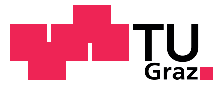

Matthias Vierthaler BSc

Consolidation and Power Analysis of Server Systems on Process Level for Data Centres

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger

Institute for Technical Informatics

Advisors

Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger
Dr. Damian Dalton (University College Dublin)

Graz, May 2015

Abstract

With the recent rise of cloud computing technologies that rely on enormous and powerful data centres the overall energy demand has increased as well. Unfortunately many technologies that helped saving energy on the consumer market are not designed for server hardware - this is addressed by several research papers that developed around the new term *Green Information Technology (IT)*. This thesis lists current trends in server efficiency and explains the need for machine consolidation - which is the act of freeing a device of all its work load in order to suspend it.

The work is based on Papillon - a software package developed by the Irish company Strategia Ltd.. Papillon is a data centre infrastructure management system that helps data centre operators to monitor their servers and aids them in receiving power values without actually using a physical meter. This goal is achieved by first creating a power model of a server and then using that model for measurements. Another requirement is that measured servers need to run an agent that regularly sends utilisation statistics to a centralised point - these values help calculating the demand of energy.

The software is extended in several ways: A user can view the impact of single processes and track them either by name or by assigning a variable to them. The latter method solves the issue of not knowing which applications belong together by constructing a logical group that shares the same variable. Multiple new goals of Papillon are defined and necessary software is implemented. Part of these solutions is a standalone Java graphical user interface that helps classifying processes and calculating the consequences of moving applications between servers. These values are measured via two newly introduced metrics that help understanding the impact. The package also holds a simple optimisation algorithm that distributes the monitored processes as good as possible in order to minimise the amount of needed servers. The thesis is concluded by comparing the previous version of Papillon with the altered one.

Kurzfassung

In den letzten Jahren hat Cloud Computing, das auf performante und große Datenzentren angewiesen ist, die IT Branche verändert. Mit dieser Entwicklung wächst jedoch auch der Energieverbrauch, der zum Betreiben neuer Datenzentren benötigt wird. Viele Technologien, die den Energieverbrauch im Anwenderbereich minimieren, können auf Server-Hardware oft nicht angewendet werden oder bringen nicht den erwünschten Erfolg. Viele Papers, die um den neu definierten Begriff der grünen IT entstanden sind, erforschen dieses Thema. Diese Arbeit beschreibt aktuelle Entwicklungen bezüglich Server-Effizienz und erklärt den Nutzen von Rechner Konsolidierung - ein Vorgang, bei dem versucht wird alle Arbeitsprozesse eines Gerätes auf andere Maschinen zu verteilen, sodass das ursprüngliche Gerät dann abgeschalten werden kann.

Die vorliegende Arbeit basiert auf Papillon - eine Software des irischen Herstellers Stratergia Ltd.. Dabei handelt es sich um eine Infrastruktur Management Software für Rechenzentren, die den Betreiber unterstützt seine Server zu überwachen und aktuelle Verbrauchswerte zu erhalten, ohne dabei ein physisches Messgerät verwenden zu müssen. Dies wird erreicht, indem erst ein Strommodell für den betreffenden Server erzeugt und dieses dann verwendet wird um Messungen anzustellen. Eine weitere Vorgabe ist, dass überwachte Server einen Agenten exekutieren, der regelmäßig Belastungswerte analyiert und an eine zentrale Stelle liefert. Diese Werte helfen, den tatsächlichen Verbrauch an Energie zu messen.

Die Software wurde um mehrere Funktionen erweitert: Ein Benutzer kann nun den Einfluss von einzelnen Prozessen einsehen und sie entweder über den Namen oder, indem eine Variable zugeordnet wird, verfolgen. Das Zuordnen von Variablen löst gleichzeitig das Problem, das der Benutzer nicht wusste welche Applikationen zusammen gehören - nun kann über die gemeinsame Variable die Zusammengehörigkeit bestimmt werden. Mehrere neue Ziele werden definiert und untersucht, notwendige Softwareanpassungen werden durchgeführt. Ein Teil dieser Lösungen ist eine unabhängige Java-Applikation mit graphischer Oberfläche, die hilft Prozesse zu klassifizieren und die Auswirkung einer Verschiebung von Applikationen zwischen Servern zu berechnen. Diese Werte werden mit Hilfe von zwei neu eingeführten Metriken beschrieben, die den Einfluss von Prozessen messen. Die Applikation enthält weiters einen einfachen Optimierungsalgorithmus, der die aufgezeichneten Prozesse so gut wie möglich auf alternative Server verteilt. Das Ziel ist, wie erwähnt, die Konsolidierung von möglichst vielen Maschinen. Eine Gegenüberstellung von der vorherigen und der neuen Version von Papillon schließt die Arbeit ab.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Date Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Datum Unterschrift

Contents

Contents **iv**

List of Figures **vi**

List of Listings **vi**

Acknowledgements **vii**

1 Introduction **1**
1.1 Motivation 1
1.2 Goals 2
1.3 Outline 3

2 Fundamentals **4**
2.1 Metrics 4
2.1.1 Energy - Watt Hours, Watt Minute, Watt Seconds 4
2.1.2 Electrical Power 4
2.1.3 CPU Utilisation 5
2.2 Energy Demand in Server Systems 5
2.3 General Methods in Use for Saving Energy 6
2.3.1 Dynamic Voltage and Frequency Scaling 6
2.3.2 Halting or Deactivation of Components 6
2.3.3 Request Batching 6
2.4 RESTful API 7
2.5 Environment Variables 7
2.6 Comparison of Server Machines 8

3 State of the Art **9**
3.1 Stratergia Papillon 9
3.1.1 Main Features 10
3.1.2 Architecture 10
3.1.3 Power Estimation and Application Power Consumption 11
3.1.4 Papillon Server 11
3.1.5 Papillon Client 12
3.1.6 Papillon Database 12

3.1.7 Papillon Dashboard 13
3.2 Shortcomings of the Existing System 14
3.2.1 Process Tracking 14
3.2.1.1 Processes with unique names 14
3.2.1.2 Processes with same names 15
3.2.2 Process Grouping 15
3.2.3 Process Visualisation 16
3.2.4 Process Costs 16
3.2.5 Consolidation of less utilized Servers 16
3.3 Related Work 16
3.3.1 Power Measuring Systems 16
3.3.1.1 WattApp 16
3.3.1.2 Powerscope 17
3.3.1.3 Koala 18
3.3.1.4 ECOSystem 18
3.3.2 Migration of Virtual Machines 18
3.3.2.1 Live Migration of virtual machines over MAN/WAN 19
3.3.2.2 High Performance Migration of Virtual Machines 19
3.3.3 Consolidation of Processes 20
3.3.3.1 Energy Aware Consolidation for Cloud Computing 20
3.3.3.2 pSciMapper 21
3.3.3.3 V-MAN 21

4 Design **22**
4.1 Specifications 22
4.1.1 Process Analysis 22
4.1.2 Manual Process Relocation 22
4.1.3 Automatic Process Relocation 23
4.2 Process Tracking 23
4.2.1 Name Tracking 23
4.2.2 Unique Process Names by using Hash Function 23
4.2.3 Introduction of Tags 24
4.3 Process Classification 25
4.3.1 Record Reason 25
4.3.2 Process Ranking 26
4.3.2.1 CPU Share 26
4.3.2.2 Energy Share 26
4.4 New Process Computation on Client 26
4.5 Introduction of Consolidator 28
4.5.1 Architecture 28
4.5.2 Work Flow of Consolidator 29
4.5.3 Selection of Hosts 30
4.5.4 Request Host Information 31

4.5.5 Compute Process Information . . . 31
4.5.6 Graphical Feedback via Charts . . . 31
4.6 Manual Process Relocation . . . 32
4.6.1 Goal of Manual Relocation . . . 32
4.6.2 Functionality . . . 32
4.6.3 Example of Process Relocation . . . 32
4.7 Automatic Process Relocation . . . 33
4.7.1 Goal of Automatic Optimisation . . . 33
4.7.2 Used Algorithm . . . 33
4.8 Results of Process Relocation . . . 34
4.9 Design of Consolidator . . . 35
4.9.1 Most important Classes . . . 35
4.9.2 Class Diagram . . . 35
4.10 Design of Server . . . 36

5 Implementation **38**
5.1 Environment . . . 38
5.1.1 Development Environment . . . 38
5.1.2 Runtime Environment . . . 38
5.1.3 Workload Generator . . . 39
5.2 Server Modification . . . 39
5.2.1 Record Reason . . . 40
5.2.2 API Call "processes" . . . 40
5.2.3 Floor and Rack inside Report API call . . . 42
5.3 Client Modification . . . 42
5.3.1 Fix Processes with same names . . . 43
5.3.2 Track Process via Name . . . 43
5.3.2.1 Implementation of the new Work Flow . . . 43
5.3.2.2 Configuration of tracked Names . . . 44
5.3.3 Track Processes via Tag . . . 44
5.3.3.1 Configuration of tracked Tags . . . 45
5.3.3.2 Proper Start of tagged Processes . . . 45
5.4 Consolidator . . . 46
5.4.1 Configuration . . . 46
5.4.2 Graphical User Interface . . . 46
5.4.3 Create Report . . . 47
5.4.4 Analyse Report . . . 47
5.4.5 Move Processes . . . 48
5.4.6 Visual Feedback regarding the Process . . . 49
5.4.7 Auto Relocation . . . 49
5.4.8 Print Function . . . 49

6 Experimental Results **50**
6.1 Verification . . . 50
6.2 Process Tracking . . . 51
6.3 Process Grouping . . . 51
6.4 Process Visualisation . . . 51
6.5 Process Costs . . . 53
6.6 Consolidation of less utilised servers . . . 53
6.7 Additional Improvements . . . 55
6.7.1 Build System . . . 55
6.7.2 Server Deployment . . . 56

7 Future Work **57**
7.1 Actual Relocation . . . 57
7.2 Process Grouping into logical Application . . . 57
7.3 CPU Load Normalization . . . 57
7.4 Refine Auto Relocation of Consolidator . . . 57
7.5 Different Process Costs . . . 58
7.6 Security Countermeasures . . . 58
7.7 Intelligent Process Tracking . . . 58
7.8 Remote Tracking of Processes . . . 58

