSE than

Table 5.4: Median performance of different ML models across 100 runs and p-values of Wilcoxon Sign Rank Tests for comparison of MAE and RMSE against EPT

Machine Learning Model	Data Set S (Small)				
	MAE	P-value	RMSE	P-value	MdMRE
RF	1.63	0.00000	3.00	0.00000	7.20
Bagging+RT	1.79	0.00000	3.00	0.00000	8.41
EPT	2.03	–	3.53	–	8.03
RT	2.21	0.00020	3.61	0.40654	10.27
RT7	2.68	0.00000	3.97	0.00000	13.80
LR	3.32	0.00000	4.80	0.00000	15.38
	Data Set L (Large)				
	MAE	P-value	RMSE	P-value	MdMRE
RF	1.25	0.00000	2.26	0.00000	7.04
EPT	1.55	–	2.87	–	7.75
Bagging+RT	1.67	0.01208	2.77	0.00452	9.77
RT	2.04	0.00000	3.46	0.00008	10.81
LR	2.68	0.00000	4.01	0.00000	15.88
RT5	3.77	0.00000	5.53	0.00000	24.93

Models are sorted in ascending order of MAE. P-values in lime (grey) indicate statistically significant difference when using Holm-Bonferroni corrections at the overall level of significance of 0.05 considering six comparisons.

EPTs. Therefore, it is also advantageous to use EPTs as opposed to RTs. Moreover, an RT with a small data set has a p-value higher than 0.005, as shown in Table 5.4, which means based on a performance measurement using RSME, RT with small data set, statistically, is not significant to be used for our prediction of CPU energy usage. Even though RTs were larger than EPTs, RTs have a parameter which allows us to restrict their size. This parameter sets a maximum depth for the tree. Therefore, it is important to analyse whether restricting an RT's depth would allow us to obtain models competitive to EPTs in terms of interpretability and predictive performance. As the depth of EPTs created using the full data sets S and L was 7 and 5, respectively, we have also trained and evaluated (using cross-validation) RTs with a maximum depth restricted to 7 and 5 (RT7 and RT5) for data sets S and L. For data set S, RT's size is reduced from the median value of 154 to 81, whereas for data set L its size is reduced from the median value of 297 to 43. The reduction in size was particularly significant for data set L, probably

because RT5's depth is lower than RT7's. This depth was similar to EPT's depth on this data set. However, RT7's and RT5's predictive performance was drastically worsened. In particular, the magnitude of the differences in MdMRE compared to the EPT was 5.77 and 17.18 for data sets S and L, respectively. Therefore, despite achieving a reduced size, RT7 and RT5 create a misleading model in the domain of CPU power prediction. Even though RT5 creates a similar number of nodes to EPT, as shown in Figure 5.3, it does not indicate statistically significantly at the 0.005 level, which means the RT5 for small (S) data set are not significant for the estimation of CPU energy usage with relation to the workload and the combination of components.

We have also built RTs and EPTs using the entire data sets, rather than only the training sets, in order to check how the interpretability of these models would be affected. When using the full data sets, RTs had 169 and 275 nodes for data sets S and L, respectively. EPTs had only 13 nodes for both data sets S and L, as shown in figures 5.9 and 5.10. This means that adding more training examples can help EPTs to find more concise representations of the relationships between configurations and workloads against their CPU power consumption. RTs, on the other hand, did not become concise even after adding extra training examples.

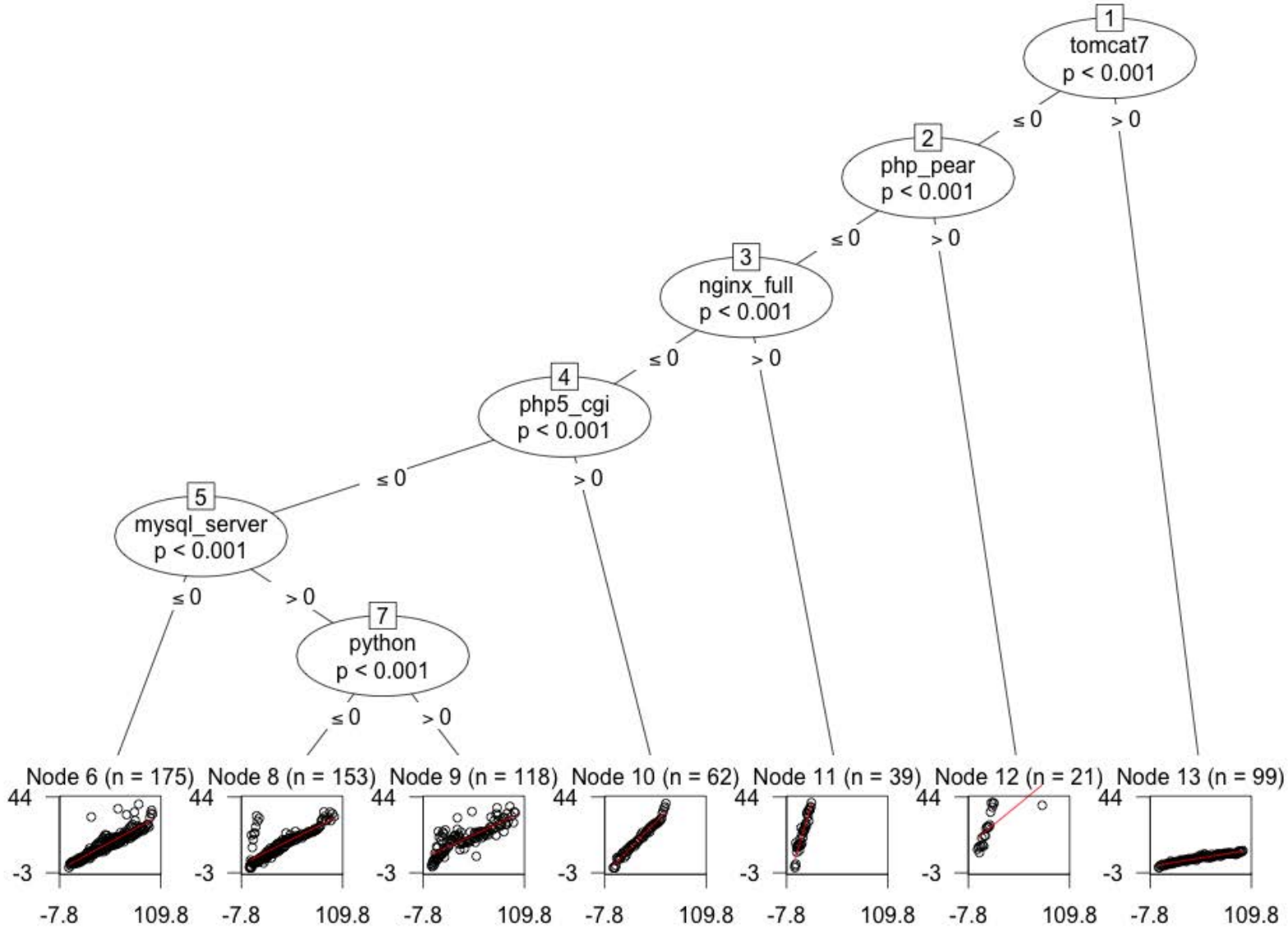


Figure 5.9: EPT created using the full data set S (Small). The circled points plotted in the leaf nodes represent training examples. The number of training examples associated with each leaf node is n. The red lines show the LR model created in each leaf.

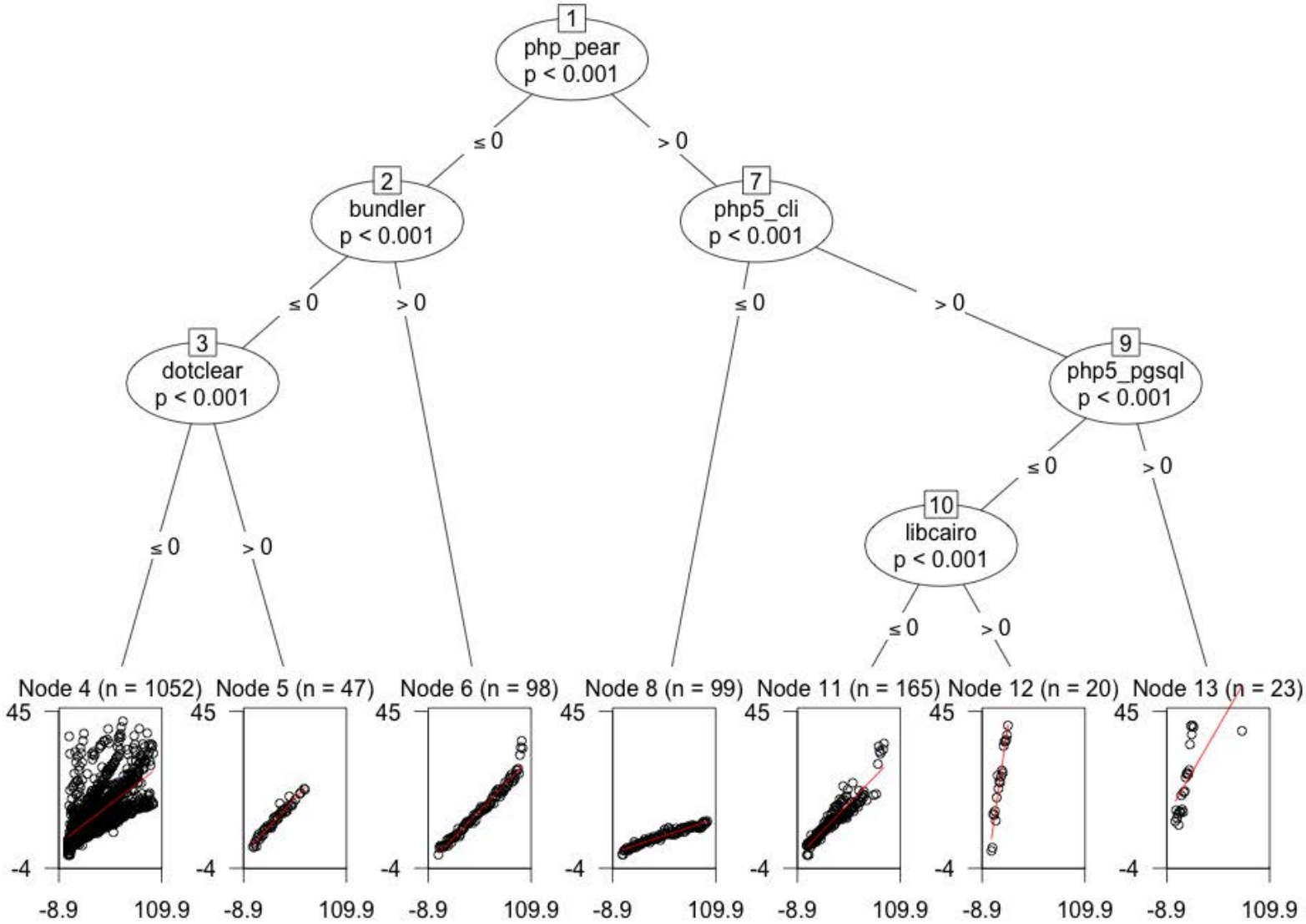


Figure 5.10: EPT created using the full data set L (Large). The circled points plotted in the leaf nodes represent training examples. The number of training examples associated with each leaf node is n. The red lines show the LR model created in each leaf.

It is worth noting that the EPTs shown in Figures 5.9 and 5.10 are easy to interpret by humans. They can be used to decide what packages to choose in order to reduce CPU power when using the Ubuntu/Debian packages repository, and to find out how much CPU power this choice is likely to save. Input variables not included in the model are considered unlikely to affect CPU power consumption significantly. Input variables at higher levels of the tree have a stronger impact on CPU power.

Moreover the EPT's predictive performance may be improved even further by eliminating outliers from the training examples associated with each leaf. Figures 5.9 and 5.10 show that the linear models created in certain nodes (e.g., node 12 for data set S and node 13 for data set L) were badly influenced by single training examples, with workload much higher than the others in that node. The use of techniques for filtering outliers from leaf nodes is proposed for future work.

The second part of our analysis consists of analysing EPT's scalability. In terms of size, the EPT's number of nodes increased from 19 to 43 when moving from the smaller data set S to the larger data set L. Data set L has 2.26 times more than the number of input variables of data set S, and its corresponding EPT's number of nodes was 2.48 times the number of nodes of data set S's EPTs. We can consider this as an acceptable increase in the number of nodes. Indeed, when using the whole data sets for building the EPTs, data set L's and S's EPTs were the same size. This is very good behaviour in terms of size scalability.

In terms of predictive performance, it is interesting to observe that EPT's performance for data set L was very similar to EPT's performance for data set S. The magnitude of the difference in MdMRE was negligible (only 0.28). The 29 configurations used to create data set L's examples represents a linear increase over data set S's 12 configurations, given the increase in the number of input variables from 23 (data set S) to 57 (data set L). Therefore, EPT's scalability in terms of predictive performance was good, i.e., it did not deteriorate as the number of input variables increased.

In Figure 5.10, nodes 4 and 13 show anomalies as training examples, the circle points

plot in the leaf nodes, have extreme differences where node 4 has a lot more plots (1052 nodes) compared with node 13 (23 nodes). We speculate that, during our measurement of energy, the first arrival of the workload to the system utilised the CPU to its maximum capacity, which resulted in spikes at the beginning of measurement. These spikes are outliers for the prediction of energy usage, which might lead to an unbalanced distribution of the training examples. We argue that by filtering the outliers, we can get a bettter prediction. The case in nodes 4 and 13 also raises new future work about method to reduce noise in the data set to predict energy usage using EPT.

5.5 Chapter Summary

This chapter has presented a technique to predict the consumption of energy by combinations of features in a Software Product Line (SPL) for a virtualised system. This prediction technique performs well for our problem, which includes combinations of workloads, features and energy usage. Section 5.1 described a prediction method using an ensemble of learning to find out the configurations of features that consumed less energy. This chapter also showed that splitting trees of Regression Trees (RTs) based on input variable workload or product configurations, can make an impact on the prediction results. Section 5.3 presented a technique called the Energy Prediction Trees (EPTs) combining the Regression Tree (RT) with Linear Regression, to produce both accurate and interpretable prediction results, by restricting the tree to make splits on input variables representing configurations. Our experiment, in Section 5.4, used the data sets that are created by retrieving the package dependencies from the Ubuntu package repository. This experiment showed that the EPTs outperform several prediction techniques to be more accurate. Furthermore, the EPTs results are easy to interpret by humans, as the EPTs combine the tree that was split based on configurations, and the linear regression models on each of its leaves.

CHAPTER 6

A DYNAMIC SOFTWARE PRODUCT LINE WITH ENERGY MANAGEMENT FOR A VIRTUALISED SYSTEM

As discussed in Chapter 5, we can select the configuration of features that consume less energy, using a consolidation of Software Product Line (SPL) engineering and approximation of energy usage. When this approach is executed in a continuously changing environment, such as a Cloud system, it becomes more challenging because the process of selection should adapt to the change of environment (e.g. workloads and energy). As a result, we need to reconfigure the software architecture within a virtualised system that corresponds autonomously to the change in environment.

This chapter presents an approach to reducing costs and energy usage for a virtualised environment that proposes a mechanism to handle different workloads and energy consumption. In this work, we build an architectural design that adapts to the change of workload and energy usage without scaling-up/down the reference architecture. Such a design reconfigures the software architecture autonomously, without interrupting the running services. We also present a technique to define and to enforce the high-level policies in a dynamic system that allows us to add new requirements without stopping or restarting the running system.

A technique to create a Dynamic Software Product Line (DSPL) with energy is introduced in Section 6.1, and we also suggest a dynamic variability – variant of features that

can be selected based on the change of the environment of a dynamic system – to support the selection of variants and a policy-based mechanism to manage the feedback-response of a dynamic system in order to build a runtime model. Section 6.2 then presents an application of a DSPL with energy using a self-adaptive load-balancer, which makes use of a Rule-Based System to create an autonomous system. In Section 6.3, we evaluate our approach by conducting an experiment on a self-adaptive load-balancer with a Rule-Based System (RBS) within a virtualised system. This experiment provides the proof of concept for our approach for an autonomous system that responds to the change of environment.

6.1 Modelling a Dynamic Software Product Line with Energy

This section introduces a technique to model a dynamic software product line to manage energy usage in a virtualised environment. Such a technique does not require the running system to be stopped and restarted.

We categorize the development of a dynamic software product line with energy for a virtualised environment into two stages. In the first stage, a platform and reference architecture is prepared and developed. This includes identifying the dynamic variability and product configurations which influence the consumption of energy. The second stage presents a runtime technique to serve a dynamic system, which creates a dynamic model that adapts to the changes of environment. The second stage also provides a mechanism to select which configuration should be applied to a system that is suitable to the current contexts, such as the change of workload. It is worth noting that the second stage depends on the first stage.