8 Concluding Remarks **59**

9 Appendix **60**
9.1 Consolidator User's Manual . . . 60
9.1.1 Functionality . . . 60
9.1.2 Requirements and Shipping . . . 60
9.1.3 Configuration of Consolidator . . . 60
9.1.4 Server Configuration . . . 61
9.1.5 GUI Overview . . . 61
9.1.6 Visualisation as Chart of Processes . . . 63
9.1.7 Manual Relocation of Processes . . . 64
9.2 Population of database . . . 64

Abbreviations **66**

Bibliography **68**

List of Figures

2.1 A process with its environment variables. 7
2.2 The parent process passes its environment to the child process. 8

3.1 Papillon Component Relationships 11
3.2 Papillon Dashboard 14
3.3 The basic work flow of WattApp consisting of three steps. Adapted from [KVN10]. 17
3.4 The data collection phase of Powerscope. A power profile for the left machine is created. The machine on the right saves the measurements from the multimeter. Adapted from [FS99]. 17
3.5 Energy consumption measured in Energy per transaction. The optimum point occurs in 70% Central Processing Unit (CPU) and 50% disk utilisation, taken from [SKZ08]. 20

4.1 The logical group of three processes formed by their shared environment variable PAPILLON_TAG. The Client should report all of these processes as one entry with a total sum of 130. 25
4.2 The new work flow of the process reporting mechanism. 27
4.3 The basic architecture of the Consolidator including the used libraries with their tasks. 28
4.4 The work flow of the newly created component Consolidator. 29
4.5 A pair of Hosts with multiple processes. All processes have an equal share of CPU cycles. 32
4.6 Process Gamma was moved from Host 1 to Host 2. 33
4.7 A basic class diagram of the Consolidator Software 36
4.8 An Application Programming Interface (API) call example: get information about a host. 37

5.1 Modifications in code concerning MyBatis functionality. Classes with white background were newly created. 41

6.1 A chart from a process that was quite cost intensive. This can be seen in the CPU contingent value. 52
6.2 A chart from a process that produces less load than the one in figure 6.1. 52
6.3 A chart from an artificial process entry. On host 300 the CPU load is always 30%. 53
6.4 A chart from an artificial process entry. On host 900 the CPU load is always 90%. 53

6.5 The two newly introduced process metrics: *CPU share* (Relative CPU) and *energy share* (Total Energy). The upper lines show host information whereas the lower lines show process information. 54
6.6 Manual relocation of different processes. 54
6.7 Automatic relocation of processes. 55
6.8 The structure of the deployment tool set of Server. The main script takes three components and produces a compressed package that can be executed on the server. 56

9.1 Overview of Papillon GUI 61
9.2 The basic structure for the scripts that populated the database with artificial values. 64

List of Listings

4.1 The difference of two example processes' environment variables. 24
5.1 The Extensible Markup Language (XML) structure of the newly created processes API call 42
5.2 Proper start for Windows Operating System (OS) in order to supply the process prime95.exe with the correct Environment Variable (EV) PAPILLON_TAG with value *MYTAG* 45

List of Algorithms

1 Auto Relocation Algorithm 34
2 Colour Calculation Algorithm 48

Acknowledgements

I would like to thank my advisor Christian Steger for giving me the opportunity to visit this beautiful island - Ireland. At the University College Dublin I was greatly supported by Damian Dalton and Abhay Vadher. I wish them all the best and much success with their company.

Although I spent most of my semester abroad at the office, to conduct my thesis, there was enough time to make new friends. The Christian Union at the UCD were a gateway to a lot of great people. I am really thankful that I have not only met but connected with them.

Another great bunch of friends that should be mentioned here are the ones that I found in Graz, which was my place of residence for five years. I hope that I can keep these friendships and always stay in touch with this nice and beautiful city.

I am very grateful for the support I received from my family. They always taught me to believe in myself, be patient if things might look bad and assured me of their love and support - this and many other reasons make me a proud son of Johanna, brother of Claudia, Carmen, Tanja and Silke, brother-in-law of Thomas and recently the uncle of David and Simon.

For me the most important person on this earth is my lovely fiancée and best friend Priska. From the people around me it was she who carried most of the burden by constantly giving me support, friendship and advice. I am really a lucky and thankful man for being with her and that, starting from August 2015, I am allowed to call her my wife. Our engagement took place in our home town Innsbruck during a short Christmas holiday - another reason why we will never forget this time in Ireland.

The last and most important being I owe thanks, and even more than that, is my heavenly father, his son Jesus and the Holy Ghost. It is pure grace how my life went and this thought gives me strength and fills me with love towards him. May everyone who reads this work be inspired because, I am sure, without God it would not have been possible.

Matthias Vierthaler
Graz, Austria, May 2015

1

Introduction

This chapter describes the motivation behind several topics that go along with this thesis. Besides stressing the importance of energy aware server machines it explains why consolidation and demand analysis on process level is an interesting feature. After explaining the goals of this thesis the outline shows the structure of the thesis.

1.1 Motivation

Cloud Computing is a relatively new field to computer industry and introduced a new form of computing and a wide area of new business models. The core requirement behind the cloud are data centres - although they were always necessary for networked computing the rise of cloud applications resulted in even greater centres. Naturally with the increasing amount of energy that is used inside data centres its share of the global energy demand is growing as well.

It is surprising that *Green IT* - the efficient use of resources concerning information technology - has just been researched in the last two decades. According to [Rut09] improvements in Information and Communications technology (ICT) can lead to positive development in many other fields due to the dependence on IT solutions. Another reason for the ongoing research is that costs for cooling are increasing [Sca06]. And with greater computation power come greater cooling requirements. Not only that this is a financial question. Cores are overheating and measures are taken to expand their lifetime. Not only blackouts (complete outages) are a problem but also brownouts (temporary reductions of energy supply) [Ver+10]. Given the fact that most data centres rely on a stable connection to the outward world these problems have to be addressed thoroughly.

There are quite some features that help using less energy in ICT. However, these approaches are mainly used in consumer hardware and most often can not be applied to servers due to the bad prediction of requests in this field. Another interesting fact is that the amount of power consumed by idle (not utilised) servers is about 65% of their peak consumption [Gre+08]. It has to be kept in mind that only decreasing the utilisation of a server will not necessarily result in actually using less energy. Combined with the fact that most servers spend most of their time utilising between 20% - 30% of their computing power [BH07] servers need to be monitored and, if possible, less utilised servers need to be consolidated, or suspended, so that higher utilised servers can take their work load and perform on more energy efficient levels.

Stratergia Ltd. [1] is an Irish company which goal is to measure and analyse the power demand of servers in data centres. Their product, called Papillon, calculates the server's power values by creating a power model per server group and using that model for later measuring. So

[1] http://stratergia.com/

far their system offers system wide values - meaning that single processes are not analysed at all. This makes consolidation difficult as it relies on the knowledge of the running applications on that machine. In addition to that there is no visualisation of relocating processes in order to ease the consolidation of "emptied" servers.

Apart from the consolidation the impact of every process is interesting to data centre operators who bill their customers not only per server but also per application.

The motivation for this thesis is firstly to analyse the existing Papillon system for shortcomings regarding the process analysis and consolidation. After that the tool set should receive a feature that eases consolidation of servers, visualises the monitored processes and gives deeper insight into the running applications. Their impact should be explained and calculated.

1.2 Goals

The goals of this thesis are to help the users of Papillon consolidate as many servers as possible. This is achieved via following objectives:

Process Tracking: In order to understand what processes are actually running on the server Papillon needs to know what applications are reported. Although there is simple process reporting the requirements of constant and reliable monitoring of processes are not met. The previous mechanism monitored only top applications according to the used CPU time. This is insufficient for distinct reporting of single processes.

Process Classification: In order to consolidate and know the energy demand of an application the user has to know the influence of a process over a certain amount of time. At least one metric has to be defined that is dependent on the load of the server machine. The metric has to give the user information regarding the used resources on the server machine.

Process Grouping: It is unknown to the Papillon system what processes belong together. A method should be found that helps connecting processes with each other but keeping the modifications as small as possible. A typical problem that needs to be solved is if two processes belong together but lose their relation because only one of them is actually monitored. This problem should be addressed.

Process Visualisation and Manual Consolidation: A way should be found that helps the user visualise the monitored processes. A Graphical User Interface (GUI) should be implemented that helps a customer relocate processes to other hosts. These relocations should be applied to past data and help the customer try out consolidation possibilities. The constellations should help the data centre operator to consolidate as many servers as possible and supply him with the knowledge of where processes are executed.

Automatic Consolidation Proposal: A automatic algorithm should be implemented that speeds up the relocation work flow. By proposing an optimal constellation with the maximum amount of consolidated servers the data centre operator should be supported in minimising the amount of needed machines.

1.3 Outline

After the motivation and goals, chapter 2 holds an introduction to used metrics, energy demands in server systems and some examples of power saving methods in computer systems. The chapter is concluded by environment variables, which will be used later on and the definition of the comparison of servers. Chapter 3 starts with an overall explanation of the Papillon system this thesis is based on. Then several shortcomings are listed and related work with linked or similar goals are shown. Chapter 4 describes the design decisions necessary to solve the listed shortcomings including new work flows and newly created concepts. Chapter 5 shows the implementation of the modifications and the new component called Consolidator. The thesis is continued by results of the implementation in chapter 6 which is structured by a comparison of the listings of shortcomings from the beginning of the thesis and the actual solution. Chapter 7 gives ideas for future projects and chapter 8 summarises the thesis in an concluding remark. The appendix in chapter 9 contains an user manual for the Consolidator and the explanation for a database script that was used during the thesis - both sections might be helpful for users of the software.

2

Fundamentals

This chapter covers all necessary background that is needed to understand the figures and metrics used throughout this thesis. It defines used parameters, elaborates on energy demand and saving possibilities in server systems, explains the used network protocol and environment variables for following chapters and points out how server machines are compared in this thesis.

2.1 Metrics

This section describes used metrics regarding energy demand in this work and its references.

2.1.1 Energy - Watt Hours, Watt Minute, Watt Seconds

The term energy is used if work is done in any way. It is always transformed from one form to another, which possibly results in lower forms such as heat. The owner of a data centre normally deals with the transformation of electrical energy to computation power and its availability and the heating that is radiated by the electrical components. The official unit for energy according to International System of Units (SI) is Joule (J) which is equivalent to "lifting a body with a weight of one Newton for one meter" [Ham15]. The energy supplier however measures the consumer's demand in Watt multiplied by the Time unit, for example Watt Hour (Wh) or Watt Seconds (Ws). One Joule is equivalent to one Watt Second. Due to the importance of costs this metric is the most favourable one. This unit can not be measured instantly but calculated over certain amount of time. In order to allow measurements at the current time a different unit must be introduced - electrical power.

2.1.2 Electrical Power

In comparison to energy electrical power can be measured without any time frame. The unit for electrical power is Watt which is the product of current (Ampere) and voltage (Volt). Watt is therefore a value that is needed at the very moment of measurement. This is not helpful for measuring costs due to the fact that certain spikes in power need could change the actual energy demand.

2.1.3 CPU Utilisation

The utilisation C of a processing unit that was idle for t_{idle} and worked for t_{work} is defined as:

$$C = \frac{t_{work}}{t_{work} + t_{idle}}$$

Normally utilisation is measured in percentage ranging from 0 - 100, however this would only hold for a single core. If multiple cores are used there are three possibilities:

1. Different values for every core ranging from 0 - 100. This holds the disadvantage that comparing values gets more complicated.
2. One value ranging from 0 - ($100 *$ number of possible threads). In this case the user must know how many threads are available at the same time.
3. One value but normalized in order to keep a range from 0 - 100. If this possibility is chosen then the comparison between systems with a different amount of cores is not trivial.

Most used tools give a percentage ranging from 0 - ($100*$number of possible threads). Therefore this format is used in most parts. Throughout this thesis it will be clarified what format is used. After defining the typical metrics the next section describes energy demand specifically in server systems.

2.2 Energy Demand in Server Systems

This section explains the challenges behind measuring the energy demand in server systems and elaborates on the problems that arise in server architectures. There are three main concerns about server systems [BR04]:

High power consumption: Servers need increasing levels of continuous power supply - this is measured in Watt. The higher the levels the more difficult it is to offer a reliable power supply. With greater power demand come higher cooling requirements. The cooling, however, adds up to the higher power consumption and therefore accelerates this development. Another important issue is the higher power density due to new technologies that try to pack more transistors on the same chip area. This, again, leads to more power demand for the same physical area. The higher density is also applied to the racks that try to bundle even more servers due to the greater performance that is necessary for many networked applications.

High energy costs: As it was previously explained, energy holds a time unit - and in case of data centres energy is always connected with financial costs. Due to the high power consumption servers and the required cooling lead to increasing energy costs. A good measurement for this development is the Power Usage Effectiveness (PUE) - a value that measures the total amount of power divided by the power that is used by IT equipment. A data centre with optimal effectiveness would have a value of 1.0. According to Uptime-Institute [Upt15] that conducted a survey with 1000 participants around the globe the average PUE of 2014 was 1.65. Although this value is better than the one from 2011 (1.89) there is still room for improvement.

High consumption even in next-generation server clusters: At the time of writing the authors claim that next-generation servers will still keep their levels of high consumption. In addition to that they stress the importance of management tools for servers - which is interesting because this thesis is based on Papillon - a Data Center Infrastructure Management (DCIM). In addition to these three points [Gre+08] gives another very important aspect:

High demand when idle: The research group measured the power demand in idle mode. they came to the result that keeping the servers in idle mode still leads to power demands of 65% of peak levels. The previously mentioned survey [Upt15] brought special attention to these "comatose" servers because their high loads decrease the average PUE tremendously. This bad behaviour of servers during idle times is caused by the next characteristic.

A problem that is not limited to server hardware but can be applied to modern computers in general is the dynamic and static share of their energy demand:

Dynamic and Static Demand: Each server's energy demand consists out of a dynamic and a static part. The static part needs energy regardless of the workload whereas dynamic parts differ depending on the workload as it is stated in [OLG10]. Both parts, however, are dependant on the machine's configuration and architecture. This is the reason why idle servers still consume such high levels of energy.

2.3 General Methods in Use for Saving Energy

After explaining the complexity in measuring energy demands this section lists some well known tactics that are used in order to decrease the consumption in computer systems.

2.3.1 Dynamic Voltage and Frequency Scaling

A method that decreases the demand by scaling either the frequency or the voltage of a circuit. Bianchini and Rajamony [BR04] states "Dynamic Voltage Scaling (DVS) relies on the fact that the dynamic power consumed by the microprocessor is a quadratic function of its operating voltage". The scaling can be applied to multiple components like CPU, Random-Access Memory (RAM) and bus systems. Due to the fact that decreasing the voltage also decreases the frequency this method causes longer execution time. However if the machine had long idle times before the degradation those times are probably shorter afterwards because the system needs more time to accomplish its tasks. This obviously marks this technology as suited for server systems due to the long idle times.

2.3.2 Halting or Deactivation of Components

In order to deal with the static part of power demand components can be switched off. Bianchini and Rajamony [BR04] differs between *halting* where a CPU no longer executes any instructions and *deactivation* where the CPU is completely switched off and only reactivated by certain instructions. Similar techniques can also be applied to disks and network cards.

Of course halting/deactivation is more interesting to consumer products where requests are only executed by one person whereas in servers it is hard to predict when the next request arrives at the system.

2.3.3 Request Batching

Elnozahy, Kistler, and Rajamony [EKR03] introduced a method especially designed for web servers called *Request Batching*. The reason for this development was the finding that DVS did not yield good results and servers with constant requests can not be put into sleep mode so easily. The main idea is that during low loads the web server is kept in a low power state. If a new packet arrives during that time it will be accumulated. A packet will only be processed if it was stalled for a certain *batching time out*. This way the CPU could stay in low power mode

much longer. The research group proposed a variable time out that changed its value depending on the activity of the system.

2.4 RESTful API

The main communication of the Papillon system is done by Hypertext Transfer Protocol (HTTP) over Transmission Control Protocol (TCP)/Internet Protocol (IP). In spite of requesting and receiving arbitrary data streams it uses a Representational State Transfer (REST)ful API. A main characteristic of this schema is that the API uses the correct HTTP calls according to the selected resource. Papillon implements the following request methods:

- **GET:** Ask for a resource. This call is used if entities are requested.
- **POST:** Create a new resource, for example hosts or new floors of the data centre.
- **PUT:** Update an already existing resource, for instance if properties of a machine have changed.
- **DELETE:** Delete an existing resource. This can be used if a machine is not needed anymore.

Besides offering a standardized interface this pattern is a good choice for applications with many participants. This is because it requires the developer to pack all needed information in one request. The server on the other hand is not required to save any information of the requester - the protocol is therefore *stateless*. Another interesting fact is that the requester is obliged not to save any information which helps keeping the overhead slim. [Wik15c].

2.5 Environment Variables

Environment Variables (EVs) are content holders managed by the operating system. They are introduced in this section because they appear in latter chapters. Although they were introduced in Unix they can be found in Windows [SS615] as well as in Mac [App15] OS. There are a lot of environment variables that are needed by the OS. The most famous is probably the PATH variable that holds the information where the system should look for executables.

In figure 2.1 a possible configuration for a process called Alpha and its environment variables is shown. It holds three variables: VAR1, TEMP and USER.

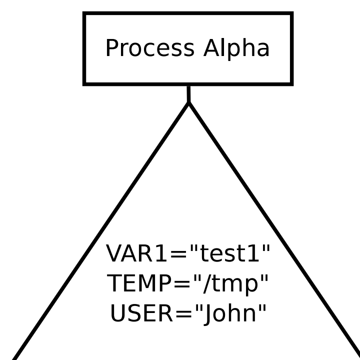

Figure 2.1: A process with its environment variables.

However there is a very interesting fact about EVs: they are inherited from the parent process. See figure 2.2 for an example. Process Beta inherits the environment variables from process Alpha.

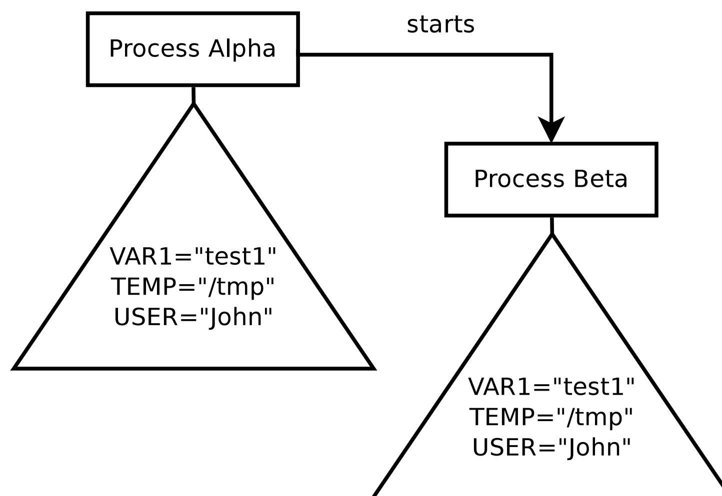

Figure 2.2: The parent process passes its environment to the child process.

This can be used in order to keep a connection between processes and form logical groups. It also gives one the possibility to supply independent applications with the same EV - this way even if none of them started the other an artificial bond between them is formed.

2.6 Comparison of Server Machines

In order to support the relocation of processes one has to know how two servers are compared. This gives the possibility to estimate the approximate impact this movement would have.

Although there are different server machines available in various environments it is assumed that two hosts are easily comparable. This should not cause too many troubles due to the fact that data centres normally serve one purpose and they tend to use machines with homogeneous properties. This yields to following rule: Given the fact that similar servers are used and run only one application and it caused an accumulated CPU percentage of $X\%$ on Host 1 then this value X is also added to the average CPU utilisation of Host 2 if that process would be moved to it.

The previous example only holds if there was only one process running on the source machine. CPU has to be divided if multiple processes are running which is nearly almost the case for server machines. For clarification a small example is given below: If host 1 has a accumulated CPU utilisation of 20% and host 2 in the same time 30% then a relocation of process Alpha running on host 1 with a CPU share of one half (contribution of 10% to overall sum of 20%) to host 2 would result in an calculated share of 40% on host2.

Of course it is harder to compare CPU values if different metrics are used as it was mentioned in section 2.1.3. This problem is solved by keeping to one metric - a value that ranges from 0 to $(100 * \text{number of possible threads})$.

3

State of the Art

This chapter describes several approaches to measuring power demands. Additional topics that came up during research are also discussed here. The first section explains the existing Papillon system. It is followed by a short discussion regarding the shortcomings of the current system which introduces the reader to the design decisions that are made in the next chapter. The following chapter can as well be found in Thomas Kriechbaumer's Masters Thesis [Kri15].