First Stage

In this stage, we begin by identifying the dynamic variabilities from a feature model to describe which features *adapt* to the change of environment. This selection is based on

input from the environment, such as workload and energy, which change continuously over a time line. Accordingly, features of a feature model should be related to the change of environment.

In a virtualised system, the workload utilises the computation unit that has always made transitions over time, because the consumption of energy dynamically shifts from low to high, or vice versa, in an unpredicted time. To identify whether a workload may influence the computation unit demands, a measurement of energy within a running system is calculated, as presented in Chapter 3 and 5. We create a simple prediction, as explained in Chapter 5, as pre-knowledge for the next stage.

Second Stage

In this stage, a dynamic system is built based on the first stage. We make use of models@runtime [24] to deal with the change of environment (workload and energy) and to reconfigure the virtual machine's software architecture. The models@runtime decides which composition should be reconfigured, based on the change of environment. We implement a policy-based mechanism to manage the change of configuration that also enables us to add new requirements into the system without interrupting the running system. To execute the decision that is made by the policy-based mechanism, a configuration manager composes the combinations of components into the system.

6.1.1 Creating a Dynamic Variability Model

This section presents a technique to create a dynamic variability model for a virtualised environment.

Unlike a product configuration for a static system, a dynamic system has variants that reconfigure, or are automatically selected when there is a change from the input values, such as the change of environment. As a result, this system should provide a mechanism to identify and to respond, based on the architecture design. There are many applications to handle the change of architecture needed to support a dynamic system, one of which

makes use of a Rule-Based System, which will be explained in Section 6.1.3.

In this work, the combinations of features change autonomously, where several features of the feature model bind to the environment parameters. For example, the variant features of HTTP (Web Server) bind to the workload and energy usage of the environment parameters that allow the system to adapt to the change of environment.

In this report, the product configurations include the following information to adapt its deployment:

- **Firstly**, the workload and energy usage of each virtual machine (VM), as the environment parameters, are monitored continuously. The Rule-Based System (RBS) makes decisions based on the change of the environment parameters.
- **Secondly**, the feature model is used as the reference model to reconfigure the system, where several features bind to the environment parameters as elements that adapt to the environment changes.

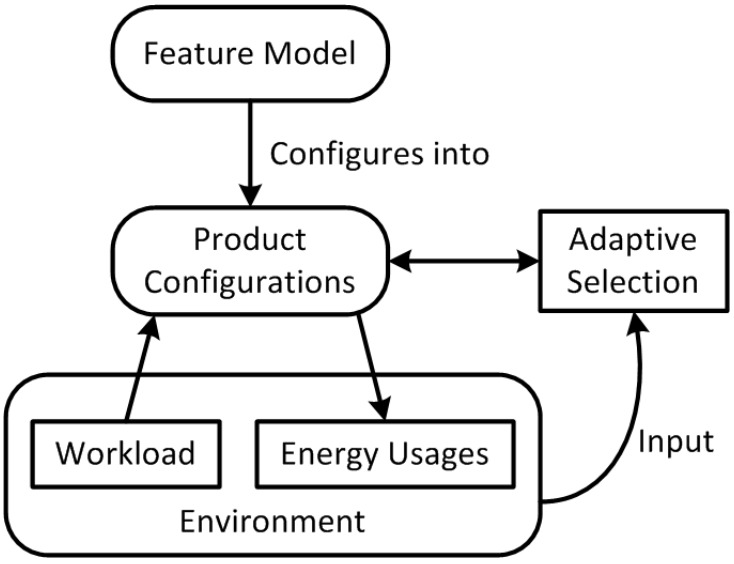


Figure 6.1: A dynamic variability that adapts to workload and energy usage.

Figure 6.1 shows the dynamic variability with workload and energy as input for an adaptive mechanism, where the feature model, as a reference model, has options to be reconfigured into different product configurations. There are two elements of the environment, energy and workload, that influence the adaptive selection of feature configurations,

where the measurement of energy and the incoming workload are used as input to decide which composition of features is suitable for the current context. Using the measurement technique explained in Chapter 3, we obtain the information of energy expenditure of the product configurations. Moreover, both the workload and energy usage are monitored continuously to support the adaptive mechanism.

6.1.2 Creating a dynamic model

This section describes our technique to create a dynamic model to accommodate the change of environment by reconfiguring the software architecture.

In a virtualised system, the software architecture is reconfigured, as depicted in Figure 6.2 (a), to cope with new requirements, such as workload and energy usages, using a feature model as the reference architecture. Another technique is to create the virtual machine instance to adapt to the change of environment by scaling the system architecture, i.e. elastic cloud, as shown in Figures 6.2 (b).

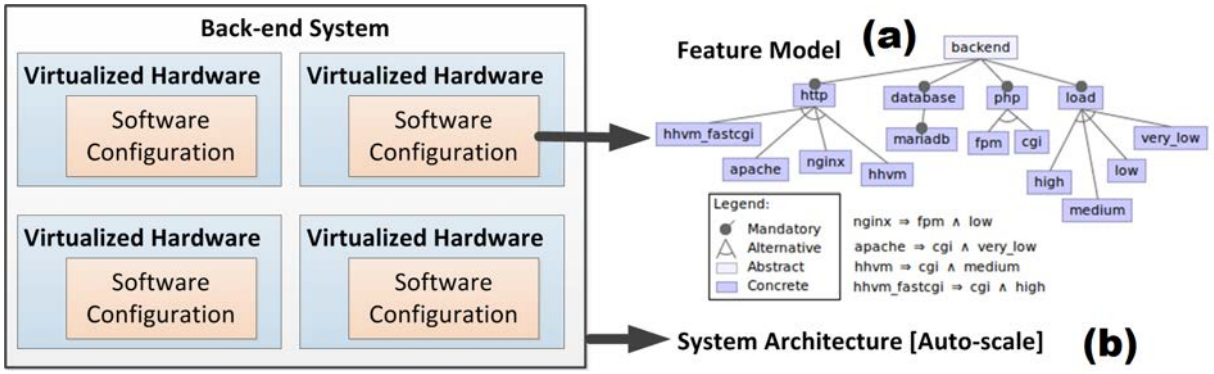


Figure 6.2: Architecture of back-end server related to (a) auto-reconfigure software architecture and (b) auto-scale infrastructure.

In our approach to reducing energy, we reconfigure the instances of a virtualised environment to select the combination of software components that consume less energy. Subsequently, to manage the artefacts or components of the system, a feature model is utilised to build a dynamic model. As shown in Figure 6.3, the feature model is attached to the software configuration of the virtual machines, where some features bind to the

runtime reconfiguration. In the runtime reconfiguration, the software configuration binds to the features in a composition based on the environment change.

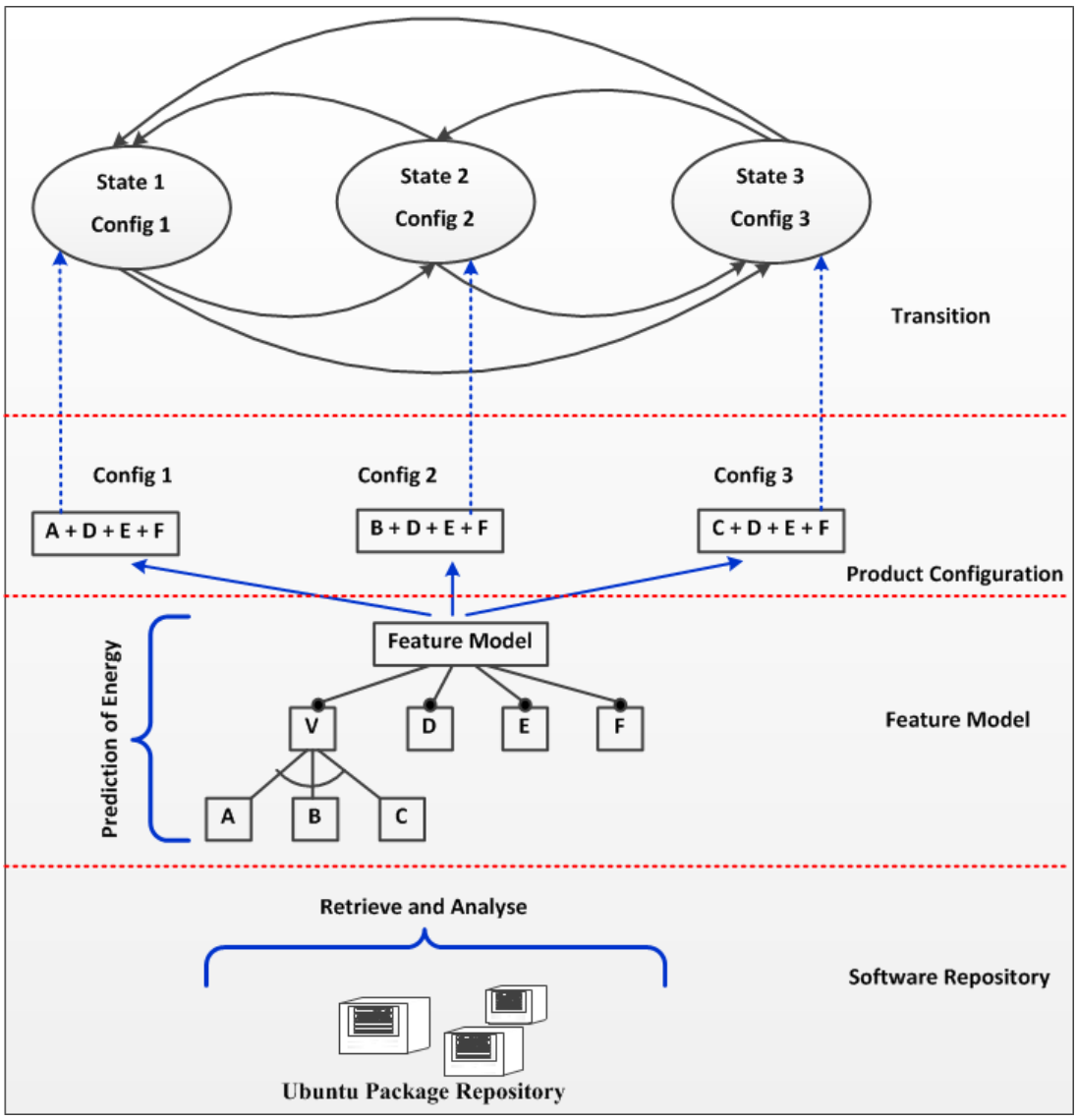


Figure 6.3: Models@runtime for auto-reconfigure of software architecture based on the change of workload and energy usage.

We adopt the models@runtime [24] to develop our Dynamic Software Product Line. As depicted in Figure 6.3, this technique has four layers that represent the stages of the process in order to build a self-adaptive system, by reconfiguring the software architecture which adapts to the change of workload and energy. The following will describe the work of each layer:

- The software repository layer provides the sources for creating the feature model by

retrieving the packages with its dependencies from the Ubuntu package repository, as explained in Chapter 4.

- The feature model layer creates a feature model from the results of the software repository layer, and provides the prediction of which combinations of software components influence the consumption of energy, as described in Section 5.

- The product configuration layer selects the combination of components that are suitable for the adaptive system, where the tools for software distribution, such as Ansible [101], are used. The combinations of components that are included into the self-adaptive system are selected based on the information from the feature model. Furthermore, the deployment of the software components into the system is run in an automatic way, based on the change of environment.

- Finally, the transition layer creates a self-adaptive system that adapts to the change of environment in order to reduce the consumption of energy. In this layer, the rule-based system manages the dynamic change of combinations of software components based on the monitoring of input data, which are the workload and the consumption of energy.

To show the work of the Dynamic Software Product Line (DSPL), a feature model is projected into a transition diagram to describe which composition of features is included in a particular state. This is accomplished by linking the features – the members of the SPL – to the product configuration, then to the state of the transition. As an example, Figure 6.3 has a mapping between a feature model, a product configuration and a transition diagram, where the feature change is related to the change of the states in the automaton. As an example, feature "A" maps to automaton "State 1" via product configuration "Config 1" to handle the change of environment.

6.1.3 The Application of a Policy-Based Approach in a Runtime System

A policy-based mechanism manages a decision to determine how to adapt to the change of environment. This mechanism creates an autonomic architecture for managing not only the workload, but also for changing the software architecture to use a configuration that consumes less energy. In addition, this mechanism also allows us to enforce a high-level policy into a running system, such as changing the requirements.

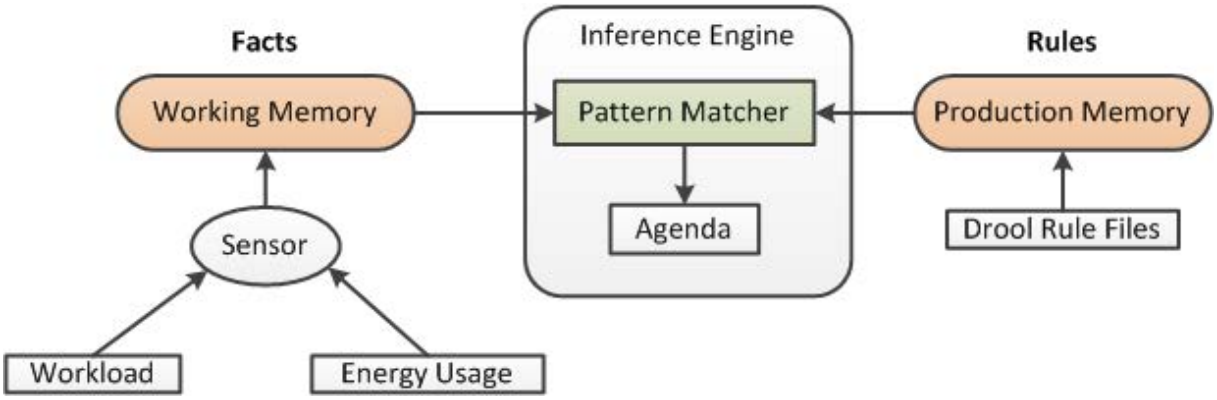


Figure 6.4: Drools Rule-Engine with workload and energy usage as facts input.

We use a Rule-Based System (RBS) to apply the management of the architecture reconfiguration, which uses a policy-based approach to manage rules – with a condition statement – to adapt to the environment changes. We use Drools [11] as the Rule-engine to manage the runtime reconfiguration that requires Working Memory (Facts) and Production Memory (Rules) as the main input, as shown in Figure 6.4. The Working Memory is short term memory, which obtains workload and energy usage as input which needs to be evaluated by the rule-engine as domain data. The Rule Base, or Production Memory, is long term memory to store rules loaded from a Drools Rule File. In this thesis, the Drools Rule File is created using our technique to measure and predict the consumption of energy in the virtualised system as "pre-knowledge" information. To execute rules based on facts, the Agenda tackles the rules sequence of execution tasks. It is worth noting that the order of rules affects the execution queues.

We use a rule-engine to make a decision about how and when to change the software configuration. The policy-based system manages the information that is coming in as input from the system monitoring process, which includes workload and energy usage as sensor data, and the duration of an event as a reference for how long this condition is occuring within the system.

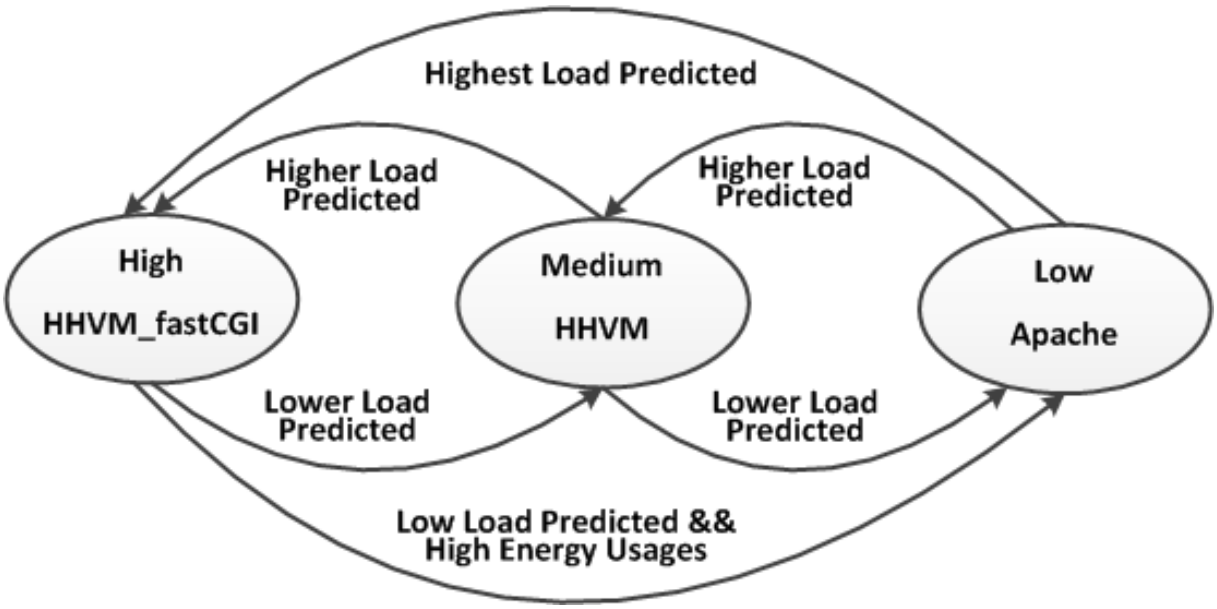


Figure 6.5: Transition of state to reconfigure the system.

The mechanism of decision and action is illustrated in the form of states within the automaton. For each state, the automaton relates to product configurations, as explained in the previous section. For example, as illustrated in Figure 6.5, the change of configuration takes place when conditions/constraints are encountered. These constraints are shown as lines to represent the change of state. The current state of the system is using "Apache" to provide services for "Low" incoming requests and uses low energy. When a "High Load" is predicted, which will consume a high amount of energy when running using the current configuration, then the rule-engine makes a decision to reconfigure the system to use "HHVM_fastcgi" to make the system cope with the Service Level Agreement (SLA).

6.1.4 Managing a Dynamic Reconfiguration

This section presents a technique to reconfigure, dynamically, the software architecture of a virtualised system, without human intervention.

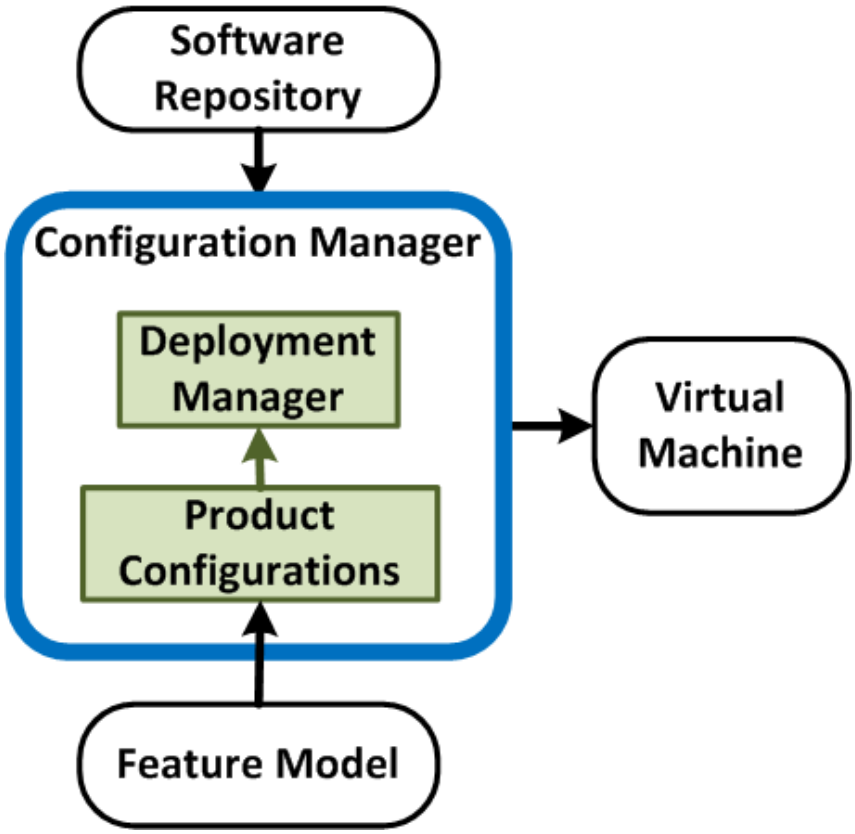


Figure 6.6: Dynamic configuration management.

In order to create a dynamic reconfiguration, we develop a mechanism that recomposes the software architecture within the virtual machine using several elements, as depicted in Figure 6.6:

- **Feature Model** creates the software model from the artefacts of feature members in the Software Repository. The feature model also acts as a reference model for the software architecture within the system.

- **Software Repository** provides all the software components or packages to be deployed in the architecture. The repository has its metadata, as explained in Chapter

2, to describe dependencies and constraints that are embedded with particular software components. An example is the Ubuntu package repository.

- **Configuration Manager** manages the transformation of feature dependencies of a feature model into component or package dependencies, which are retrieved from the software repository. In addition, before the deployment, a product configuration is checked as to whether or not any configuration violates the software configuration version or has misplaced component dependencies. The Deployment Manager subsequently executes a new line-up of software configuration by retrieving from the repository and installing or deploying the software components into the virtual machine.

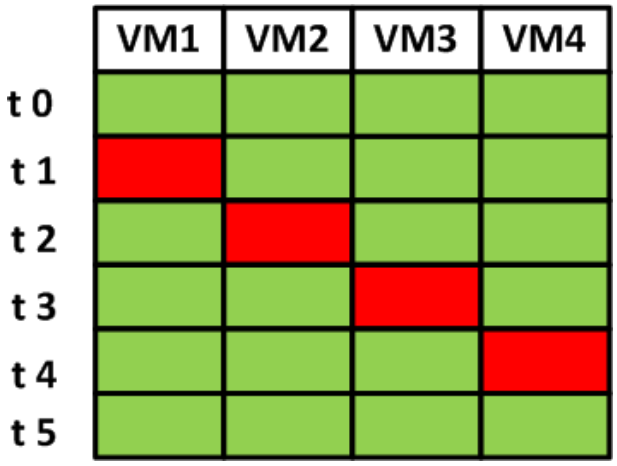


Figure 6.7: A sequential deployment by the Configuration Manager [VM = Virtual Machine, t = session].

In managing the dynamic reconfiguration, we deploy the software components in a sequence that does not interrupt the running services. This technique consumes various time and energy to deploy configurations depending on component dependencies and the virtualised environment specification. As depicted in Figure 6.7, the configuration manager disables the incoming request to the virtual machine instance being reconfigured (in red), and then enables it when finished (in green). It is worth noting that the reconfiguration can be executed in parallel across a cluster of back-end servers. For example,

when there are 100 instances of Virtual Machines, we can reconfigure 10 instances for each session.

6.2 The Application of Dynamic Software Product Line with Energy

This section presents the application of a Dynamic Software Product Line to reduce energy in a dynamic system for a virtualised environment. This application is explained as our reference architecture for a self-adaptive load-balancer architecture with a Rule-Based System. As illustrated in Figure 6.8, our architecture consists of the components as explained in the following subsections.

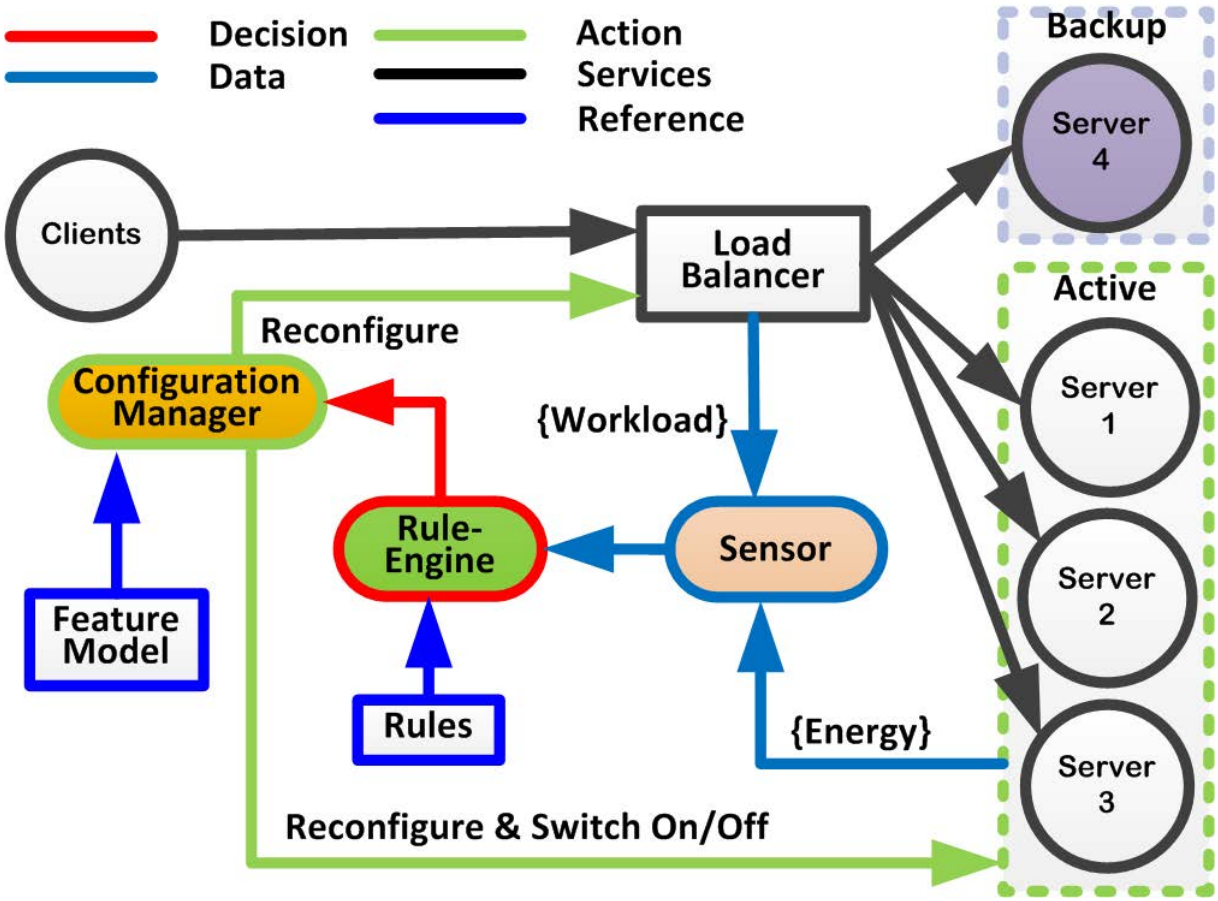


Figure 6.8: An architecture of a Self-adaptive load balancer with a Rule-Based System.

6.2.1 Components of a Self-adaptive Load-Balancer with a Rule-Based System

Load-Balancer

A load-balancer acts as a runtime manager which determines an instance of the virtual machine being disabled or enabled from the running services, and identified what the object is before making a decision over an action. The load-balancer is a key component that will address the TCP and HTTP incoming workloads. We use HAProxy [204] as a load-balancer in this work. In our reference architecture, we assume that this component has socket stream access that allows us to provide the monitoring data available to a rule-engine. For example, we can acquire the information concerning the workload for a particular back-end server and use this information as a means to reconfigure the system. Furthermore, a load-balancer also has the capability to reconfigure the reference infrastructure without any detriment to the services. One method is through enabling or disabling the network traffic session of the particular back-end server.

Back-End Servers

A back-end server is a web server that runs a web application. A web server responds to requests by executing tasks that utilize the CPU, which may then increase the total amount of energy usage, and then sends the information about the consumption of energy to the Rule-engine. In addition, the back-end servers also have options to reconfigure with a different software architecture. In this work, we build the back-end servers as a cluster of virtual machines, and the reconfiguration is based on a feature model [54], a tree-based structure describing the possible combinations of the features of a system, as explained in detail in Section 6.1.2.

Rule-Engine

The Rule-Based System (RBS) dynamically enforces high-level policies in a system. This is important for a self-adaptive system in order to avoid an interruption when a new policy/requirement needs to be implemented in a running system. By using a RBS we are addressing the design of an adaptive system for a virtualised environment that applies the system design by auto-scaling the infrastructure and accommodating the high-level policy. We use Drools [11] as an RBS, where the Drools rule-engine requires a working memory and rules as its main input. The working memory is short-term memory, which is obtained from the sensors that needs to be evaluated by the rule-engine as domain data. The rules are long term memory that is stored as a Drools Rule File that defines the running configuration, based on the input from the sensors, which acts as the working memory. The configurations change from one set of components to another using a state machine that is described as a transition mechanism. For example, states S1, S2, S3 and S4 are a set of rules, and current running configuration is S1. If the workload and energy values match the conditions within the Rule-engine, then the state changes to S2.

We adopted the models@runtime [24] approach to abstract the rules that are implemented using a runtime model, which model an autonomous software architecture reconfiguration in a self-adaptive system. Furthermore, the runtime model state transition is based on a feature model and the user requirements, where the architecture reconfiguration is established in three dimensions: energy usage, incoming traffic requests and the current software architecture configuration; it is also supported by the load-balancer traffic information and the ACPI. In addition, our architecture uses a Linear Model [157] to predict when the architecture should be reconfigured and to which state, based on the Sensor information.

Sensor

The Sensor component continuously monitors the workload of the load-balancer and its energy usage with respect to the back-end servers. This monitoring supports the rule-engine to make a decision, to help the system become more adaptive to the change of environment by providing the measurement data of the current running system. Such a component uses a message broker software, like RabbitMQ [28], to transfer workload and energy usage data as messages from the load-balancer and the back-end servers to the rule-engine. Using a queueing system, all messages are managed as a stack of queue data transactions.

Configuration Manager

This component configures the load balancer and back-end servers to reduce energy usage, without interrupting the running system. Configuration management tools, the Ansible configuration manager [101], deploy and update the software architecture within the back-end servers and reconfigure the load balancer. To reconfigure the software architecture, a feature model [54] is used as the model of composition that allows us to have options for combinations of software components to be included into the software architecture. A feature model consists of a tree hierarchy and cross-tree relationship to manage constraints within a model. A great advantage of using such a model is the capacity to reason over the possible configurations of a system and to avoid possibly faulty combinations from the use of constraints.

Furthermore, using a sequence of procedures, the configuration manager provides an uninterrupted service while reconfiguring the architecture. The reconfiguration process uses the software packages that are provided by the external system, e.g., the Ubuntu package repository.

6.2.2 The Mechanism of a Self-adaptive Load Balancer with a Rule-based System

This subsection explains the mechanism of the self-adaptive load-balancer with a rule-based system, as depicted in Figure 6.8.

We assume that the virtual machine instances are running on one CPU, 512 MegaByte of memory and 10 GigaByte of storage. The steps are as follows:

1. The load-balancer checks the health of each back-end server to find out which one is ready to receive requests, then it allocates requests to the servers.

2. The sensor monitors the workload of the load-balancer and energy usage of each back-end server, continuously. Then the data stream is sent, continuously, to the rule-engine as a stack queue transaction using the middleware system (i.e. RabbitMQ [28]).

3. The rule-engine examines the information from the sensor. When a condition matches the rules, the rule-engine's decision is sent to the configuration manager.

4. In response to the rule-engine's decision, the configuration manager assesses the valid combinations of features. This is accomplished by renewing the current feature model and its constraints. After that, the configuration manager retrieves the software components from the software repository. The configuration manager then deploys the components into the back-end servers using a sequential action, which is managed by the load-balancer, to maintain the current running system to provide the services.

5. Repeat the steps above.

6.3 Case Study: Energy Management of a Self-Adaptive Load-Balancer with a Rule-Based System

In this section, we conduct an experiment to evaluate our approach to the Dynamic Software Product Line (DSPL) with energy. This experiment builds a self-adaptive load-balancer with a Rule-Based System (RBS) to reduce energy in a virtualised system.

In order to evaluate our approach, we implement a self-adaptive and energy-aware system that follows our proposed architecture, as explained in the previous section. We then test the system against changes in the workload, providing evidence of its applicability and efficiency. The evaluation, as depicted in Figure 6.9, begins with setting up the environment of the experiment and system. We then perform an experiment to obtain information on energy usage as pre-knowledge. After that, we create rules for the rule-engine as high-level policies, guided by the pre-knowledge. Subsequently, we run the system, increasing the workload from 60 to 600 users/minutes, to simulate the incoming requests. Hence, we obtain the median for each minute of measurement as energy values. Finally, we compare our results with a simplified elastic approach.