3.1 Stratergia Papillon

The work of this thesis is based on the existing Stratergia Papillon system. Stratergia Ltd., an Irish-based company developed the Papillon system and tries to improve data centre workflows and energy efficiency. According to their own definition[1]:

> Stratergia is the global leader in non-intrusive, meter-less, data centre power measurement and management systems.

The introduction and definition of the Papillon system and the mission statement can be obtained from their website[2]:

> Profiling and Auditing of Power Information in Large and Local Organisational Networks (PAPILLON) is a unique and innovative, comprehensive power management system, measuring server and application - level power consumption in continuous time without the requirement of any additional hardware. It is simple and intuitive to use, operates in any common operating system and can be downloaded and installed ready from the Stratergia website.

This section was jointly written by Thomas Kriechbaumer and Matthias Vierthaler, who both worked at the same time on individual parts and experiments in the context of Papillon.

[1] `http://stratergia.com/`
[2] `http://stratergia.com/papillon/introduction/`

3.1.1 Main Features

The needs of a data centre operator are spread over a broad band of DCIM functionalities and components. Papillon provides a uniform access pattern to the most important areas in relation to energy, efficiency, and infrastructure management in general. The main features can be outlined as follows:

- overview of registered servers
- current and past estimated power usage per server
- current and past top three processes with respect to power and CPU usage
- reports with different metrics
- give power related information without the need of an power measuring device

Since every component and feature build on top of a central database it is easily extendable and configurable to the specific needs of the data centre operator. In addition to the software solution provided by Stratergia, it is possible to create extensions utilising the available data to provide custom solutions.

3.1.2 Architecture

In order to keep a flexible approach that scales well with many servers a server-client approach was chosen. In this scenario a client is one of the machines whose power demands are in question. The client periodically sends its necessary utilisation characteristics to the server. This is also referred to as *active client*, since the client is the one sending data messages. The server is only receiving and storing the information in a database for later use. For each new client utilisation message the server computes the corresponding power value and stores them together with additional information about individual application usage metrics.

Every Papillon-monitored environment has one designated Papillon Server which keeps track of all clients and their power-related values. The communication between the server and client, as well as other components, is handled via a single interface which builds on a REST web API with HTTP as exchange protocol. This means that for the default HTTP calls RESTful API endpoints are implemented which handle `GET`, `POST`, `PUT` and `DELETE` according to the REST software architecture style.

Finally a user interface is provided in form of the Papillon Dashboard, which visualises the gathered information and manipulates database entries via API calls on the server. This component is optional and is normally installed on the same machine as the server software is running on. Since the API is well documented and accessible to everybody each customer can add functionality specific to their use case and data centre structure.

The relationship between these components is depicted in figure 3.1. In the currently deployed Papillon systems the clients only use a single API call to report data to the server. The server stores and manages data entries in the database and serves as data source for the dashboard or other user interfaces.

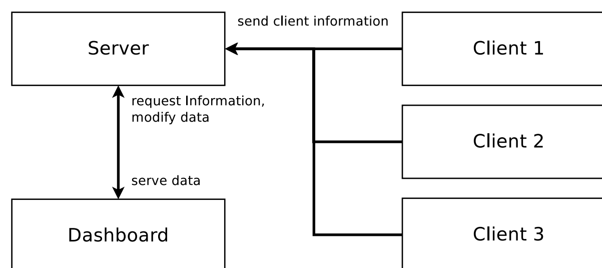

Figure 3.1: The relationships between Server, Client and Dashboard in the Papillon system.

3.1.3 Power Estimation and Application Power Consumption

Since it is cumbersome for a data centre operator to install physical power meters for all servers and devices the Papillon system provides a virtual power meter to monitor the devices. The power consumption for each server can be viewed in one minute intervals. Additionally to the total power consumption per server each data entry contains the applications running on individual servers. The total power consumption can be split up to indicate the shares contributed by each application. With this feature it is possible to compute a specific power consumption for individual applications and process groups, such as database, email, or web servers.

To allow Papillon to estimate the used power for a device, a power model is created specifically for this hardware configuration and server type. The first stage is to generate such a power model once before the server goes into live operation within the data centre. This power model is then used by the Papillon system to estimate the needed power based on live data collected from each monitored server. Since most data centres tend to have a very homogeneous hardware equipment and a limited list of server manufacturers the number of different power models which have to be created will be manageable.

3.1.4 Papillon Server

The Papillon Server is the central information hub in a Papillon environment. It provides an interface to the database entries and services API requests for host clients and other Papillon applications. Each Papillon Client sends data to the central Papillon Server which computes power values and stores all the messages in a database.

The server application is a Java-based web application deployed to a single Apache Tomcat [Pro15a] instance. The back-end database store is accessed via an interchangeable Java Structured Query Language (SQL) adapter and uses MySQL in the default installation. Since Apache Tomcat is available on all modern platforms the Papillon Server can be deployed to almost all OSs and web-capable platforms. The interface provided by Apache Tomcat is an open source project which exactly fulfils the purpose of executing Java code an a web-based architecture.

The publicly documented API accepts either JavaScript Object Notation (JSON) or XML data messages and is capable to distinguish between those formats on a request-by-request basis.

The currently implemented Papillon Server API does not support authentication or authorisation. Therefore it is necessary for the data centre operator to provide a suitable security model on top of deployed Papillon system. This is likely to change in the upcoming versions as development advances.

3.1.5 Papillon Client

The Papillon Client is a small application installed on every monitored host. Its small footprint does not affect normal operation and does not require special privileges. Running as background service allows it to silently collect usage information and report it to the Papillon Server.

Usage data is gathered every minute for individual metrics and each sample is sent to the server automatically. Therefore it must be supplied with necessary configuration about Papillon environment:

- domain name or IP address of the Papillon Server
- TCP port on which the server is listening
- host ID, individual to each server

Some usage metrics need higher granularity on their reading, therefore the Papillon Client may perform a small amount of work more often then the regular one-minute interval. Since the only communication between client and server is done via the public API, there is no limitation if both are installed on the same machine. It is therefore possible for a Papillon Server to monitor itself as well as other servers only running the Papillon Client application.

Similar to the Papillon Server, the client is designed as Java-based application. This allows the data centre operator to monitor Microsoft Windows, Linux-based, and Mac OS X-based systems in a unified infrastructure using the Java Virtual Machine.

Due to the regular monitoring interval the Papillon Client is expected to run during all times a server is operating in the data centre. This will generate a steady stream of usage information stored in the Papillon Server and can be accessed and analysed at any time by the data centre operator.

In addition to the main usage metrics a set of application-specific information is recorded and stored together with the regular entries. Currently the three CPU-heaviest processes on the system are recorded and transmitted to the Papillon Server to indicate which application cause power spikes or are responsible for high power consumption periods.

3.1.6 Papillon Database

The Papillon Server is the only component which requires direct access to the database. All data manipulation operation pass through an Structured Query Language (SQL) interface to the MySQL database management system running in the Papillon environment. Since the queries are all defined as prepared statements the actual database system could be swapped out for a different engine, such as PostgreSQL or MariaDB using a different Java database connector library.

All data entries are stored in a relational database. Depending on the access pattern various indices are added to improve performance and querying speeds. Since many hosts are monitored at the same time it might be necessary to implement sharding to reduce performance bottlenecks. The basic database schema contains the following data models and tables:

Data Centres, Floors, Racks provide a simple way to topologically group multiple hosts into individual subgroups based on the actual physical placement inside a data centre.

Hosts are servers with an active Papillon Client running and reporting data back to the Papillon Server. The *host ID* from this database entry must be configured in each Papillon Client to identify the data messages during transmission.

Power Model Groups and Power Models store the information necessary to estimate power based on activity metrics. These power models are created and grouped by different server manufacturers, server models, and hardware configurations.

Activity and Application Usage Entries contain all the activity information a Papillon Client has transmitted to the Papillon Server on a regular 60 second interval. These values are used to estimate the power value with the power model assigned to the host. Each entry is timestamped and stored in the database after the power estimation finished.

3.1.7 Papillon Dashboard

Since every monitoring system needs an interface to visualise and present the collected data the Papillon system provides the Papillon Dashboard as default user interface for data centre operators. It provides easy access to all management functionalities and database access for basic add, modify, and delete operations. Since the dashboard uses the public API of the Papillon Server it is easily exchangeable for a customised dashboard or even completely different application sitting on top of the API.

The dashboard can be used to add new hosts and view the power consumption and application-specific power values over a selectable time period. Different evaluation methods and reports can be created and downloaded via the browser. It is an optional component that is not required for regular operation, but acts merely as a user interface. The dashboard is a single-page web application and uses AngularJS and Bootstrap as web frameworks. The key features provided to the user are:

- manage infrastructure in data centres, racks, floors, and hosts
- manage power model groups and power models for different hardware configurations
- assign hosts with power models
- show inactive hosts and hot-spots
- render a power consumption graph per host and applications
- give visual feedback regarding the power value of hosts

Figure 3.2 shows the host administration page. This page provides information about all Papillon-monitored hosts and power model mappings. The *host ID* can be acquired from this list to configure individual Papillon Clients. Basic search and filtering functionality is accessible via the top right corner menu. On the left an access menue to various other data models is visible.

Papillon Beta Home Reports Admin

ADMIN
Heartbeat
Datacentres
Floors
Racks
Hosts
Powermodel Groups
Powermodels
Upload Datacentre

Hosts

Datacentre Select datacentre

Type to start searching Add Host

ID	Floor	Rack	Host	Host Type	Host Description	IP Address	CPUs	Model Group	VMs	
299	1	9	host-5811	SERVER	customer-04	192.168.0.99	2	Dell-PowerEdge-SC14XX-V1.0	N/A	Edit Delete
300	1	21	host-0821	VM_SERVER	customer-12	192.168.0.100	1	Dell-PowerEdge-R711-V2.0	4	Edit Delete
301	1	42	host-3891	SERVER	customer-32	192.168.0.101	1	Dell-Latitude-D820-V1.0	N/A	Edit Delete
302	1	42	host-9203	SERVER	customer-42	192.168.0.102	1	Dell-Latitude-D820-V1.0	N/A	Edit Delete

Copyright © Stratergia Ltd. All Rights Reserved 2015 : v0.0.9-dev

Figure 3.2: The dashboard shows the host administration page for the data centre operator to manage, add, and modify host entries in the Papillon database.

3.2 Shortcomings of the Existing System

Due to the fact that this work should be based upon the Papillon system this section discusses the shortcomings and desired features at the time of the start of the thesis.