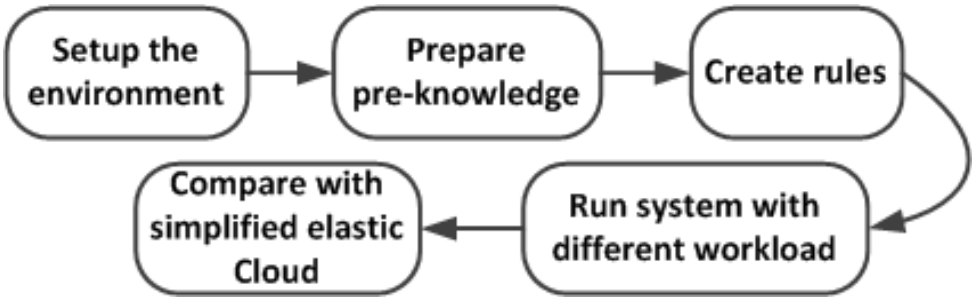


Figure 6.9: Flow of the Evaluation

6.3.1 Implementation of the Dynamic System

Before actually running the experiment, we need to implement the system based on our reference architecture. The implementation is described in the following subsections.

6.3.1.1 High-Level Policies

The self-adaptive system needs to cope with high-level policies that are defined by non-technical users or stakeholders at the managerial level of Cloud providers. Part of these policies can be expressed in the form of a feature model, as illustrated in Figure 6.10.

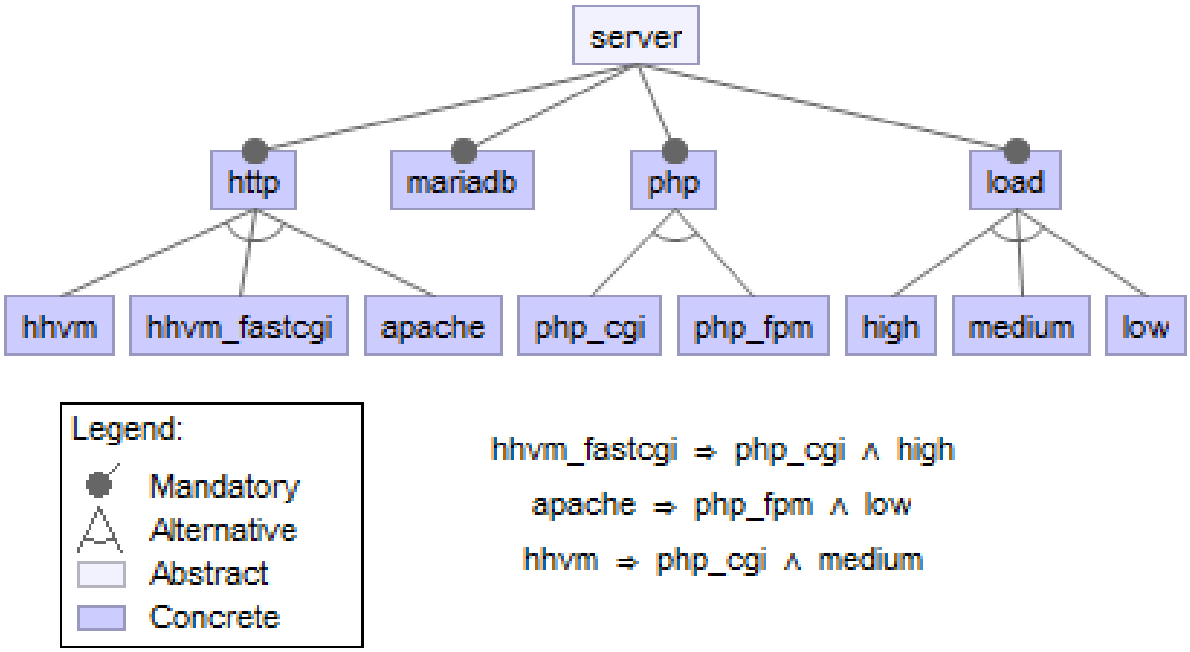


Figure 6.10: Feature Model of Simple Back-end Server

In our approach, the prior knowledge is the main source for these high-level policies, similar to the prediction of energy usage. In this implementation, the back-end servers are created from the Debian/Ubuntu package repository. Such a web application consists of the HTTP server and Database Server. Using our measurement method from previous work [161], we could verify that these servers consume a different amount of energy, using a similar workload arriving to the servers. As depicted in Figure 6.10, the feature model has three variants, which are "HTTP", "PHP", and "load". We can calculate, with the help of satisfiability solvers, the possible configurations of the system: ("Apache"-"MariaDB"-"PHP-FPM"-'Low"), ("HHVM"-"MariaDB"-"PHP-CGI"-"Medium") and ("HHVM_fastcgi"-"MariaDB"-"PHP-CGI"-"High").

In this self-adaptive system, the back-end server is reconfigured without any human

intervention. During a change in the configuration process, all traffic is rerouted by the load-balancer using runtime configuration techniques [204], so the change of software configurations will not be detrimental to any incoming traffic

6.3.1.2 Creation of Rules from Transition Diagram

Figure 6.5 shows the implementation of the runtime model following the models@runtime paradigm. Based on the energy predictions from our previous work [162], we can define the following runtime states:

High Workload: when the incoming workload has increased to more than 300 requests for more than 20 minutes, the back-end server is reconfigured with "HHVM_fastcgi" and "PHP-CGI". This option will reduce the amount of energy consumed by the system.

Medium Workload: when the back-end servers received a workload between 240 and 300 requests, and the consumption of energy is more than 18 Watts, the software of the back-end server is configured using "HHVM" and "PHP-CGI", which improves the energy usage.

Low Workload: when the energy usage is more than 18 Watts and the workload is less than 240 requests, the server's software architecture reconfigures to have "Apache" and "PHP-FPM".

6.3.2 Experimental Setup

In this subsection, we describe the requirements of an experiment on a load-balancer with a rule-based system in a virtualised environment.

6.3.2.1 Experiment Setup

We setup the experiment environment using HAProxy as the load balancer and HTTP servers as back-end with "Wordpress" installed in each back-end in the Virtualbox [219] using Ubuntu 14.04_x64 images. The computer host is a laptop HP Elitebook 8440p with an i5 processor and 4 GigaByte of memory running Ubuntu 14.04_x64, and using Vagrant

[56] as the virtual environment manager. Each virtual machine is provisioned with 1 CPU, 512 MB of memory and 10 GB of storage running Ubuntu 14.04.

In our experiment, the load-balancer, the configuration manager and the back-end servers are virtual machines in the same network to simulate the Cloud environment. The rule-engine obtains information from the load-balancer and back-end servers, and sends a message using a *work queue*. For example, when the energy consumption of the back-end servers is high and the incoming workload is low, the rule-engine sends a message to the configuration manager to reconfigure the back-end with a different HTTP server. The configuration manager then executes the process by using a sequence procedure that disables the back-end being configured and enables it when the processes are complete.

We built the load balancer in HTTP mode to serve three active back-ends with one back-up server. We use *socat*, a Linux utility, to relay bidirectional data transfer, which is able to listen on UNIX sockets to monitor the environment of the load balancer, such as the workload of the back-end servers. To monitor current energy usage in a web server, we captured the information regarding the power usage of battery electric drain. The information from monitoring results is sent to the rule-engine every 5 minutes. We used *cron*, a Linux time-based scheduler, to create a schedule of the data transmission between machines. The RabbitMQ [28] middleware is used to create communication between the load balancer, back-end servers, the configuration manager and the rule-based engine, using queue messaging.

6.3.2.2 Measurement of Energy

We measured the energy usage of four combinations of HTTP servers, database management server and web middleware. The combination of web application servers is explained in Section 6.3.1.1. All web applications are configured with "MariaDB" and "Wordpress" in Ubuntu 14.04. As depicted in Figure 6.11, for the 120 users accessing the system under examination, the configuration with "Apache" consumed the highest amount of energy and "HHVM_fastCGI" the lowest. When the number of users increased to more than

240 users per second, the configuration with "Apache" consumed the highest amount of energy.

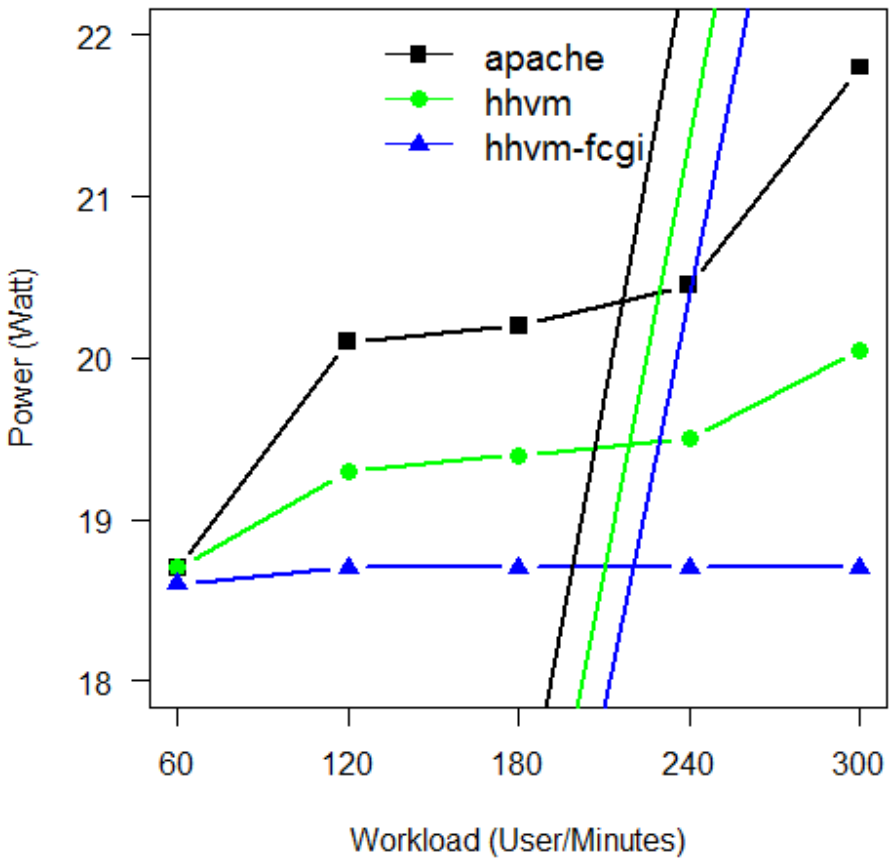


Figure 6.11: Energy Usages for Web Application with combination of Optional HTTP ("Apache", "HHVM" and "HHVM-FastCGI") and optional PHP ("PHP-CGI" and "PHP-FPM") - with prediction of energy usage associated with workload [Solid lines with different colours show the prediction of energy usages.]

This measurement is performed using the method in our previous work [161], which uses workload to execute tasks within the system under examination, operating a virtual machine with hardware specification of 1 CPU, 512 MB memory and 10 GB storage. When measuring the energy usage using a different hardware specification the results might be different. In addition, the result of this measurement is used as the prior knowledge for the Rule-Based System.

In our experiment, we reconfigure the back-end servers at runtime, where the package to install is retrieved directly from the Ubuntu package repository. During the reconfiguration process, we send a workload to the system continuously every 300 users/minute. The reconfiguration steps are as follows: firstly, the load-balancer disables services to a particular back-end, secondly, when there is no traffic activity, the configuration manager executes the configuration process, thirdly, the load-balancer enables services to a particular back-end to receive the incoming workloads.

The configuration process takes different amounts of time, which also leads to a distinctive amount of energy. The time of configuration is measured based on "real" wall clock time, "user" is the amount of CPU time spent in user mode within the process or the actual CPU time used in executing processes, and "sys" is the amount of CPU time spent in the kernel within the processes. As shown in Table 6.1, "HHVM_fastCGI" spent the shortest real time and HHVM had the longest amount of time. In the user mode and kernel process, "Apache2" spent the least time.

Table 6.1: Time transition for HTTP configuration (in Seconds)

Configuration	real	user	sys
Apache2	24,353	1,604	2,016
HHVM	35,185	2,376	3,165
HHVM_fastCGI	3,548	2,239	3,378

We also experimented on the reconfiguration of web applications. This procedure is similar to the HTTP reconfiguration; we performed "Wordpress" basic packages and "Wordpress" with additional three plugins, with the average workload of 300 users/minute. As shown in Table 6.2, the "real" time spent with the basic configuration is lower than the advanced one. It is noteworthy that in our measurement, we capture energy usage for each second, so that, for 60 seconds, we produce 60 results. Then we use the median data as the value.

Table 6.2: Time transition for Wordpress configuration (in Seconds)

Configuration	real	user	sys
Basic	45,377	3,090	4,023
Advanced	318,8	4,994	6,516

6.3.2.3 Rule-Based

In this experiment, we run two scenarios. In the first scenario, we send a workload to a load-balancer in order to reconfigure the software architecture at runtime. We varied the workload in the following fashion: 60, 120, 180, 240, 300, 360, 420, 480, 540 and 600 users/minutes, taking 5 minutes for each load. When workloads arrive, the load-balancer should respond based on a decision from the rule-engine. In the second scenario, workloads trigger the rule engine to switch-on/off or to enable the hot backup of the back-end server. The process to switch-off the back-end server is started by disabling the server by reconfiguring the load-balancer at runtime. Then we switch off the virtual machine when there is no transaction according to the Server-side Session (SS) information. We assume that the current configuration is a load-balancer running with three active back-end servers and one back-up server. To allow these two scenarios, the rules are defined as follows:

Rule Reconfigure Back-end: is activated when the drainage of electric power is more than 18 Watts, with a real-time incoming workload of less than 120 requests. Then the load-balancer reroutes the traffic of the virtual machine. After that, the web application is reconfigured to have "Apache" as the HTTP server and activates the routing to the particular virtual machine.

Rule Switch-Off: is activated when the load-balancer receives the incoming workload of less than 20 requests for more than 30 minutes, rerouting the traffic and shutting down the back-end servers to have only two servers left running and one as hot back-up. The shut down command is sent when there are no more HTTP transactions within a particular VM.

Table 6.3: Transition of back-end software configuration.

HTTP server	Wordpress Instalation	Database
apache	basic	mariadb
hhvm	basic	mariadb
hhvm_fcgi	basic	mariadb

Rule Switch-On: is activated when the load-balancer receives a workload of more than 100K requests for more than 20 minutes, enabling a hot back-up server and switching on another back-up server.

As we use the HAProxy [204] as the load-balancer, clients connected to the disabled back-end server will not be affected by this condition, as long as the Server-side Session is not disrupted or disconnected.

Figure 6.12 shows the energy usage transition of back-end configuration from one HTTP server to another, where the transition to "HHVM" consumed the highest amount of energy. Each transition consumed a different amount of energy and time. In doing so, the requirement can be set up to have optimal services by modifying the rules. Table 6.3 shows the reconfigurations of back-end servers, where the transitions are serialised to have uninterrupted services.

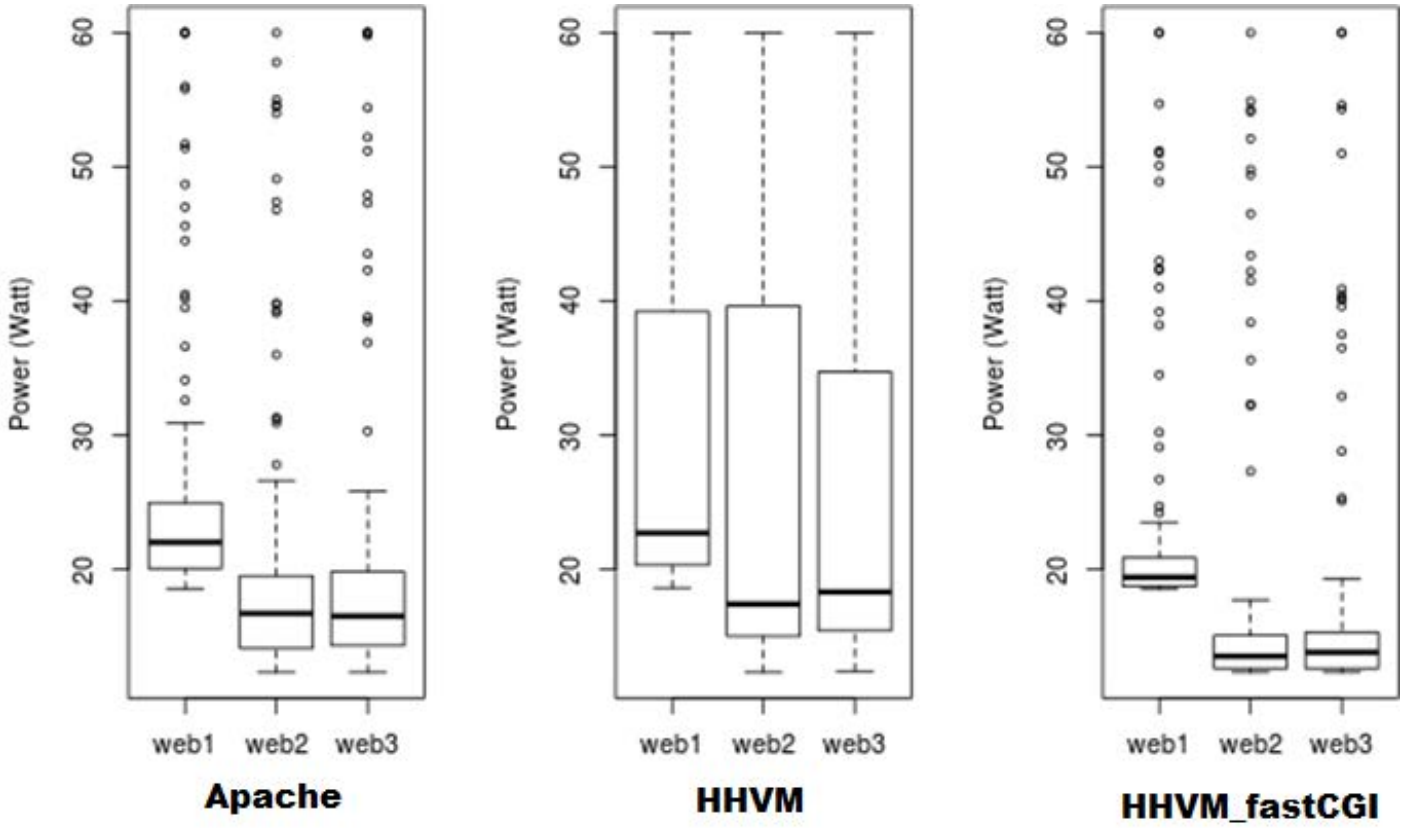


Figure 6.12: Boxplot of energy usage for software architecture reconfiguration [web1–web3 = back-end servers, details in Table 6.3]

Figure 6.13: Comparison of elastic Cloud and models@runtime energy expenditure.