3.2.1 Process Tracking

This section deals with the problems concerning the monitoring of processes. Two groups can be formed:

1. A machine runs only processes with unique names. This is not very likely but can happen if servers execute a limited set of applications.

2. A machine runs processes and at least two of them share their name. This can happen, for instance, if an application started two worker threads. In that case they probably have the same name.

3.2.1.1 Processes with unique names

As mentioned in section 3.1 the Papillon system's main task was to collect data from agents, analyse their current demand for various parameters and to give a estimation based upon a mathematical model that had been created beforehand.

From a customer perspective the value of this measurement would be increased if not only single server machines can be analysed but also single applications. The Papillon system already offered a simple mechanism by letting agents send their top three applications. This lead to a few considerations:

1. If a user wants to have a quick glance at the current usage the top three applications are sufficient. He would have to deal with changing entries but nevertheless he receives immediate feedback.

2. Following the progress of an application over a certain amount of time is hard because

   (a) applications could be left untracked due to the fact that they don't use enough resources in order to be recorded as one of the top three applications.
   (b) if an application has varying participation in the CPU load it would be hard to realize that because they could have only occurred very seldom.

3. The cost of an application is impossible to determine because the system is not able to guarantee a continuous dataset of the queried application.

3.2.1.2 Processes with same names

Another substantial problem regarding the process analysis is the fact that two processes with the same name are not taken into account. Here, again, different considerations arise:

1. If process A1 would be part of the top three group and process A2, with the same name, is not then:
   (a) only process A1 would be recorded all the time. Process A2 would not be recorded at all.
   (b) a user would have no knowledge of the existence of the second process A2 with the same name.
   (c) if A2 would suddenly increase its load and became a member of the top three groups new problems would appear - see next point.
2. If process B1 and process B2 with the same name would be part of the top three group then:
   (a) only one of the processes will be recorded. The other one will be left out. This leads to inconsistent database entries.
   (b) it is very likely that not the same process is recorded all the time - this would result in distortion of the recorded data.
   (c) a user, again, would have no knowledge of the existence of the double names.

3.2.2 Process Grouping

The Papillon system is able to report single processes. However it does not support any information about the relationship between the reported processes. There can be three different relationships between two or multiple processes:

- Not connected in any way. For example a web browser and a text editor.
- Process One started process Two and they depend on each other in some way. Example: Worker Threads started by Main Thread in Graphical Processing of an Image.
- Whether no process was started by the other one yet they still depend on each other. Example: Apache Web Server depends on running Database Management System like MySQL.

In some cases it would be interesting to know about these relationships and the combined costs. Nonetheless the version of the system that was handed to the author does not offer any of those.

3.2.3 Process Visualisation

The Papillon dashboard gave simple information regarding the top three applications. If tracking single applications over a longer period of time was needed then this kind of visualisation was cumbersome. Although it offered the possibility of listing general information regarding the connected data centres it did not show statistics and impact of single processes.

3.2.4 Process Costs

Another shortcoming was the lack of knowledge of the influence of each process. A process' fraction of energy and therefore financial costs is important to most data centre operators. This point also depends on proper tracking functionality because gaps in the data points would lead to inconsistent calculations.

3.2.5 Consolidation of less utilized Servers

Although there was a report-functionality that helped finding less and more utilized servers it did not give any understanding of the monitored applications and their impact of the overall demand. There was no possibility to find out how the relocation of processes would affect the concerned servers. Especially in connection with process tracking and other process considerations a flexible solution that aids in relocation would be helpful.

3.3 Related Work

The rest of this chapter describes other approaches and gives a short description about them. They are grouped in logical work packages, depending on their original goal.

3.3.1 Power Measuring Systems

This section covers research projects that have the similar goal as Papillon: measure the energy demand of servers. In order to give new ideas about energy analysis on process level this section focuses on papers that specifically deal with process monitoring.

3.3.1.1 WattApp

Koller, Verma, and Neogi [KVN10] try to analyse heterogeneous applications in order to build up a better power model - this includes the use of the parameter throughput that helps classify applications. The authors claim to create a power model that is independent from "a specific class of application". Every application results in its own model by performing multiple test runs. The resulting power models are usable for one specific application. The work flow after the power model is finished is depicted in figure 3.3. After the Application Profile is created based on its throughput by the *App Profiler* a *Placement Generator* creates a placement and asks a *Power Modeler* for an accurate power estimation. This request is used by the placement controller to validate this placement with the given server specifications and the needed performance. After this check is done a final placement is returned.

Although the authors claim not to focus on any type of program they need it to be measurable in throughput. Despite their approach results in fast and accurate (5% error) measurements it is not feasible for Papillon due to its requirement to monitor any kind of application.

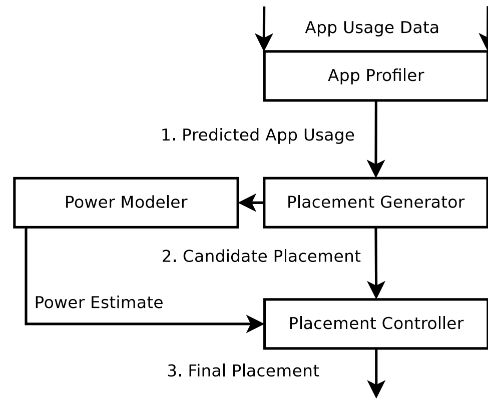

Figure 3.3: The basic work flow of WattApp consisting of three steps. Adapted from [KVN10].

3.3.1.2 Powerscope

Flinn and Satyanarayanan [FS99] try to "profile energy consumption to program structure, in much the same way that CPU profilers map processor cycles to specific processes and procedures". Powerscope is executed in two stages. The first stage is needed for gathering data and power values - this is done by using a modified Berkeley Software Distribution (BSD) OS. These modifications include reporting mechanisms that are started each time process related commands like *fork()*, *exec()* or *exit()* are used. figure 3.4 shows the test environment for that stage.

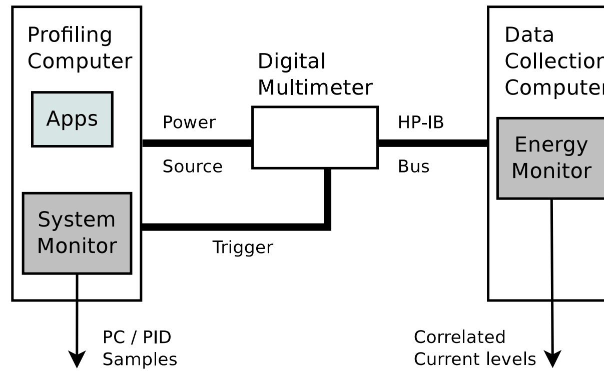

Figure 3.4: The data collection phase of Powerscope. A power profile for the left machine is created. The machine on the right saves the measurements from the multimeter. Adapted from [FS99].

In the second stage the combination of Process Identifier (PID) and the trace of the mentioned commands results in detailed Energy Usage Statistics of user programs. By using their tool they analyse a video application and, by using that knowledge in the application's source code, succeed in decreasing their energy consumption by 46%. Unfortunately their approach is not applicable on Papillon because no modified OS kernel shall be used.

3.3.1.3 Koala

The Koala system described in [Sno+09] is based upon [SPH07] that uses various performance counters which are available in CPUs to predict the current energy demand. Two models are used: a time and an energy model. The time model specifies a waiting time for the end of an execution. The energy model, however, uses this time model to predict the energy demand under different assumptions. Not everything is modelled - for instance the Input/Output (IO) is ignored. Performance counters are included that estimate the waiting time for bus and memory. Example events that are linked with the counters are stall cycles and cache misses. The OS was altered so that it monitored the performance counters regularly.

Koala extends the previous work by not only viewing the process' impact but also by offering policies that help managing the system's power behaviour. Target functions are minimum energy and maximum performance. The approach is solely based on Linux OS and is therefore too narrow to be used in Papillon's work flow.

3.3.1.4 ECOSystem

Zeng et al. [Zen+02] manage "Energy as a first class Operating System Resource". Their approach lets the OS decide how much resources a process is allowed to receive. If programs use either processing time, disk IO or network communication a unified model sums up their demand.

Resources are counted in *currentcies* standing for a mixture of electrical current and currency. Their goal is to maximise the battery lifetime of a mobile device. Therefore several subcomponents of a laptop are assigned to certain energy consumptions. The authors explicitly give countermeasures against wrong calculations that result from various contributions to the overall power demand of different processes. If process A uses the CPU and process B the Network Interface Controller (NIC) then their model perceives their different contributions as such. One *currentcy* can be understood as a guarantee for a certain amount of Joules with the difference that it carries an expiry date. The amount of available *currentcies* is calculated per formula that depends on the desired battery life and assigned to each time slot. If multiple processes compete with each other a user-specified properties mechanism divides the remaining power units between those processes.

A Linux OS was changed so that it would follow the mentioned guides - this includes a new kernel thread that wakes up periodically and inspects the consumed *currentcies* per process. If a process' balance of the current time epoch is zero or negative the process has to sleep until the next epoch. The three devices CPU, disk and NIC are analysed. The research team extended their work in [ZEL05]. The approach is not suitable for Papillon System because it again depends on the modification of a Linux system and would not support Windows OS at all.

3.3.2 Migration of Virtual Machines

A field that originates from around 1970 where computers were rare and multiple users had to share one is the concept of Virtual Machines. Such a machine runs like a process on the physical device - for the outside world there is no difference between a virtual and a physical server. However, cloud computing and the connected demand of greater data centres support the heavier use of this technique. If hosts are started virtually they can be suspended and relocated quite easily. This leads to the question of migrating Virtual Machine (VM)s to different servers thus consolidating the source hosts. Thanks to the good portability of virtual hosts a server can probably be used more effectively if most of its work is done by virtual guests. Due to the

fact that this topic is related to the motivation of most consolidation research a few papers are mentioned here as well.

3.3.2.1 Live Migration of virtual machines over MAN/WAN

Travostino et al. [Tra+06] show how VMs can be seamlessly moved over a Wide Area Network (WAN). This is useful for load balancing between different data centres or in a case where a VM needs to be evacuated in case of a future failure. The authors know that their proposed goal has certain requirements like a stable broad band connection between the source and target hosts and is not safe from security attacks. In order to make sure that the relocation is possible available resources have to be checked beforehand. A key component is the *Xen* [Fou15a] environment that supports migration commands between hosts. The team created multiple components that took care of the connectivity, the security by requesting a token from the target host and the pre-allocation of necessary resources for the VM.

Although there was a round trip time in WAN that was approximately 1000 times higher compared to the same request in Local Area Network (LAN) the caused application downtime was only 5 to 10 times higher. The experiments were conducted on a relocation between San Diego in U.S.A. and Amsterdam in Netherlands.

3.3.2.2 High Performance Migration of Virtual Machines

Clark et al. [Cla+05] try to migrate a VM with as little downtime as possible. They use various techniques that help clients not to lose any service that is offered by the migrated machine. Their main motivation is enabling administrators to separate software and hardware from each other. The network is expected to be LAN and the pages of the VM are copied in steps via a self called method *pre-copy*. This technique starts by pushing the pages that are not modified frequently (calculated by an estimation algorithm) from the source to the target machine. The transmission happens in rounds so that the caused traffic is not too demanding, after transmitting the majority of data the old VM is suspended and the new one receives the last state of CPU and NIC - thus keeping the last state. After that an Address Resolution Protocol (ARP) request is broadcast to all the connected clients in order to inform them about the new Media Access Control (MAC) Address. The research group did not take local hard disk into account but only network attached storage (NAS).

The exact step of the algorithm are as follows: A target host is selected. The selection is followed by a request that informs the target host about the migration that is about to begin and, in addition to that, checks if enough resources are available. Afterwards the *pre-copy* process starts sending all pages that are unlikely to change to the target host. The source host keeps operating in normal state. After all rounds are finished the source host is suspended and all remaining pages that were changed too often to be copied in running state or became inconsistent on the target host are transferred. The target host starts up when all the pages are consistent and reports to the source host about the successful transmission. Subsequently reattachment of devices and the advertising of the new IP Address is done. The team tested their prototype on various environments: on a simple Web Server they achieved a downtime of 165 ms while using around 80 seconds for transmission. A game server and a complex Web server produce similar downtimes. The only environment that results in greater downtimes are servers that change data faster than their network speed. This way too many pages become inconsistent and have to be copied again.

3.3.3 Consolidation of Processes

Another research field is the consolidation of processes. The goal is to distribute a given set of processes on a certain amount of servers. The main goal is to keep the energy demand as small as possible. This does not necessarily mean that one has to keep the amount of servers as small as possible. This is due to the fact that machines which are not utilised still have a great power demand.

3.3.3.1 Energy Aware Consolidation for Cloud Computing

Srikantaiah, Kansal, and Zhao [SKZ08] conducted basic research and coined some important terms. Major goals are to consolidate different processes onto available servers thus minimising the energy demand. They carried out some tests and used a metric defined as *energy consumption per transaction* which is influenced by resource utilisation and performance. This helps understanding the non-linear behaviour of power demand of servers as it is depicted in figure 3.5 that was taken from their paper. It was previously stated that weakly utilised servers normally

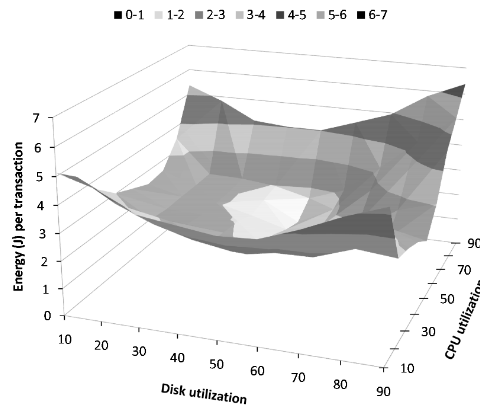

Figure 3.5: Energy consumption measured in Energy per transaction. The optimum point occurs in 70% CPU and 50% disk utilisation, taken from [SKZ08].

still use around half of their peak power level. The research group however also states that very highly utilised servers results in performance degrading and longer execution times. Their work focusses on CPU and disk values and claims a optimum point of 70% CPU and 50% disk utilisation. However, they also say that this optimum point depends strongly on the system and has to be measured every time. After explaining the challenges they describe the consolidation problem as a multi dimension Bin Packing Problem (BPP). Servers are understood as multi dimensional bins (CPU, disk, network,...) and applications as objects in which their size is their needed resources. The goal is to pack as many objects into a bin and keeping the amount of bins as small as possible. The maximum size of a bin is the previously mentioned optimum point. Two differences are that firstly too many objects in a bin could cause performance degradation and a solution where one bin ends up having the last object is no ideal solution. Sizes are measured in Euclidean distance to the optimal point. This scalar metric is applied to two dimensions (CPU and disk) but is defined for even more dimensions. Their heuristic algorithm receives the applications, tries to assign them to the available bins by using the Euclidean distance from the optimum point of each server and moves on to the next application. Assuming that no servers have free resources left a new server is started and the allocation is restarted. Their algorithm does not yield the optimal solution but serves as a prototype implementation. The authors discuss that they have not considered multi-tier applications, server heterogeneity and application feedback. Still the basic concept of the paper is valuable.

3.3.3.2 pSciMapper

Zhu and Agrawal [ZA10] focus on an application group called scientific work flow that tries to cluster all tasks that are used by most scientists. The research team claim this application group occurs regularly in the fields of astronomy, bioinformatics, physics and seismic research. These work flows describes the progress spanning from dataset selection to computation until visualisation. Due to the typical size of the reviewed projects the goal of the research team was to consolidate multiple tasks from the work flows by splitting and merging them. Firstly a resource usage profile is created for every task by training a Hidden Markov Model (HMM). Afterwards dissimilar workloads are merged in order to utilise a machine completely. Another reason for this design is that similar tasks would result in higher stalls for that component - e.g. memory. While forming the clusters the Nelder-Mead-Algorithm is applied that maps the groups of tasks to actual servers. This algorithm is a heuristic search method that, in that case, tries to minimise the consumed power and furthermore to find the best trade off between execution time and energy demand. The research group use 5 scientific work flows as test data to check their impact. They claim to decrease the power consumption by 56% while keeping the maximum slowdown at 15%. Given the fact that Papillon is not only applied to different applications Zhu's approach is not suitable for Papillon.

3.3.3.3 V-MAN

Marzolla, Babaoglu, and Panzieri [MBP11] create a decentralised system with the goal of consolidating servers by gossiping. It focuses on VMs but the principle can also be applied to processes. The idea is that huge data centres with several thousands of servers are expected to run a certain amount of VMs each. If a central unit should keep track of the currently running machines it could become a bottleneck. This would result in degraded optimisation and affecting the stability. Therefore they propose a system that consists of hosts that communicate independently with each other but only know a small fraction of the available servers. The researcher expect a maximum amount of VMs running on a machine and that every one of them can be moved to another physical device without attached problems. Furthermore although every host is probably connected with each other via network their protocol expects a host only to talk to just a few servers. Every physical host runs two algorithms: the first algorithm broadcasts the current number of running virtual machines to all connected nodes. The second algorithm checks the number of running VMs in its view and makes sure that the server with the smallest amount of guests pushes some guests to the server with the highest amount of guests. Thus consolidating the one with the smallest number of hosts after some iterations. They test their approach by using a virtual network which results in a good convergence up to the previously calculated optimal point of servers. Furthermore, because of the decentralised approach, the system can tolerate the outage of servers quite easily.

The main idea of V-MAN, keeping the migration decentralised, is interesting but not helpful to Papillon which uses a centralised structure. However, the approach can be used as an extension if data centres get too large to handle.

The list of related projects concludes this chapter, chapter 4 introduces the reader to the solutions and design decisions that deal with the listed shortcomings of Papillon.

4

Design

This chapter describes the design for the new features of the system and the modification that are planned. It is based on the Papillon system and will expand the already existing functions using best practises discussed in the previous chapter.

4.1 Specifications

This section describes the basic specifications for further decisions. The already existing Papillon system should be used as a basis for additional software packages. The shortcomings described in section 3.2 are split into three parts.

4.1.1 Process Analysis

Tracking: In order to help the user track applications, following extensions should be created:

- Give insight what processes are having the most impact on the power demand.
- Tracking of processes via names.
- Tracking of processes regardless of their name.

Grouping: In order to solve the problem described in section 3.2.2 a new strategy should be proposed that helps reporting relationships between processes.

Cost function: A cost function should be proposed that helps understanding the connection between overall power demand and a single process.

4.1.2 Manual Process Relocation

In order to consolidate less utilized hosts a subsystem called *Consolidator* should be created that

- shows costs of processes over a certain amount of time,
- allows moving processes around via GUI thus creating a new configuration of servers and processes,
- checks if a new configuration is feasible,

- and prints new configurations to an exportable format that can be used by a data centre operator.

4.1.3 Automatic Process Relocation

The subsystem should hold an automatic process relocation with following possibilities:

- A user can give certain constraints about the relocation.
- Manual intervention should still be possible.
- An optimal setting should be calculated.

After defining the specifications the rest of this chapter explains how their expectations are met.

4.2 Process Tracking

Process Tracking is the action of monitoring specific processes. The question which programs are recorded and how they are tracked are to be answered.

4.2.1 Name Tracking

The Client wakes up every minute, besides calculating the system utilization for network, CPU and hard disk it analyses the processes which have been running during the preceding minute. There could be a huge number of running processes, therefore not all but only certain programs are reported. So far the system only informed the user about the top three applications based on the used CPU cycles.

A first extension is tracking by name. Processes are tracked if their name appears in the configuration file of the Client. The file can hold an arbitrary amount of names of processes that shall be tracked. The next section explains what should be done if two processes share the same name.

4.2.2 Unique Process Names by using Hash Function

It was already established in section 3.2.1.2 that the process tracking needs refinement due to its dependence on unique names. In order to track processes with same name a hash function should be used in order to produce two distinct identifiers. A hash function takes an arbitrary input of variable length and produces an output with a fixed length - it is very unlikely that two inputs produce the same hash output - this is called a conflict. The hash function could be applied on the name but then two processes with the same name have the same hash value and nothing is gained. Additional parameters have to be used.

The SIGAR library [Hyp14] that is used for getting system information offers a high amount of different parameters. In the following list the most appropriate parameters are listed and explained briefly:

- `name`: The name of the program - it is important to note that certain programs change their name after they have been started.
- `environment variables`: Contain all EVs for that process. These type of variables are used by the OS. Please refer to section 2.5 for further explanation.

- `Process Identifier` The PID is a number created by the OS that uniquely numbers a process. Numbers are lost if either the process or the OS is restarted.
- `arguments` The parameters the program was started with. Simple programs have only a few parameters while complex applications like a Java environment could include much information in their arguments.