6.3.2.4 Comparison of Our Dynamic System to a Simplified Elasticity

We compare the energy consumption between a simplified elasticity and our architecture using models@runtime techniques. We aim to find which system consumes less energy corresponding to workload and infrastructure. It is important to note that there are many elasticity approaches for a virtualised system. This experiment uses a simple elasticity to add virtual machines when a certain value of the incoming workload matches a rule. In the simplified elastic system, we set up the rules of the rule-engine to add a back-end server when the workload reaches 300 and 540 users/minute. In our autonomous system using models@runtime, the software architecture reconfigures to have "HHVM" when the workload reaches 300 users/minute and "HHVM_fastcgi" on 540 users/minute.

Figure 6.13 shows the comparison. At the beginning, both systems are running with 2 back-end servers configured with "apache2" as the HTTP server. The models@runtime uses only 2 back-end servers, and the elastic system adds 2 more nodes during this experiment.

When the workload reaches 240 users/minute, both techniques consume more than 16 Watts. The simplified elastic system then adds a node to the back-end server, thus reducing the consumption of energy to less than 16 Watts. Our system also reconfigures to have

"HHVM" as the HTTP server, so it slightly reduces the energy consumption. However, the simplified elastic system increases its energy expenditure to 17 Watts when the workload is increased to 360 users/minute. When the energy rises above 18 Watts, the simplified elastic system adds one node server that slightly reduces the energy. After that, when the workload increases to 600 users/minute, the energy usage of the simplified elastic system also increases to more than 19 Watts. In our system, when requests reach 480 user/minutes, the back-end consumes more than 16 Watts. This action reduces a large amount of energy expenditure into less than 14 Watts with requests of 540 users/minute, fewer than the simplified elastic system technique.

Our approach proves 11.68% more efficient after 360 users/minute, as illustrated in Figure 6.13. Moreover, when the requests increase to 540 users/minute the energy efficiency increases to 46%, after the elastic added one more virtual machine, to total four instances, and our self-adaptive system reconfigured to "HHVM_fastcgi".

6.4 Chapter Summary

In this chapter, a Dynamic Software Product Line (DSPL) with energy is introduced. This approach uses the Rule-Based System to manage the nature, and timing of a software architecture based on the change of environment. To present the self-adaptive mechanism, we monitor the workload and energy usage as sensors, as depicted in Section 6.2.1. These monitoring results are continuously sent to the rule-engine to make a decision on reconfiguration action. This chapter has also proposed an application of a Dynamic Software Product Line using a self-adaptive load-balancer with a Rule-Based System, as explained in Section 6.1.3. In Section 6.3 we performed an experiment, based on our proposed approach, that showed a good result compared to a simplified elastic approach. Overall, our technique has offered an alternative approach to reduce energy consumption, by reconfiguring the software architecture within instances of the virtualised system, autonomously, without auto-scaling the infrastructure.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

This chapter concludes the work presented in this thesis. Section 7.1 presents the summary of this thesis. Section 7.2 discusses the future work that could be undertaken to extend and improve upon this research.

7.1 Thesis Summary

This thesis presents a software-based approach in order to reduce energy within the virtualised environments. In particular, the measurement of energy from the combination of software components that are retrieved from a large software repository, an approximation technique to find the combinations that consumes low energy, and a dynamic system that uses less energy, are prepared for the purpose of analysis. In order to approximate the energy usage from combinations of components, the proposed approach uses machine learning algorithms to build models of energy consumption.

More precisely, the first approach uses the Software Product Line Engineering (SPL) to develop the product configurations with energy usage, allowing the developer to select which configurations consume less energy suitable for their needs. This technique uses the machine learning ensemble to find the combination of software components that influence the consumption of energy. The second approach uses the Dynamic Software Product Line (DSPL) to create a self-adaptive system with energy. This system reconfigures the software architecture to reduce the consumption of energy based on current running

information, concerning workload and energy usage. The feasibility of these approaches was demonstrated through case studies. We believe both approaches used in this thesis, to find the combination of features within a set of product configurations and the autonomous runtime configuration that consumed less energy, can be generalised and applied in the domain of software product line engineering. Furthermore, to clarify our approach, the thesis presents a comparison study of energy measurement within different virtualised environment infrastructures.

In Chapter 2, the thesis presents an overview of the essential material, introducing Cloud computing, including its models of service. An introduction of energy measurement, in particular the measurement of energy in virtualised environments, is then presented. This is followed by a review of existing works on the measurement of energy and power models for a virtualised system. This is continued by an introduction to the software product line and the dynamic software product line, which includes reviews of existing research on energy in product configurations. Next, an introduction to large software repositories, in particular the Ubuntu/Debian package repository, is presented, followed by an introduction to machine learning, including the machine learning ensemble. Finally, related work on the Software Product Line, that uses machine learning as a tool to achieve its goals, is reviewed.

Chapter 3 describes a technique to measure energy usage in a virtualised environment, which is then used to provide the values of energy consumption for the Software Product Line and Dynamic Software Product Line approaches. This chapter uses a workload scenario to simulate users accessing the system under examination, including sending the workload using network traffic. Moreover, Chapter 3 provides the mechanism for the measurement of energy that is used for most of this thesis where a set of mechanisms to conduct a measurement was defined. This mechanism is described with the help of a running example.

Chapter 4 presents a method to create a feature model from a large software repository automatically, by retrieving and transforming the components within a repository into

a feature model. The components retrieved are transformed into a tree-hierarchy and cross-tree constraints using a graph approach. The transformation of components from a repository is described in two stages - creating a graph for component dependencies, and transforming package dependencies into constraints for a feature model. To conduct the transformation, a set of rules are defined in the second stage that is described with the help of a running example.

Chapter 5 proposes a Software Product Line with energy to find which features consumed less energy in a set of product configurations. This chapter begins with a few combinations of features where the product configuration and its energy usage represents a data set for the machine learning process. This chapter also suggests the machine learning approach to predict which combinations of features would consume less energy in a larger feature model, because it is infeasible to measure energy usage for all possible configurations. After that, we introduce the Energy Prediction Trees (EPT), a combination of a Linear Model and a Regression Trees model, in order to enhance the prediction accuracy for SPL with energy. In order to demonstrate the feasibility of our approach, this chapter discusses how the running example can be modelled in a combination of machine learning and Software Product Line approaches to find a set of features that consume low energy. To evaluate our approach, we built a case study using feature models that are retrieved from the Ubuntu package repository.

Chapter 6 describes a Dynamic Software Product Line (DSPL) with energy for a virtualised system. Moreover, this chapter introduces an autonomous technique using a runtime model that focuses on the reduction of energy by reconfiguring a software architecture. To build an autonomous model, a set of rules is defined from a simple prediction of energy. These are used by the rule-engine to decide whether a cluster of virtual machines should be reconfigured, based on input from real-time measurement.

As an overall review, this thesis contributes, firstly, a technique to measure energy in virtualised systems. This technique measures the consumption of energy using a workload to execute tasks in a system. Furthermore, for each unique combination of software

components execute different tasks. We use a scenario of the workload to simulate a real environment, such as users accessing an online shopping system, for the purpose of executing a combination of software components in the system. In this technique, we measure the combination of components that chiefly run the web services in the Linux environment. Secondly, the creation of a feature model from a software repository where, as in a typical large software repositories such as the Ubuntu package repository, it has a high number of package dependencies. We propose a technique to transform package dependencies into a constraint of a feature model with the help of graph approaches. This technique is also able to resize the number of components, in order to create a feature model, by limiting the dependencies to exclude the conflict packages and reduce the depth of dependencies with the help of the Debian/Ubuntu utility, Debtree. Our technique only covers the Ubuntu package repository; to implement on different repositories further research might be needed. Thirdly, a technique to predict which combinations of features consumes low level of energy. We extend the Software Product Line (SPL) engineering with energy usage. This technique predicts the consumption of energy in order to search for combinations of features that consume low energy. This prediction technique builds their data set for the machine learning process from a combination of workload, configuration of features that are retrieved from a large software repository and the CPU power usage. A comparative study of several machine learning approaches is performed in order to find which algorithms perform well with this data set. In our experiment, we found that the noise of data, which we speculate is caused by the first workload arriving at the system under measurement utilised the CPU to its maximum capacity, influenced the prediction results. Fourthly, an approach using a dynamic system to reduce energy usage within a virtualised environment. We extend the Dynamic Software Product Line (DSPL) engineering to adapt to the change of environment by reconfiguring the features that consume low energy. The models@runtime is adopted to integrate the Software Product Line (SPL) model to the dynamic elements, such as energy usage and workload. This approach creates a relationship between the features of a feature model and a rule-

based system to manage the combination of features that should be included and the timing of the execution of the runtime reconfiguration based on the change of workload and the consumption of energy. These techniques contribute towards two product line approaches - the Software Product Line engineering and the Dynamic Software Product Line engineering - that aim to reduce energy in virtualised systems.

7.2 Future Work

The management of a Software Product Line (SPL) and a Dynamic Software Product Line (DSPL) is an essential issue, especially with evolving feature dependencies and new requirements in conjunction with the requirement for the reduction of energy. The SPL model in this research, as presented in Chapter 5, used combinations of energy measurements, the feature model and the machine learning ensemble to provide options for configurations of features that consumes less energy. This approach is suitable for a design delays architecture, such as an SPL, where the training data for prediction uses a batch or offline setting.

In a dynamic system, large volumes of training data (i.e. workloads and energy usage) become available sequentially. To deal with a large volume sequential data set, our technique can be extended to support the DSPL using an online learning ensemble [170], an online version of the machine learning ensemble. Furthermore, in this learning ensemble, the learning algorithm processes each training instance on-arrival without saving or reprocessing all the training instances [171] that give benefit for data that arrives continuously (streaming data) [22]. Streaming data may arrive at such a high speed that the algorithms must process them under a very strict constraint of time and space. As a result, we face several challenges, which include resources of the computation unit and the change of volumes of data.

In relation to our work in Chapter 6, as shown in Figure 7.1, the online learning ensemble receives the data stream from the sensor, then feeds into the rule-engine in

order to make a decision over the change of environment. In this self-adaptive system, the Rule-engine sends instructions to the configuration manager as to whether the system needs to be reconfigured, based on the online prediction. Therefore, the ensemble diversity affects accuracy in a changing environment [156], where the old prediction results help the learning of the new prediction.

Plans are also being drawn up to augment the robustness of models@runtime to support a reduction of energy for a self-adaptive system in a virtualised system [40, 5, 20], where the existing work does not support the online (real-time) approximation. Initial work revealed that the online approximation might create an adaptivity of a system in a more flexible way. Time and context are the essential components to make a decision in a self-adaptive system. Furthermore, because features are configured from a large software repository, such as the Ubuntu package repository, a delay of configuration should be avoidable - in the case of a good network connection.

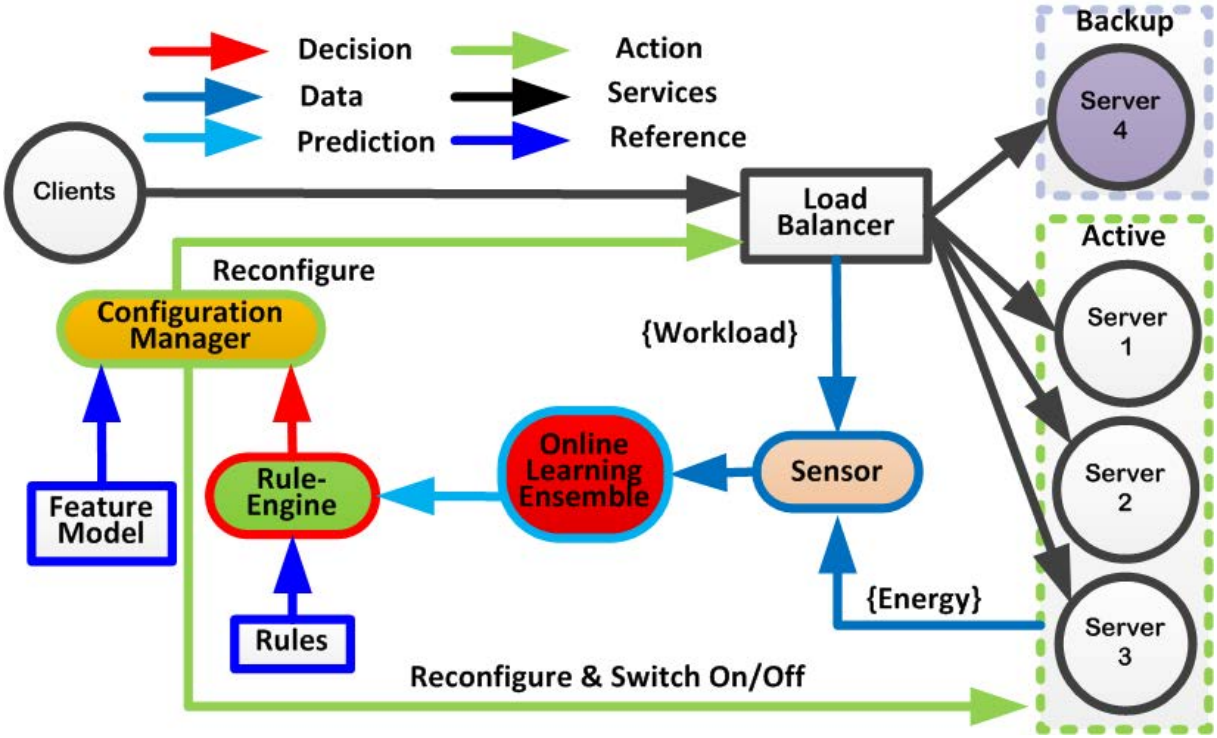


Figure 7.1: A self-adaptive load-balancer with online learning ensemble.

An automatic configuration that adapts to the change of component dependencies (i.e. components evolution) from a large software repository is a crucial technique. Such

a technique supports the back-end of a self-adaptive system, as shown in Figure 7.1, to reconfigure the software architecture automatically . In creating product configurations, time and speed are the essential constraints; thus any on-the-fly configurations issues must be addressed before this framework deploys a running system.

Another technique to address the time and speed with automatic (on-the-fly) reconfiguration from a large repository is by applying a prediction on software configurations. The online machine learning ensemble has the vision and the ability to predict all valid software configurations available in the repository - in a real-time manner. Therefore, one area of research would be the development of an automatically generated valid product configuration from one or more large software repositories for real-time systems, such as a load-balancer for a virtualised environment.

In this thesis, we measure the consumption of energy, mostly, in web services system such as Wordpress using a workload in order to execute tasks within the system. We also draw a plan to measure energy in a different domain such as security system to find out the relationship between energy usage and methods of security. Furthermore, we also plan to do a deep study on the noise within the data of our measurement of energy in the direction of finding the symptoms that trigger such a problem.

The software repository evolves over a time line that may change or create new component dependencies. We speculate that the repository evolution creates a defect to the product configurations for an existing SPL. To address this, a transition architecture design should be developed, in order to avoid the failure of the existing system. We envisage that graph approaches, such as Formal Concept Analysis (FCA) [65] and numerical FCA (nFCA) [142], provide a conceivable solution. Moreover, the features (components) dependencies can be analysed using the FCA [186, 66, 140] from all possible product configurations within the SPL model. The online learning ensemble makes a prediction of the possible product configurations that correspond to the change of environment.

APPENDIX A

REGRESSION TREE

```
Whole RT:
REPTree
============

Workload < 45.5
|   Workload < 25.5
|   |   Workload < 1.5 : 22.27 (4/35.86) [2/36.49]
|   |   Workload >= 1.5
|   |   |   Workload < 14.5
|   |   |   |   Workload < 8.5
|   |   |   |   |   Workload < 5.5
|   |   |   |   |   |   c < 0.5 : 5.62 (10/0.09) [2/0.15]
|   |   |   |   |   |   c >= 0.5
|   |   |   |   |   |   |   Workload < 3.5 : 5.96 (5/0.09) [1/0.01]
|   |   |   |   |   |   |   Workload >= 3.5 : 6.25 (6/0.01) [0/0]
|   |   |   |   |   Workload >= 5.5 : 6.62 (14/0.15) [4/0.05]
|   |   |   |   Workload >= 8.5
|   |   |   |   |   Workload < 11.5 : 7.52 (11/0.08) [7/0.88]
|   |   |   |   |   Workload >= 11.5
|   |   |   |   |   |   c < 0.5 : 8.14 (6/0.03) [3/0.09]
|   |   |   |   |   |   c >= 0.5
|   |   |   |   |   |   |   Workload < 12.5 : 8.17 (2/0.01) [1/0.03]
|   |   |   |   |   |   |   Workload >= 12.5 : 8.94 (4/0.05)
    [2/0.27]
|   |   |   Workload >= 14.5
|   |   |   |   Workload < 17.5 : 9.35 (14/1.77) [4/0.12]
|   |   |   |   Workload >= 17.5
|   |   |   |   |   Workload < 21.5
|   |   |   |   |   |   c < 0.5 : 10.16 (8/0.07) [4/0.09]
|   |   |   |   |   |   c >= 0.5 : 10.58 (7/0.04) [5/0.09]
|   |   |   |   |   Workload >= 21.5
|   |   |   |   |   |   c < 0.5 : 10.99 (8/0.09) [4/0.08]
|   |   |   |   |   |   c >= 0.5 : 11.7 (6/0.12) [6/0.68]
|   Workload >= 25.5
|   |   Workload < 35.5
|   |   |   Workload < 28.5
|   |   |   |   a < 0.5
|   |   |   |   |   Workload < 26.5 : 12.26 (2/0) [2/0.28]
|   |   |   |   |   Workload >= 26.5 : 13.07 (6/0.18) [2/0.24]
|   |   |   |   a >= 0.5 : 11.91 (4/0.17) [2/0.14]
|   |   |   Workload >= 28.5
|   |   |   |   f < 0.5
```

```
|   |   |   |   |   Workload < 33.5
|   |   |   |   |   |   Workload < 29.5 : 12.55 (2/0.18) [3/0.11]
|   |   |   |   |   |   Workload >= 29.5
|   |   |   |   |   |   |   b < 0.5
|   |   |   |   |   |   |   |   Workload < 32.5
|   |   |   |   |   |   |   |   |   Workload < 30.5 : 15.57 (3/0.2) [1/45.88]
|   |   |   |   |   |   |   |   |   Workload >= 30.5 : 13.4 (5/0.05) [3/0.1]
|   |   |   |   |   |   |   |   Workload >= 32.5 : 13.85 (4/0.35) [0/0]
|   |   |   |   |   |   |   b >= 0.5 : 13.06 (3/0.07) [1/0.06]
|   |   |   |   |   Workload >= 33.5 : 14.26 (5/0.1) [5/0.6]
|   |   |   |   f >= 0.5 : 14.44 (4/2.86) [3/1.94]
|   |   Workload >= 35.5
|   |   |   Workload < 39.5 : 15.09 (16/0.47) [8/0.17]
|   |   |   Workload >= 39.5
|   |   |   |   Workload < 42.5 : 15.96 (13/1.14) [5/0.32]
|   |   |   |   Workload >= 42.5 : 16.86 (10/0.16) [8/0.36]
Workload >= 45.5
|   Workload < 76.5
|   |   Workload < 61.5
|   |   |   Workload < 53.5 : 18.71 (33/2.51) [15/1.23]
|   |   |   Workload >= 53.5
|   |   |   |   c < 0.5
|   |   |   |   |   Workload < 58.5
|   |   |   |   |   |   e < 0.5 : 20.84 (2/0.1) [3/6.34]
|   |   |   |   |   |   e >= 0.5 : 19.46 (6/0.03) [4/0.56]
|   |   |   |   |   Workload >= 58.5 : 20.61 (5/0.04) [4/0.03]
|   |   |   |   c >= 0.5
|   |   |   |   |   Workload < 54.5 : 21.59 (2/3.76) [1/1.25]
|   |   |   |   |   Workload >= 54.5
|   |   |   |   |   |   Workload < 56.5 : 20.48 (4/0.21) [2/0.44]
|   |   |   |   |   |   Workload >= 56.5
|   |   |   |   |   |   |   Workload < 59.5 : 20.97 (5/0.02) [4/0.08]
|   |   |   |   |   |   |   Workload >= 59.5 : 21.62 (3/0.03) [3/0.1]
|   |   Workload >= 61.5
|   |   |   Workload < 67.5
|   |   |   |   Workload < 63.5
|   |   |   |   |   c < 0.5 : 21.12 (3/0.08) [3/0.07]
|   |   |   |   |   c >= 0.5 : 21.89 (4/0.13) [2/0.31]
|   |   |   |   Workload >= 63.5
|   |   |   |   |   c < 0.5 : 21.93 (9/0.02) [3/0.44]
|   |   |   |   |   c >= 0.5 : 22.69 (8/0.24) [4/0.36]
|   |   |   Workload >= 67.5
|   |   |   |   c < 0.5
|   |   |   |   |   Workload < 73.5
|   |   |   |   |   |   Workload < 69.5 : 22.35 (4/0.06) [2/0.01]
|   |   |   |   |   |   Workload >= 69.5 : 22.81 (7/0.05) [5/0.07]
|   |   |   |   |   Workload >= 73.5 : 23.57 (7/0.09) [2/0.08]
|   |   |   |   c >= 0.5
|   |   |   |   |   Workload < 71.5
|   |   |   |   |   |   e < 0.5 : 23.84 (5/0.06) [3/0.84]
|   |   |   |   |   |   e >= 0.5 : 23.06 (2/0.03) [2/0.13]
|   |   |   |   |   Workload >= 71.5
|   |   |   |   |   |   e < 0.5
|   |   |   |   |   |   |   Workload < 72.5 : 24 (2/0.12) [0/0]
|   |   |   |   |   |   |   Workload >= 72.5 : 25.23 (3/0.04)
```

```
      [5/4.33]
 97 |   |   |   |   |   |   e >= 0.5 : 24.16 (4/0.12) [1/0.86]
 98 |   Workload >= 76.5
 99 |   |   Workload < 89.5
100 |   |   |   Workload < 84.5
101 |   |   |   |   c < 0.5
102 |   |   |   |   |   Workload < 77.5 : 26.59 (3/11.39) [0/0]
103 |   |   |   |   |   Workload >= 77.5
104 |   |   |   |   |   |   Workload < 82.5
105 |   |   |   |   |   |   |   Workload < 80.5
106 |   |   |   |   |   |   |   |   Workload < 79.5 : 24.34 (3/0.04)
      [3/0.05]
107 |   |   |   |   |   |   |   |   Workload >= 79.5 : 24.78 (3/0.06)
      [0/0]
108 |   |   |   |   |   |   |   Workload >= 80.5 : 25.23 (5/0.1) [1/0.6]
109 |   |   |   |   |   |   Workload >= 82.5 : 25.71 (5/0.19) [1/0]
110 |   |   |   |   c >= 0.5
111 |   |   |   |   |   Workload < 78.5 : 25.06 (6/0.02) [0/0]
112 |   |   |   |   |   Workload >= 78.5
113 |   |   |   |   |   |   Workload < 80.5 : 25.98 (5/0.12) [1/0.06]
114 |   |   |   |   |   |   Workload >= 80.5 : 27.03 (8/0.09) [4/1.47]
115 |   |   |   Workload >= 84.5
116 |   |   |   |   c < 0.5 : 26.66 (10/0.3) [5/0.18]
117 |   |   |   |   c >= 0.5
118 |   |   |   |   |   Workload < 88.5
119 |   |   |   |   |   |   f < 0.5
120 |   |   |   |   |   |   |   Workload < 87 : 27.26 (3/0.08) [1/0.44]
121 |   |   |   |   |   |   |   Workload >= 87 : 27.75 (2/0) [2/0.03]
122 |   |   |   |   |   |   f >= 0.5 : 27.19 (4/0.05) [0/0]
123 |   |   |   |   |   Workload >= 88.5 : 28.01 (3/0.24) [0/0]
124 |   |   Workload >= 89.5
125 |   |   |   Workload < 93.5
126 |   |   |   |   a < 0.5 : 28.82 (8/1.92) [8/0.25]
127 |   |   |   |   a >= 0.5 : 27.56 (7/0.04) [1/0]
128 |   |   |   Workload >= 93.5
129 |   |   |   |   Workload < 96.5
130 |   |   |   |   |   c < 0.5
131 |   |   |   |   |   |   Workload < 95.5 : 28.34 (3/0.06) [3/0.06]
132 |   |   |   |   |   |   Workload >= 95.5 : 28.76 (2/0.03) [1/0.29]
133 |   |   |   |   |   c >= 0.5
134 |   |   |   |   |   |   d < 0.5 : 29.62 (4/0.17) [2/0.18]
135 |   |   |   |   |   |   d >= 0.5 : 30.18 (3/0.54) [0/0]
136 |   |   |   |   Workload >= 96.5 : 30.46 (13/4.19) [11/0.39]
137
138 Size of the tree : 133
```

LIST OF REFERENCES

[1] Debtree. http://collab-maint.alioth.debian.org/debtree/. Accessed: 03-02-2015.

[2] Rescue: An energy-aware scheduler for cloud environments. *Sustainable Computing: Informatics and Systems*, 4(4):215 – 224, 2014. Special Issue on Energy Aware Resource Management and Scheduling (EARMS).

[3] José Á.Galindo, Fabricia Roos-Frantz, Jesús García-Galán, and Antonio Ruiz-Cortés. Extracting orthogonal variability models from debian repositories. In *Proc. of 2nd International Workshop on Formal Methods and Analysis in Software Product Line Engineering (FMSPLE 2011), co-located with Software Product Line Conference 2011 (SPLC 2011)*, pages 8–8, Munich, 08/2011 2011. Fraunhofer, Fraunhofer.

[4] Marwah M. Alansari and Behzad Bordbar. An Architectural Framework for Enforcing Energy Management Policies in Cloud. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 717–724. IEEE, jun 2013.

[5] G.H. Alférez, V. Pelechano, R. Mazo, C. Salinesi, and D. Diaz. Dynamic adaptation of service compositions with variability models. *Journal of Systems and Software*, 91:24 – 47, 2014.

[6] Apache. Apache jmeter. http://jmeter.apache.org. Accessed: 04/01/2014.

[7] Apache-Project. Apache http server benchmarking tool. https://httpd.apache.org/docs/2.4/programs/ab.html. Accessed: 02/03/2016.

[8] Uwe Aßmann, Sebastian Götz, Jean-Marc Jézéquel, Brice Morin, and Mario Trapp. *A Reference Architecture and Roadmap for Models@run.time Systems*, pages 1–18. Springer International Publishing, Cham, 2014.

[9] RandallC. Bachmeyer and HarryS. Delugach. A conceptual graph approach to feature modeling. In Uta Priss, Simon Polovina, and Richard Hill, editors, *Conceptual Structures: Knowledge Architectures for Smart Applications*, volume 4604 of *Lecture Notes in Computer Science*, pages 179–191. Springer Berlin Heidelberg, 2007.

[10] Ed Bailey. *Maximum RPM*. Sams, Indianapolis, IN, USA, 1st edition, 1997.

[11] Michal Bali. *Drools JBoss Rules 5.0 Developer's Guide*. Packt Publishing, 2009.

[12] J. Baliga, R.W.A. Ayre, K. Hinton, and R.S. Tucker. Green cloud computing: Balancing energy in processing, storage, and transport. *Proceedings of the IEEE*, 99(1):149 –167, jan. 2011.

[13] Robert Basmadjian, Hermann De Meer, Ricardo Lent, and Giovanni Giuliani. Cloud computing and its interest in saving energy: the use case of a private cloud. *Journal of Cloud Computing*, 1(1):1–25, 2012.

[14] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An open source software for exploring and manipulating networks, 2009.

[15] Vladimir Batagelj and Andrej Mrvar. Pajek - analysis and visualization of large networks. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 477–478. Springer Berlin Heidelberg, 2002.

[16] Anton Beloglazov. *Energy-efficient management of virtual machines in data centers for cloud computing*. PhD thesis, University of Melbourne, February 2013.

[17] Anton Beloglazov and Rajkumar Buyya. Energy efficient allocation of virtual machines in cloud data centers. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 577–578, Washington, DC, USA, 2010. IEEE Computer Society.

[18] Nelly Bencomo. *Supporting the Modelling and Generation of Reflective Middleware Families and Applications using Dynamic Variability*. PhD thesis, Lancaster University, 2008.

[19] Amel Bennaceur, Robert France, Giordano Tamburrelli, Thomas Vogel, Pieter J. Mosterman, Walter Cazzola, Fabio M. Costa, Alfonso Pierantonio, Matthias Tichy, Mehmet Akşit, Pär Emmanuelson, Huang Gang, Nikolaos Georgantas, and David Redlich. *Mechanisms for Leveraging Models at Runtime in Self-adaptive Software*, pages 19–46. Springer International Publishing, Cham, 2014.

[20] Andreas Bergen, Nina Taherimakhsousi, and Hausi A. Müller. Adaptive management of energy consumption using adaptive runtime models. In *Proceedings of*

the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS '15, pages 120–126, Piscataway, NJ, USA, 2015. IEEE Press.

[21] Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wasowski, and Krzysztof Czarnecki. A study of variability models and languages in the systems software dom ain. *IEEE Transactions on Software Engineering*, 39(12):1611–1640, 2013.

[22] Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. Leveraging bagging for evolving data streams. In *Proceedings of the 2010 European Conference on Machine Learning and Knowledge Discovery in Databases: Part I*, ECML PKDD'10, pages 135–150, Berlin, Heidelberg, 2010. Springer-Verlag.

[23] C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer, USA, 2006.

[24] G. Blair, N. Bencomo, and R.B. France. Models@ run.time. *Computer*, 42(10):22–27, Oct 2009.

[25] A. E. Husain Bohra and V. Chaudhary. Vmeter: Power modelling for virtualized clouds. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8, April 2010.

[26] A. Borovyi, V. Kochan, Z. Dombrovskyy, V. Turchenko, and A. Sachenko. Device for measuring instant current values of cpu's energy consumption. In *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2009. IDAACS 2009. IEEE International Workshop on*, pages 126–130, 2009.

[27] Jan Bosch, Gert Florijn, Danny Greefhorst, Juha Kuusela, J.Henk Obbink, and Klaus Pohl. Variability issues in software product lines. In Frank van der Linden, editor, *Software Product-Family Engineering*, volume 2290 of *Lecture Notes in Computer Science*, pages 13–21. Springer Berlin Heidelberg, 2002.

[28] Sigismondo Boschi and Gabriele Santomaggio. *RabbitMQ Cookbook*. Packt Publishing, 2013.

[29] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[30] L. Breiman, J. Friedman, C. Stone, and R. Olshen. *Classication and Regression Trees*. Wadsworth and Brooks, 1984.

[31] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[32] Broad-consulting. Power market, power pricing and data centres in europe. http://www.broad-group.com/reports/power-market. Accessed: 09/06/2016.

[33] Rajkumar Buyya. Market-oriented cloud computing: Vision, hype, and reality of delivering computing as the 5th utility. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 1–, Washington, DC, USA, 2009. IEEE Computer Society.

[34] Rajkumar Buyya, Anton Beloglazov, and Jemal H. Abawajy. Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. *CoRR*, abs/1006.0308, 2010.

[35] Rajkumar Buyya, Rodrigo N. Calheiros, and Xiaorong Li. Autonomic cloud computing: Open challenges and architectural elements. *CoRR*, abs/1209.3356, 2012.