Now every parameter is considered to be the designated hash input:

The name should always be part of the unique identifier of a process because a user will probably identify the name before any other parameters. Hashing the name would make it unrecognisable.

The environment variables do not have to differ necessarily, see following example: On a Linux Operating System the Eclipse Integrated Development Environment (IDE) is started - Eclipse is executed through a Java process. After starting the Consolidator (the new subsystem, mentioned in section 4.1.2) through Eclipse two processes with the same name *java* are running. Both have a set of environment variables with a size of around 1 kilo byte. One would expect both environments to be different but actually they differ only in the lines shown in listing 4.1.

```
1 NLSPATH=/usr/dt/lib/nls/msg/%L/%N.cat
2 XFILESEARCHPATH=/usr/dt/app-defaults/%L/Dt
3 LIBOVERLAY_SCROLLBAR=0
4 SWT_GTK3=0
```

Listing 4.1: The difference of two example processes' environment variables.

Apart from that they are equal. Which is another problem because considering the environment variables as difference would result in different entries if they were started in different environments - which is not always desired. If the Consolidator is not started from Eclipse but from a terminal the difference is larger but the point remains valid: environment variable differences between two processes are hard to predict - therefore they are not suited to be part of the unique identifier.

The PID changes if an application or the OS is restarted - it is therefore not usable to differentiate between two programs. Especially if applications are relocated a user could be confused because the application would not appear with the same name again.

The last possibility is to apply the hash function on the arguments. Arguments are always different if a process is started multiple times with various tasks . If, however, the name and the arguments of two processes are identical the system shall treat those programs as one. For more information regarding that decision please see section 4.3. In order to keep the original name of the process the unique identifier is a combination of the process' name and the hash on its arguments, delimited by a # character:

```
'<process name>' + '#' + 'Hash(Arguments)'
```

4.2.3 Introduction of Tags

In order to address the process grouping problem described in section 3.2.2 a new property is introduced: the *tag*. The explanation in section 2.5 shows that environment variables are saved for every process and inherited by the calling process. This can be used elegantly to solve the already listed problems concerning the relationship between processes.

A tagged process holds an environment variable called *PAPILLON_TAG* with a certain value. The Client needs to be told via a configuration file which values of the environment

variable are to be tracked. Every tagged process that holds a tracked value for PAPILLON_TAG will then be tracked. This way it is ensured that logical application groups are formed out of processes that share the same tag. figure 4.1 depicts an example where three processes share the value *groupA* inside their environment variable PAPILLON_TAG. On the assumption that during the last minute process A1 used 100 CPU cycles, A2 20 and A3 10 the Client should calculate a sum of 130 CPU cycles and send that process group to the Server. If the tag is not monitored the programs are not recorded. However, if one of the programs is tracked via name their impact result will still be sent to the Server.

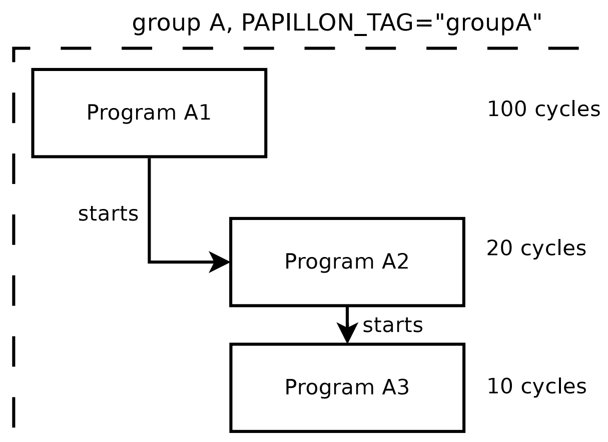

Figure 4.1: The logical group of three processes formed by their shared environment variable PAPILLON_TAG. The Client should report all of these processes as one entry with a total sum of 130.

4.3 Process Classification

This section describes the design how to classify different processes.

4.3.1 Record Reason

Including the two new ones from the previous section there are three possible reasons why a process is recorded:

1. It was one of the top three processes.
2. It was tracked via name.
3. It was tracked via tag.

The system should save the reason for the record because then they can be treated differently. The greatest difference between them is that tracking via name or tag ensures that if a process was executed without interruption the database will reflect that with a continuous set of data points. If the process was tracked because it was one of the top three processes this continuous set is most probably not available.

4.3.2 Process Ranking

Given the fact that it is very difficult to map network traffic or disk usage to a certain process the CPU utilisation is still the best way of ordering the process impact because it is fairly easy to find out the used CPU cycles for a process. In addition to that the Papillon System should stay as open as possible - solutions where the client's OS has to be modified or only certain applications are to be used are not well suited for this task. Another important factor is the existing database schema that should only be altered as little as possible. It already offers a table that stores the CPU time. This should be taken into account as well. Furthermore the Server offers a report function that lists less and more utilised hosts - this is again decided based upon their CPU levels.

These considerations lead to the decision that CPU based metrics are good enough for a rough ranking of processes. The rest of this section explains two metrics for which the process' share is used: Energy in Watt Hours and CPU usage in percentage.

4.3.2.1 CPU Share

It is assumed that the share should be calculated over a certain time period. Let C be the CPU percentage and S the share of the host CPU percentage ranging from 0-1. Process events are all recorded entries in the time frame. The result should be in an averaged form. Following formula is used:

$$C_{process} = \frac{\sum_{\text{process events}}(C_{host} \cdot S_{\text{process event}})}{\text{Number of process events}} \tag{4.1}$$

It is assumed that there are no missing process events inside the time frame. This is based on the assumption that the Client software is running without any interruption and is reporting the process continuously.

4.3.2.2 Energy Share

The energy share formula is similar to one used in (4.1) with the only difference that calculating the average is not necessary because the energy value is measured in absolute values - according to that decision following statement is true: if a process is analysed over a longer time period its absolute energy value increases.

Let E be the energy value, S the share of the host CPU percentage ranging from 0-1, this value is used again due to the reasoning at the beginning of this section - CPU values should be used for calculating the energy impact of a process. process events are all recorded entries in the time frame. The result is an absolute value.
Following formula is used:

$$E_{process} = \sum_{\text{process events}} (E_{host} \cdot S_{\text{process event}}) \tag{4.2}$$

4.4 New Process Computation on Client

After explaining the tracking methods and classification of processes this section describes how they should affect the client's work flow. The old work flow was simple because only the top three processes were announced by sorting all processes after CPU and cutting the list after the fourth entry. The new work flow, inspired by the new tracking mechanisms, is depicted in figure 4.2 and explained in the following list.

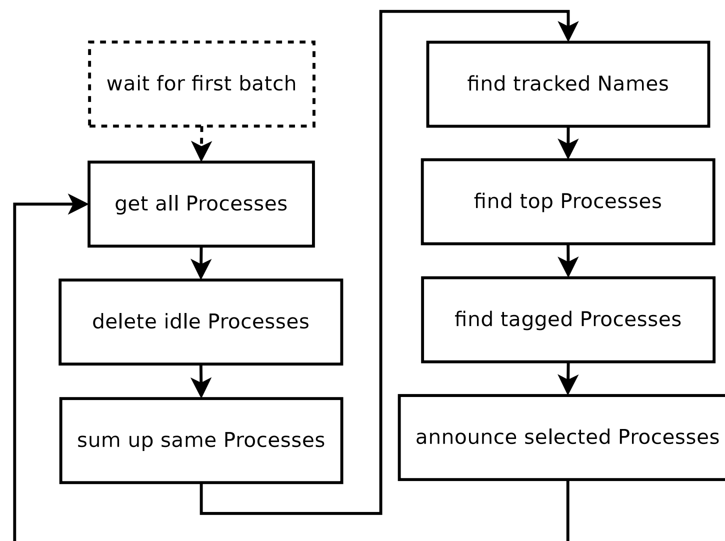

Figure 4.2: The new work flow of the process reporting mechanism.

- **wait for first batch:** The client reads the first batch of processes and sleeps for 60 seconds. This happens only once when the software is started and helps computing the actual CPU run times.
- **get all processes:** After that the current processes are viewed. Depending on the previous batch the consumed CPU time is computed.
- **delete idle processes:** Processes that did not consume any CPU time during the last 60 seconds are not considered.
- **sum up same processes:** If two processes have the same name and the same argument they are merged and treated as one process.
- **find tracked names:** Find processes with desired names. This involves all names that were defined in the configuration file.
- **find top processes:** If processes are not already tracked and part of the top three consumers they shall be added to the list. This is because there should not be duplicates inside the database.
- **find tagged processes:** If processes carry a tag that is selected they shall be added as well. Processes that share the same variable form its own logical group. This group holds the sum of CPU times of all included processes.
- **announce selected processes:** The accumulated list is announced to the server. This is done by sending a POST request to the Papillon Server.

4.5 Introduction of Consolidator

The specifications in section 4.1.2 say that a new subsystem called *Consolidator* should be created. Parts of its responsibilities are to help the user relocate processes and show the consequences concerning its assumed power consumption. This section explains the structure of the Consolidator, how hosts are selected and process information is gathered.

4.5.1 Architecture

The main architecture is depicted in figure 4.3 and explained in this section.

Client and server are already written in Java, therefore it can be assumed that a Java compatible environment is available. This, again, is very likely due to the multi-OS friendly nature of Java. [Ora15a]

A standalone application should be created that communicates with the server by API calls. The Papillon server information and other additional parameters can be saved in a configuration file. Reports that normally are extracted via the Papillon dashboard can directly be analysed via the application. The Consolidator has to send its requests via HTTP - the library *Okhttp* [Squ15] is used for this purpose.

In order to give graphical feedback the *Swing* library, a standard GUI toolkit provided by Oracle, shall be used. Swing offers all needed features like drawing windows and giving users the possibility to use their mouse in order to select and work with the application. Swing is typically shipped with the executable and a good and well supported option if a graphical interface is needed inside a Java application.

The server needs requests to either be in XML or JSON format. For this the *XStream* library is used [Pro15b]. This software collection offers an easy to use API that helps serialising objects to XML and backwards.