[36] Rafael Capilla, Jan Bosch, Pablo Trinidad, Antonio Ruiz-Corts, and Mike Hinchey. An overview of dynamic software product line architectures and techniques: Observations from research and industry. *Journal of Systems and Software*, 91:3 – 23, 2014.

[37] Eugenio Capra, Chiara Francalanci, and Sandra A. Slaughter. Measuring application software energy efficiency. *IT Professional*, 14(2):54–61, 2012.

[38] Walter Cazzola. *Evolution as Reflections on the Design*, pages 259–278. Springer International Publishing, Cham, 2014.

[39] Francesco Cesarini and Simon Thompson. *ERLANG Programming*. O'Reilly Media, Inc., 1st edition, 2009.

[40] C. Cetina, P. Giner, J. Fons, and V. Pelechano. Autonomic computing through reuse of variability models at runtime: The case of smart homes. *Computer*, 42(10):37–43, Oct 2009.

[41] Carlos Cetina, Pau Giner, Joan Fons, and Vicente Pelechano. Prototyping dynamic software product lines to evaluate run-time reconfigurations. *Sci. Comput. Program.*, 78(12):2399–2413, December 2013.

[42] Scott Chacon and Ben Straub. *Pro Git*. Apress, https://git-scm.com/book/en/v2, 2014.

[43] T. Chai and R. R. Draxler. Root mean square error (rmse) or mean absolute error (mae)? arguments against avoiding rmse in the literature. *Geoscientific Model Development*, 7(3):1247–1250, 2014.

[44] Lianping Chen, Muhammad Ali Babar, and Nour Ali. Variability management in software product lines: A systematic review. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 81–90, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[45] Qingwen Chen, P. Grosso, K. van der Veldt, C. de Laat, R. Hofman, and H. Bal. Profiling energy consumption of vms for green cloud computing. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 768 –775, dec. 2011.

[46] D. Cheng, J. Rao, C. Jiang, and X. Zhou. Elastic power-aware resource provisioning of heterogeneous workloads in self-sustainable datacenters. *IEEE Transactions on Computers*, 65(2):508–521, Feb 2016.

[47] David Chou. Rise of the cloud ecosystems. http://blogs.msdn.com/b/dachou/archive/2011/03/16/rise-of-the-cloud-ecosystems.aspx. Accessed: 28/12/2015.

[48] Jack Clark. IT Electricity Use Worse Than You Thought. http://www.theregister.co.uk/2013/08/16/. Accessed: 02/07/2014.

[49] P. Clements and L.M. Northrop. *Software Product Lines: Practices and Patterns*. The SEI Series in Software Engineering. Prentice Hall, 2002.

[50] Louis Columbus. Roundup of cloud computing forecasts and market estimates q3 update. http://www.forbes.com/sites/louiscolumbus/2015/09/27/roundup-of-cloud-computing-forecasts-and-market-estimates-q3-update-2015, September 2015. [Online; posted 27-September-2015].

[51] Technologies Corp. Web servers market position report. http://w3techs.com/technologies/market/webserver/10. Accessed: 25/11/2013.

[52] National Research Council. *Advancing the Science of Climate Change*. The National Academies Press, 2010.

[53] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.

[54] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative programming: methods, tools, and applications*. ACM Press, New York, NY, USA, 2000.

[55] Debian. Debian packages. Accessed: 5/01/2015.

[56] Richard Delaney. Vagrant. *Linux J.*, 2014(244), August 2014.

[57] Pierre Delforge and Josh Whitney. Data Center Efficiency Assessment. Technical report, Natural Resources Defence Council, 08 2014.

[58] Roberto Di Cosmo and Stefano Zacchiroli. Feature diagrams as package dependencies. In *Proceedings of the 14th International Conference on Software Product Lines: Going Beyond*, SPLC'10, pages 476–480, Berlin, Heidelberg, 2010. Springer-Verlag.

[59] Thomas G. Dietterich. Ensemble methods in machine learning. In *Proceedings of the First International Workshop on Multiple Classifier Systems*, MCS '00, pages 1–15, London, UK, UK, 2000. Springer-Verlag.

[60] F Dorey. *In Brief: The P Value: What Is It and What Does It Tell You?*, pages 2297 – 2298. Number 468(8). Journal of Clinical Orthopaedics and Related Research, PMC, 2010.

[61] Brian Dougherty, Jules White, and Douglas C. Schmidt. Model-driven auto-scaling of green cloud computing infrastructure. *Future Gener. Comput. Syst.*, 28(2):371–378, February 2012.

[62] Clemens Dubslaff, Sascha Klüppelholz, and Christel Baier. Probabilistic model checking for energy analysis in software product lines. In *Proceedings of the 13th International Conference on Modularity*, MODULARITY '14, pages 169–180, New York, NY, USA, 2014. ACM.

[63] Robert Dyer. *Bringing ultra-large-scale software repository mining to the masses with Boa*. PhD thesis, Iowa State University, 2013.

[64] Bruce Eckel. *Thinking in Java (4th Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[65] ecos. eCos. http://ecos.sourceware.org/. Accessed: 5/01/2015.

[66] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Derivation of feature component maps by means of concept analysis. In *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, CSMR '01, pages 176–, Washington, DC, USA, 2001. IEEE Computer Society.

[67] Elastichost. Iaas Architecture. http://www.elastichosts.co.uk/. Accessed: 16/04/2013.

[68] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen North, Gordon Woodhull, Short Description, and Lucent Technologies. Graphviz open source graph drawing tools. In *Lecture Notes in Computer Science*, pages 483–484. Springer-Verlag, 2001.

[69] P. T. Endo, D. Sadok, and J. Kelner. Autonomic cloud computing: Giving intelligence to simpleton nodes. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 502–505, Nov 2011.

[70] EPA. Carbon dioxide emissions. https://www3.epa.gov/climatechange/ghgemissions/gases/co2 Accessed: 09/06/2016.

[71] Magnus Eriksson and Alvis Hagglunds. An introduction to software product line development. In *Proceedings of Umeå's Seventh Student Conference in Computing Science*, pages 26–37, 2003.

[72] Michel Ezran, Maurizio Morisio, and Colin Tully. *Practical Software Reuse*. Springer-Verlag, London, UK, UK, 2002.

[73] FAMA. Fama tool suite. http://www.isa.us.es/fama/. Accessed: 26-01-2015.

[74] Xi Fang, Satyajayant Misra, Guoliang Xue, and Dejun Yang. Smart grid - the new and improved power grid: A survey. *IEEE Communications Surveys and Tutorials*, 2011.

[75] D.G. Feitelson. *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge University Press, 2015.

[76] DrorG. Feitelson. Workload modeling for performance evaluation. In MariaCarla Calzarossa and Salvatore Tucci, editors, *Performance Evaluation of Complex Systems: Techniques and Tools*, volume 2459 of *Lecture Notes in Computer Science*, pages 114–141. Springer Berlin Heidelberg, 2002.

[77] Jason Flinn and M. Satyanarayanan. Powerscope: a tool for profiling the energy usage of mobile applications. In *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, pages 2–10, Feb 1999.

[78] Martin Fowler. *Domain Specific Languages*. Addison-Wesley Professional, 1st edition, 2010.

[79] FreeBSD. FreeBSD Packages. http://pkg.freebsd.org/. Accessed: 5/01/2015.

[80] Cristina Gacek, Peter Knauber, Klaus Schmid, and Paul Clements. Successful software product line development in a small organization. *Institut Experimentelles Software Engineering Fraunhofer - No. 013.01*, 2001.

[81] Jos Galindo, David Benavides, and Sergio Segura. Debian packages repositories as software product line models. towards automated analysis. In *ACoTA*, volume 688 of *CEUR Workshop Proceedings*, pages 29–34. CEUR-WS.org, 2010.

[82] Nadia Gamez, Lidia Fuentes, and Miguel A. Aragüez. Autonomic computing driven by feature models and architecture in famiware. In *Proceedings of the 5th European Conference on Software Architecture*, ECSA'11, pages 164–179, Berlin, Heidelberg, 2011. Springer-Verlag.

[83] EmdenR. Gansner and Yifan Hu. Efficient node overlap removal using a proximity stress model. In IoannisG. Tollis and Maurizio Patrignani, editors, *Graph Drawing*, volume 5417 of *Lecture Notes in Computer Science*, pages 206–217. Springer Berlin Heidelberg, 2009.

[84] Jim Gao. Machine learning applications for data center optimization. http://research.google.com/pubs/pub42542.html, 2014.

[85] K. Geihs, P. Barone, F. Eliassen, J. Floch, R. Fricke, E. Gjorven, S. Hallsteinsen, G. Horn, M. U. Khan, A. Mamelli, G. A. Papadopoulos, N. Paspallis, R. Reichle, and E. Stav. A comprehensive solution for application-level adaptation. *Softw. Pract. Exper.*, 39(4):385–422, March 2009.

[86] Zoubin Ghahramani. Unsupervised learning. In Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rtsch, editors, *Advanced Lectures on Machine Learning*, volume 3176 of *Lecture Notes in Computer Science*, pages 72–112. Springer Berlin Heidelberg, 2004.

[87] Nicolas Edward Gillian. *Gesture Recognition for Musician Computer Interaction*. PhD thesis, Queen's University of Belfast, UK, 2011.

[88] Ravi Giri and Anand Vamchi. Increasing data center efficienncy with server power measurements. intel white paper, 2010.

[89] Git. Git SCM. https://git-scm.com/. Accessed: 5/11/2014.

[90] T.H. Glisson. *Introduction to Circuit Analysis and Design*. Springer Netherlands, 2011.

[91] Steven Goodman. A dirty dozen: Twelve p-value misconceptions. *Seminars in Hematology*, 45(3):135 – 140, 2008. Interpretation of Quantitative Research.

[92] Google. Data Centres - Google Green. http://www.google.com/green/efficiency/datacenters/. Accessed: 10/02/2013.

[93] Google. Data Centres-Renewable Energy. https://www.google.com/about/datacenters/. Accessed: 09/06/2016.

[94] Ian Gordon. Introduction to Gawk. *Linux Journal*, 1996(25es), May 1996.

[95] Sebastian Götz, Claas Wilke, Sebastian Cech, and Uwe Aßmann. Runtime variability management for energy-efficient software by contract negotiation. In *Proceedings of the International Workshop on Models@ run. time*, 2011.

[96] Paul Grace, Gordon S. Blair, Carlos Flores Cortes, and Nelly Bencomo. Engineering complex adaptations in highly heterogeneous distributed systems. In *Proceedings of the 2Nd International Conference on Autonomic Computing and Communication Systems*, Autonomics '08, pages 27:1–27:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[97] Paul Grace, Danny Hughes, Barry Porter, Gordon S. Blair, Geoff Coulson, and Francois Taiani. Experiences with open overlays: A middleware approach to network heterogeneity. *SIGOPS Oper. Syst. Rev.*, 42(4):123–136, April 2008.

[98] James Gray. Go green, save green with linux. *Linux J.*, 2008(168), April 2008.

[99] J. Guo, K. Czarnecki, S. Apel, N. Siegmund, and A. Wasowski. Variability-aware performance prediction: A statistical learning approach. In *ASE'13*, pages 301–311, 2013.

[100] Peter Bauer Karl Kohne H. M. James Hung, Robert T. O'Neill. The behavior of the p-value when the alternative hypothesis is true. *Biometrics*, 53(1):11–22, 1997.

[101] Daniel Hall. *Ansible Configuration Management*. Packt Publishing, Birmingham, UK, 2013.

[102] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[103] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE TSE*, 38(6):1276–1303, 2012.

[104] S. Hallsteinsen, M. Hinchey, Sooyong Park, and K. Schmid. Dynamic software product lines. *Computer*, 41(4):93–95, April 2008.

[105] F. Harary. *Graph Theory*. Addison-Wesley Series in Mathematics. Perseus Books, 1994.

[106] R. Hertzog and R. Mas. *The Debian Administrator's Handbook: Debian Squeeze from Discovery to Mastery*. 2012.

[107] M. Hinchey, Sooyong Park, and K. Schmid. Building dynamic software product lines. *Computer*, 45(10):22–26, Oct 2012.

[108] Ralph Hintemann and Klaus Fichter. Materialbestand der Rechenzentren in Deutschland - Eine Bestandsaufnahme zur Ermittlung von Ressourcen- und Energieeinsatz. Technical report, Umweltbundesamt, 11 2010.

[109] HP. Hewlett-packard loadrunner. http://www.hp.com/go/loadrunner. Accessed: 02/03/2016.

[110] Jin Huang. *Performance Measures of Machine Learning*. PhD thesis, Ontario, Canada, 2006. AAINR30363.

[111] IBM. Ibm green data centre. Accessed: 20/01/2013.

[112] IBM. KVM Open Virtualization. http://www-03.ibm.com/systems/virtualization/. Accessed: 19/04/2013.

[113] IBM. An architectural blueprint for autonomic computing. Technical report, 2005.

[114] Intel. Intel energy checker. http://software.intel.com/en-us/articles/intel-energy-checker-sdk. Accessed: 10/01/2013.

[115] Intel. Intel power gadget. http://software.intel.com/en-us/articles/intel-power-gadget-20. Accessed: 10/01/2013.

[116] Intel Open Source Technology Center. *Getting maximum mileage out of tickless*, June 2007.

[117] Nathalie Japkowicz and Mohak Shah. *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press, New York, NY, USA, 2011.

[118] Y. Jiang, C. s. Perng, T. Li, and R. Chang. Self-adaptive cloud capacity planning. In *Services Computing (SCC), 2012 IEEE Ninth International Conference on*, pages 73–80, June 2012.

[119] Sincero Julio and Wolfgang. The linux kernel configurator as a feature modeling tool. 2008.

[120] Dimitris Kalamaras. Social network analysis made easy. http://socnetv.sourceforge.net/. Accessed: 04/07/2015.

[121] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[122] KyoC. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. Form: A feature-;oriented reuse method with domain-;specific reference architectures. *Annals of Software Engineering*, 5:143–168, 1998.

[123] Aman Kansal, Feng Zhao, Jie Liu, Nupur Kothari, and Arka A. Bhattacharya. Virtual machine power metering and provisioning. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 39–50, New York, NY, USA, 2010. ACM.

[124] Karen. Assesing the Fit Regression Models. http://www.theanalysisfactor.com/assessing-the-fit-of-regression-models/. Accessed: 5/09/2016.

[125] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, Jan 2003.

[126] E. Kocaguneli, T. Menzies, A. Bener, and J. W. Keung. Exploiting the essential assumptions of analogy-based effort estimation. *IEEE TSE*, 38(2):425–438, 2012.

[127] Ekrem Kocaguneli, Tim Menzies, and Emilia Mendes. Transfer learning in effort estimation. *Empirical Software Engineering*, 20(3):813–843, 2014.

[128] Sergiy S. Kolesnikov, Sven Apel, Norbert Siegmund, Stefan Sobernig, Christian Kastner, and Semah Senkaya. Predicting quality attributes of software product lines using software and network measures and sampling. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS '13, pages 6:1–6:5, New York, NY, USA, 2013. ACM.

[129] Jerzy Kolinski, Barry Press, Ram Chary, and Andrew Henroid. *Building the Power-Efficient PC: A Developer's Guide to ACPI Power Management with CD-ROM.* Intel Press, 2001.

[130] Jeff Kramer and Jeff Magee. The evolving philosophers problem: Dynamic change management. *IEEE Trans. Softw. Eng.*, 16(11):1293–1306, November 1990.

[131] KVM. Virtio. http://www.linux-kvm.org/page/Virtio. Accessed: 25/09/2013.

[132] B. Lantz. *Machine Learning with R.* Packt Publishing, Birmingham, UK, 2013.

[133] Kien Le, Ricardo Bianchini, Jingru Zhang, Yogesh Jaluria, Jiandong Meng, and Thu D. Nguyen. Reducing electricity cost through virtual machine placement in high performance computing clouds. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 22:1–22:12, New York, NY, USA, 2011. ACM.

[134] Jian Li, Kai Shuang, Sen Su, Qingjia Huang, Peng Xu, Xiang Cheng, and Jie Wang. Reducing operational costs through consolidation with resource prediction in the cloud. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, CCGRID '12, pages 793–798, Washington, DC, USA, 2012. IEEE Computer Society.

[135] Yanfei Li, Ying Wang, Bo Yin, and Lu Guan. An online power metering model for cloud environment. In *Network Computing and Applications (NCA), 2012 11th IEEE International Symposium on*, pages 175–180, 2012.

[136] Yang Li, Di Wang, Saugata Ghose, Jie Liu, Sriram Govindan, Sean James, Eric Peterson, John Siegler, Rachata Ausavarungnirun, and Onur Mutlu. Sizecap: Efficiently handling power surges in fuel cell powered data centers. In *2016 IEEE International Symposium on High Performance Computer Architecture, HPCA 2016, Barcelona, Spain, March 12-16, 2016*, pages 444–456, 2016.

[137] linfo. The linux information project. http://linfo.org. Accessed: 04/05/2016.

[138] Ning Liu, Ziqian Dong, and Roberto Rojas-Cessa. Task scheduling and server provisioning for energy-efficient cloud-computing data centers. In *Proceedings of the 2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops*, ICDCSW '13, pages 226–231, Washington, DC, USA, 2013. IEEE Computer Society.

[139] Zhen Liu, Nicolas Niclausse, and César Jalpa-Villanueva. Traffic model and performance evaluation of web servers. *Perform. Eval.*, 46(2-3):77–100, October 2001.

[140] Felix Loesch and Erhard Ploedereder. Restructuring variability in software product lines using concept analysis of product configurations. In *Proceedings of the 11th European Conference on Software Maintenance and Reengineering*, CSMR '07, pages 159–170, Washington, DC, USA, 2007. IEEE Computer Society.

[141] Roberto E. Lopez-Herrejon and Don S. Batory. A standard problem for evaluating product-line methodologies. GCSE '01, pages 10–24, London, UK, UK, 2001. Springer-Verlag.

[142] Junheng Ma. *Contributions to Numerical Formal Concept Analysis, Bayesian Predictive Inference, and Sample Size Determination.* PhD thesis, Case Western Reserve University, Ohio, 2011.

[143] Zaheed Mahmood, David Bowes, Peter Lane, and Tracy Hall. What is the impact of imbalance on software defect prediction performance? In *PROMISE*, pages 1–4, 2015.

[144] Kevin Maney. Cloud computing is killing the traditional office. http://www.newsweek.com/2014/07/25/cloud-computing-killing-traditional-office, July 2014. [Online; posted 17-July-2014].

[145] David Meisner, Brian T. Gold, and Thomas F. Wenisch. Powernap: Eliminating server idle power. *SIGARCH Comput. Archit. News*, 37(1):205–216, March 2009.

[146] David Meisner and Thomas F. Wenisch. Does low-power design imply energy efficiency for data centers? In *Proceedings of the 17th IEEE/ACM International Symposium on Low-power Electronics and Design*, ISLPED '11, pages 109–114, Piscataway, NJ, USA, 2011. IEEE Press.

[147] Peter M. Mell and Timothy Grance. Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, United States, 2011.

[148] T. Menzies, E. Kocaguneli, L. L. Minku, F. Peters, and B. Turhan. *Sharing Data and Models in Software Engineering.* Morgan Kaufmann, 2014.

[149] Andreas Merkel and Frank Bellosa. Balancing power consumption in multiprocessor systems. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, EuroSys '06, pages 403–414, New York, NY, USA, 2006. ACM.

[150] Microsoft. Joulemeter. http://research.microsoft.com/. Accessed: 07/01/2013.

[151] Microsoft. Microsoft data centres - renewable energy. https://www.microsoft.com/about/csr/environment/renewable-energy/. Accessed: 09/06/2016.

[152] Frederic P. Miller, Agnes F. Vandome, and John McBrewster. *Advanced Configuration and Power Interface: Open Standard, Operating System, Power Management,*

Cross- Platform, Intel Corporation, Microsoft, Toshiba, ... Sleep Mode, Hibernate (OS Feature), Synonym. Alpha Press, 2009.

[153] Lauri Minas and Brad Ellison. *Energy efficiency for information technology: How to reduce power consumption in servers and data centers.* Intel Press, 2009.

[154] Leandro L. Minku and Xin Yao. Ensembles and locality: Insight on improving software effort estimation. *Information and Software Technology*, 55(8):1512–1528, 2013.

[155] Leandro L. Minku and Xin Yao. How to make best use of cross-company data in software effort estimation? In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 446–456, New York, NY, USA, 2014. ACM.

[156] Leandro Lei Minku. *Online ensemble learning in the presence of concept drift.* PhD thesis, University of Birmingham, UK, July 2011.

[157] Thomas M. Mitchell. *Machine Learning.* McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[158] A. H. Mohsenian-Rad and A. Leon-Garcia. Coordination of cloud computing and smart power grids. In *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, pages 368–372, Oct 2010.

[159] Moodle. Test site generator. https://docs.moodle.org/25/en/Test-site-generator. Accessed: 04/12/2015.

[160] Hans Martin Mrsk-Mller and Bo Nrregaard Jrgensen. *A Software Product Line for Energy-efficient Control of Supplementary Lighting in Greenhouses*, pages 37–46. SciTePress - Science and Technology Publications, 2011.

[161] I Made Murwantara and Behzad Bordbar. A simplified method of measurement of energy consumption in cloud and virtualized environment. In *Big Data and Cloud Computing (BdCloud), 2014 IEEE Fourth International Conference on*, pages 654–661, Dec 2014.

[162] I Made Murwantara, Behzad Bordbar, and Leandro L. Minku. Measuring energy consumption for web service product configuration. In *Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '14, pages 224–228, New York, NY, USA, 2014. ACM.

[163] Andrew Ng. Advice for Applying Machine Learning, Lecture Material. http://cs229.stanford.edu/materials/, 2016.

[164] Nginx. Nginx. http://nginx.org/. Accessed: 5/01/2013.

[165] Nicolas Niclausse. *Modlisation, valuation de performances et dimensionnement du World Wide Web*. PhD thesis, Universit de Nice Sophia-Antipolis, June 1999.

[166] F. Norouzi and M. A. Bauer. Toward an autonomic energy efficient data center. In *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*, pages 487–493, Nov 2012.

[167] Adel Noureddine. *Towards a Better Understanding of the Energy Consumption of Software Systems*. PhD thesis, Université des Sciences et Technologie de Lille - Lille I, France, 2014.

[168] Robbert Christiaan van Ommering. *Building Product Populations with Software Components*. PhD thesis, 2004.

[169] OpenSuSe. RPM Packages. https://en.opensuse.org/openSUSEStandardsRpmMetadata. Accessed: 5/01/2015.

[170] Nikunj C. Oza. *Online Ensemble Learning*. PhD thesis, The University of California, Berkeley, CA, Sep 2001.

[171] Nikunj C. Oza and Stuart J. Russell. Online bagging and boosting. In *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics, AISTATS 2001, Key West, Florida, US, January 4-7, 2001*, 2001.

[172] Erik Pacheco. *Unsupervised Learning with R*. Packt Publishing.

[173] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communication ACM*, 15(12):1053–1058, December 1972.

[174] D. L. Parnas. On the design and development of program families. *IEEE Transaction of Software Engineering*, 2(1):1–9, January 1976.

[175] Carlos Parra. *Towards Dynamic Software Product Lines: Unifying Design and Runtime Adaptations*. Theses, Université des Sciences et Technologie de Lille - Lille I, March 2011.

[176] Raúl Peña Ortiz. *Accurate workload design for web performance evaluation.* PhD thesis, Editorial Universitat Politècnica de València, 2013.

[177] PennState. Hypothesis Testing (P-value approach) - Eberly College of Science. https://onlinecourses.science.psu.edu/statprogram/node/138. Accessed: 1/09/2016.

[178] Michael Pilato. *Version Control With Subversion*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2004.

[179] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[180] F. Quesnel, H.K. Mehta, and J.-M. Menaud. Estimating the power consumption of an idle virtual machine. In *GreenCom 2013*, Aug 2013.

[181] Chet Ramey. What's gnu: Bash-the gnu shell. *Linux J.*, 1994(4es), August 1994.

[182] William Rice. *Moodle E-Learning Course Development*. Packt Publishing, 2006.

[183] F.M. Roth, C. Krupitzer, and C. Becker. Runtime evolution of the adaptation logic in self-adaptive systems. In *Autonomic Computing (ICAC), 2015 IEEE International Conference on*, pages 141–142, July 2015.

[184] Deborah Rumsey. *Statistics for Dummies*. Wiley, Hoboken, NJ, 2016.

[185] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[186] Uwe Ryssel, Joern Ploennigs, and Klaus Kabitzsch. Extraction of feature models from formal contexts. In *Proceedings of the 15th International Software Product Line Conference, Volume 2*, SPLC '11, pages 4:1–4:8, New York, NY, USA, 2011. ACM.

[187] Kai Sachs, Samuel Kounev, Jean Bacon, and Alejandro P. Buchmann. Workload characterization of the specjms2007 benchmark. In *Formal Methods and Stochastic Models for Performance Evaluation, Fourth European Performance Engineering Workshop, EPEW 2007, Berlin, Germany, September 27-28, 2007, Proceedings*, pages 228–244, 2007.

[188] A.L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, July 1959.

[189] Sergio Segura, David Benavides, Antonio Ruiz-Cortés, and Pablo Trinidad. Generative and transformational techniques in software engineering ii. chapter Automated Merging of Feature Models Using Graph Transformations, pages 489–505. Springer-Verlag, Berlin, Heidelberg, 2008.

[190] David C. Sharp. Component-based product line development of avionics software. In *Proceedings of the First Conference on Software Product Lines : Experience and Research Directions: Experience and Research Directions*, pages 353–370, Norwell, MA, USA, 2000. Kluwer Academic Publishers.

[191] Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wasowski, and Krzysztof Czarnecki. The variability model of the linux kernel. In *Fourth International Workshop on Variability Modelling of Software-intensive Systems (VAMOS'10)*, Linz, Austria, 2010.

[192] M. Shepperd and S. McDonell. Evaluating prediction systems in software project estimation. *IST*, 54(8):820–827, 2012.

[193] N. Siegmund, S. Kolesnikov, C. Kästner, S. Apel, D. Batory, M. Rosen-müller, and G. Saake. Predicting performance via automated feature-interaction detection. In *ICSE'12*, pages 167–177, 2012.

[194] N. Siegmund, M. Rosenmuller, M. Kuhlemann, C. Kastner, and G. Saake. Measuring non-functional properties in software product line for product derivation. In *Software Engineering Conference, 2008. APSEC '08. 15th Asia-Pacific*, pages 187–194, Dec 2008.

[195] Norbert Siegmund. *Measuring and Predicting Non-Functional Properties of Customizable Programs*. Phd theses, University of Magdeburg, Germany, November 2012.

[196] Norbert Siegmund, Marko Rosenmüller, and Sven Apel. Automating energy optimization with features. In *Proceedings of the 2Nd International Workshop on Feature-Oriented Software Development*, FOSD '10, pages 2–9, New York, NY, USA, 2010. ACM.

[197] V.K. Singh, K. Dutta, and D. VanderMeer. Estimating the energy consumption of executing software processes. In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pages 94–101, Aug 2013.

[198] N. Sklavos and O. Koufopavlou. Implementation of the sha-2 hash family standard using fpgas. *J. Supercomput.*, 31(3):227–248, March 2005.

[199] James W Smith and Ian Sommerville. Workload classification & software energy measurement for efficient scheduling on private cloud platforms. *arXiv preprint arXiv:1105.2584*, 2011.

[200] Intel Open Source. Powertop. https://01.org/powertop/. Accessed: 11/01/2013.

[201] Gabriele Taentzer. Agg: A graph transformation environment for modeling and validation of software. In JohnL. Pfaltz, Manfred Nagl, and Boris Bhlen, editors, *Applications of Graph Transformations with Industrial Relevance*, volume 3062 of *Lecture Notes in Computer Science*, pages 446–453. Springer Berlin Heidelberg, 2004.

[202] Roberto Tamassia. *Handbook of Graph Drawing and Visualization (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC, 2007.

[203] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2007.

[204] Willy Tarreau. HAProxy - The Reliable, High-Performance TCP/HTTP Load Balancer. http://haproxy.org, 2012. Accessed: 02/08/2015.

[205] Patrick Thibodeau. Data centres are the new polluters. http://www.computerworld.com/article/2598562/data-center/data-centers-are-the-new-polluters.html. Accessed: 25/11/2015.

[206] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. Featureide: An extensible framework for feature-oriented software development. *Sci. Comput. Program.*, 79:70–85, January 2014.

[207] Kevin Tian, Ke Yu, Jun Nakajima, and Winston Wang. How virtualization makes power management different. In *Linux Symposium*, page 205, 2007.

[208] Peter Toft, Derek Coleman, and Joni Ohta. A cooperative model for cross-divisional product development for a software product line. In *Proceedings of the First Conference on Software Product Lines : Experience and Research Directions: Experience and Research Directions*, pages 111–132, Norwell, MA, USA, 2000. Kluwer Academic Publishers.

[209] Tsung. Tsung user's manual. http://tsung.erlang-projects.org. Accessed: 10/01/2013.

[210] B. Turhan and E. Mendes. A comparison of cross- versus single- company effort prediction models for web projects. In *Euromicro Conference on Software Engineering and Advanced Applications*, pages 285–292, 2014.

[211] Ubuntu. Ubuntu Package Repository. http://packages.ubuntu.com/. Accessed: 10/01/2016.

[212] C. Vasudev. *Combinatorics And Graph Theory*. New Age International (P) Limited, 2007.

[213] W. Vereecken, D. Colle, B. Vermeulen, M. Pickavet, B. Dhoedt, and P. Demeester. Estimating and mitigating the energy footprint of icts. Report of meeting of Focus Group on ICT and Climate Change on International Telecomunication Union, 2008.

[214] EkhiotzJon Vergara and Simin Nadjm-Tehrani. Energybox: A trace-driven tool for data transmission energy consumption studies. In Jean-Marc Pierson, Georges Da Costa, and Lars Dittmann, editors, *Energy Efficiency in Large Scale Distributed Systems*, volume 8046 of *Lecture Notes in Computer Science*, pages 19–34. Springer Berlin Heidelberg, 2013.

[215] Thomas Vogel and Holger Giese. Model-driven engineering of self-adaptive software with eurema. *ACM Trans. Auton. Adapt. Syst.*, 8(4):18:1–18:33, January 2014.

[216] Jérôme Vouillon and Roberto Di Cosmo. On software component co-installability. *ACM Trans. Softw. Eng. Methodol.*, 22(4):34:1–34:35, October 2013.

[217] Hao Wang, Jianwei Huang, Xiaojun Lin, and Hamed Mohsenian-Rad. Exploring smart grid and data center interactions for electric power load balancing. *SIGMETRICS Perform. Eval. Rev.*, 41(3):89–94, January 2014.

[218] Shuo Wang and Xin Yao. Using class imbalance learning for software defect prediction. *IEEE Transaction Reliability*, 62(2):434–443, 2012.

[219] Jon Watson. VirtualBox: Bits and Bytes Masquerading As Machines. *Linux Journal*, 2008(166), February 2008.

[220] Wattsupp. Watts Up Power Meter. http://www.wattsupmeters.com. Accessed: 20/01/2013.

[221] Jan Gerben Wijnstra. Component frameworks for a medical imaging product family. In Frank van der Linden, editor, *Software Architectures for Product Families*, volume 1951 of *Lecture Notes in Computer Science*, pages 4–18. Springer Berlin Heidelberg, 2000.

[222] Peng Xiao, Zhigang Hu, Dongbo Liu, Guofeng Yan, and Xilong Qu. Virtual machine power measuring technique with bounded error in cloud environments. *Journal of Network and Computer Applications*, 36(2):818 – 828, 2013.

[223] Yingfei Xiong, Arnaud Hubaux, Steven She, and Krzy sztof Czarnecki. Generating range fixes for software configuration. In *Proceedings of the 34th International Conference on Software Engin eering*, ICSE '12, pages 58–68, Piscataway, NJ, USA, 2012. IEEE Press.

[224] Yi Yu and Saleem Bhatti. Energy measurement for the cloud. In *International Symposium on Parallel and Distributed Processing with Applications*, pages 619–624. IEEE, 2010.

[225] A. Zeileis and K. Hornik. Generalized m-fluctuation tests for parameter instability. *Statistica Neerlandica*, 61(4):488–508, 2007.

[226] A. Zeileis, T. Hothorn, and K. Hornik. Model-based recursive partitioning. *Journal of Computational and Graphical Statistics*, 17(2):492–514, 2008.

[227] Y. Zhang, Y. Wang, and C. Hu. Cloudfreq: Elastic energy-efficient bag-of-tasks scheduling in dvfs-enabled clouds. In *Parallel and Distributed Systems (ICPADS), 2015 IEEE 21st International Conference on*, pages 585–592, Dec 2015.

[228] Ziming Zhang and Song Fu. Profiling and analysis of power consumption for virtualized systems and applications. In *29th International Performance Computing and Communications Conference, IPCCC 2010, 9-11 December 2010, Albuquerque, NM, USA*, pages 329–330, 2010.

[229] Albert Y. Zomaya and Young Choon Lee. *Energy Efficient Distributed Computing Systems*. Wiley-IEEE Computer Society Pr, 1st edition, 2012.