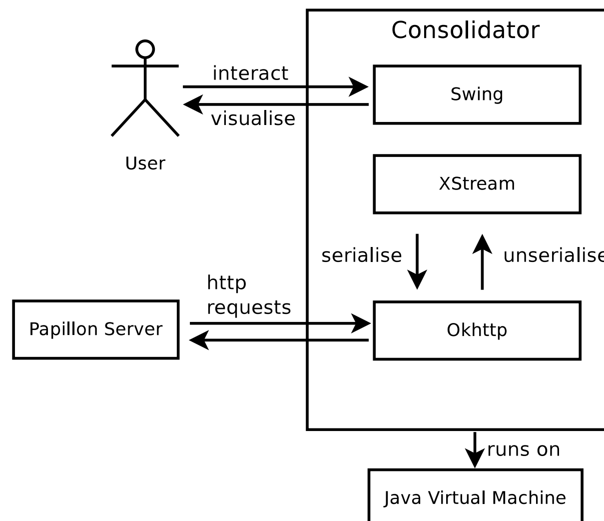

Figure 4.3: The basic architecture of the Consolidator including the used libraries with their tasks.

4.5.2 Work Flow of Consolidator

This section defines how the Consolidator should be used. The work flow of the Consolidator is depicted in figure 4.4, see the explanation below.

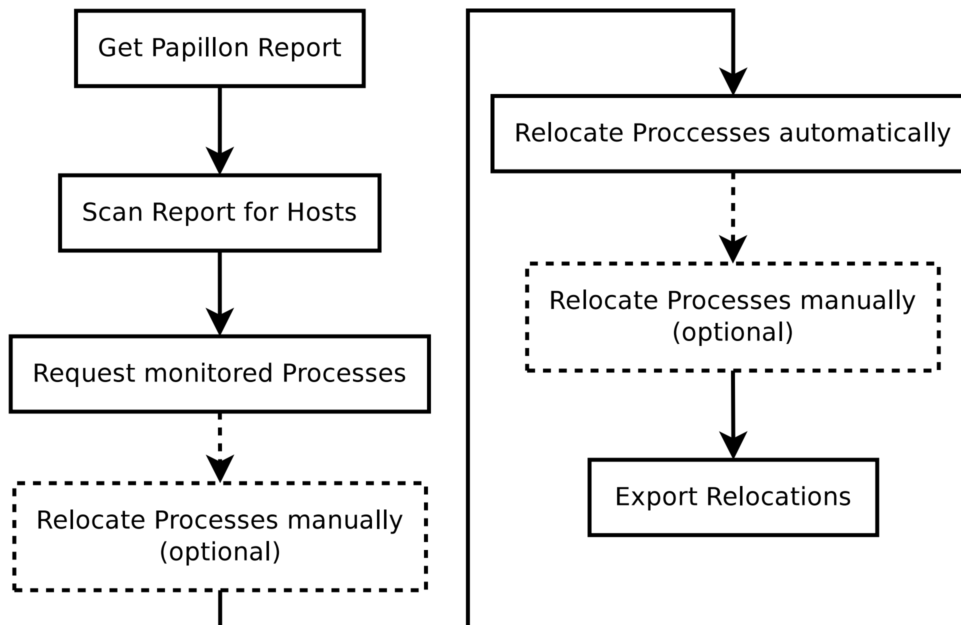

Figure 4.4: The work flow of the newly created component Consolidator.

- **Get Papillon report:** This is a report that can be queried from the Papillon Server. It holds general information regarding the hosts and helps selecting them. The report must be created for the correct time frame.
- **Scan report for more and less utilised hosts:** The report alone is not sufficient to decide what machines should be selected. Therefore the Consolidator has to implement its own checks.
- **Request monitored processes:** After the hosts are selected the Server needs to be queried for the monitored processes. This will be done by a newly created API call.
- **Optionally: relocate processes manually:** Relocations in this step are meant as a pre-optimising step before the automatic relocation takes place.
- **Relocate processes automatically**: This is based on the previous relocations and has the goal to consolidate as many machines as possible.
- **Optionally: relocate processes manually:** Similar to the pre-optimising step this can change the automatic relocations if necessary.
- **Export relocations:** In order to help data centre operators applying the new mapping of processes to servers the Consolidator exports its relocations to a file that is saved to the disk.

4.5.3 Selection of Hosts

The first question to answer is how hosts along with their monitored processes are selected. Papillon already offers a report functionality that is used.

The report gives information about:

- `top Hosts`: more utilized hosts, the threshold is defined by the server.
- `bottom Hosts`: less utilized hosts, the threshold is defined by the server.
- `host information`: HostId, HostName, Host-IP-Address, average CPU Usage, energy consumption, energy costs

This obviously is sufficient in order to decide which hosts should be selected: A report must be requested and the reported hosts then analysed. However there are three problems with the report.

Missing floor and rack information: Hosts are always ordered after their data centre, the floor of their position and the rack where they are placed. The Consolidator is meant to be used on only one data centre at a time. Therefore the data centre ID is saved inside its configuration file. However, the report only returns the ID of a host - this is not enough information and therefore the floor and rack ID has to be added to the report.

Fixed number of hosts: The server holds a configuration file that saves the desired amount of hosts. The API call will always report that number of less and more utilised hosts - even if there are not that many servers with low utilisation. Therefore an additional check must take place on the Consolidator's side.

Threshold on server side: The server holds a configuration file that not only sets the number of reported hosts but also the thresholds for utilisation - thus setting the limits for "less" (bottom hosts) and "more" utilised hosts (top hosts).

Bottom hosts have a maximum CPU usage, top hosts have a minimum usage. Although there is an optimal working point for some servers the user could have reason to target a different average load. Thus it is preferable to allow the user to redefine the limits on his own. Although this is possible via the already mentioned server configuration file it is a cumbersome way because of several reasons:

- The user needs server access. This is problematic because the server is the major component of the Papillon system and its authentication data should not be passed to many users.
- The server needs a restart in order to register the new thresholds - during this time the server does not respond to any Client requests.
- The user will not perceive the altered behaviour instantly - he still has to create a new report.

Considering this aspects the Consolidator shall implement its own threshold configuration. Here, again, the Consolidator's configuration file is a good place to store that information.

4.5.4 Request Host Information

Although the report supports different financial energy costs and PUE it is decided that these values could irritate the user and are therefore omitted. The only metrics that are dynamic and actually gained from the report are average CPU load of a host and energy in Watt hours. To sum up - the used values for a host, all gained from the report, are:

- Host ID, rack ID and floor ID (the latter two have to be implemented)
- Host IP address
- Host name
- Average CPU
- Host IP address

After the host is selected its process information can be considered. This step is described in the next section.

4.5.5 Compute Process Information

In order to gather the recorded processes for each host the server has to be queried. Due to the fact that such a process query was not of importance there was no adequate API call that preserved all important information. The interesting parameters for a process are:

- Process Name
- Relative CPU share according to (4.1)
- Energy share according to (4.2)

According to that and the need for a record reason described in section 4.3.1 a new API call named `processes` shall return the total CPU utilisation, in the formula denoted with C_{host} the CPU share information for that process ($S_{\text{process event}}$) and the energy value for each recorded process event (E_{host}). More information regarding the design of the API call can be found in section 4.10.

4.5.6 Graphical Feedback via Charts

In order to compare two processes visually a graphical feedback shall be implemented. Graphs should show all important information listed in section 4.5.5. Features like Export functionality and ease of use were reasons to look for a library - after some research the JFreeChart - library [Lim15] was chosen as supporting library. The viewed information dependent on time of a process should be:

- **Power:** The power value of the whole system at the moment of measurement.
- **CPU utilisation:** The utilisation of the whole system at the moment of measurement. This helps finding idle servers visually.
- **CPU contingent:** The share of the process of the whole CPU utilisation. This helps estimating the costs of the process visually.

4.6 Manual Process Relocation

This section explains how the manual process functionality of the Consolidator should be implemented.

4.6.1 Goal of Manual Relocation

The goal of process relocation can either mean to fill up an already loaded host or to clear less or more utilised hosts. This is done by moving processes between hosts and compare the new estimated impact with the target utilisation.

4.6.2 Functionality

A user should be able to select a process of a host and move it to other hosts. The information regarding the impact of the process both on the source and the destination host should be indicated. Due to the fact that the Papillon system does not support client modification in any way the Consolidator can only be applied to data records of the past. The Swing library which is able to support selections inside list elements either via mouse events or keyboard strokes should be used.

4.6.3 Example of Process Relocation

In figure 4.5 an artificial example of two hosts is shown. Host 1 with average CPU load of 30% and used 30 Wh recorded three processes over the considered time period. Host 2, on the other hand, has an average load of 50%, an energy demand of 30 Wh and recorded two processes. All processes have an equal share of CPU cycles - this yields to following absolute shares in percentage and energy:

- process Alpha: 10%, 10Wh
- process Beta: 10%, 10Wh
- process Gamma: 10%, 10Wh
- process Delta: 25%, , 15Wh
- process Epsilon: 25%, 15Wh

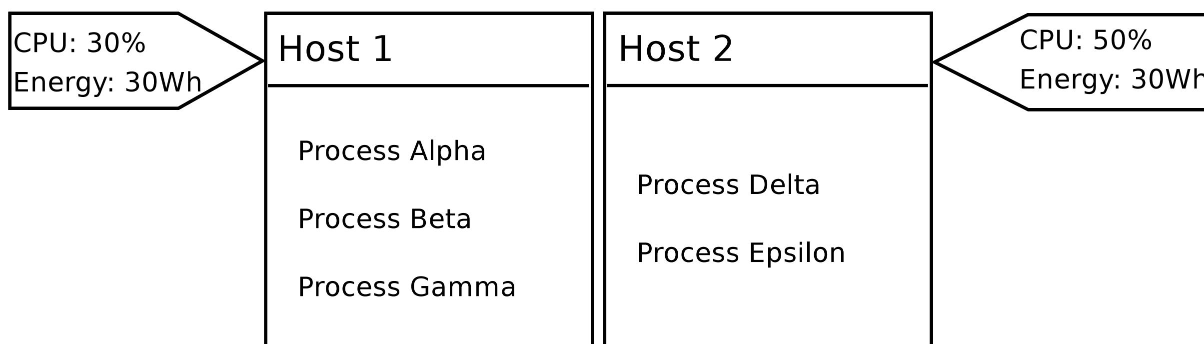

Figure 4.5: A pair of Hosts with multiple processes. All processes have an equal share of CPU cycles.

According to section 2.6 both machines are similar and easily comparable concerning their CPU levels. Therefore if process Gamma is moved from Host 1 to Host 2 the new allocation depicted in figure 4.6 is active. The Consolidator should highlight moved processes. A good solution is

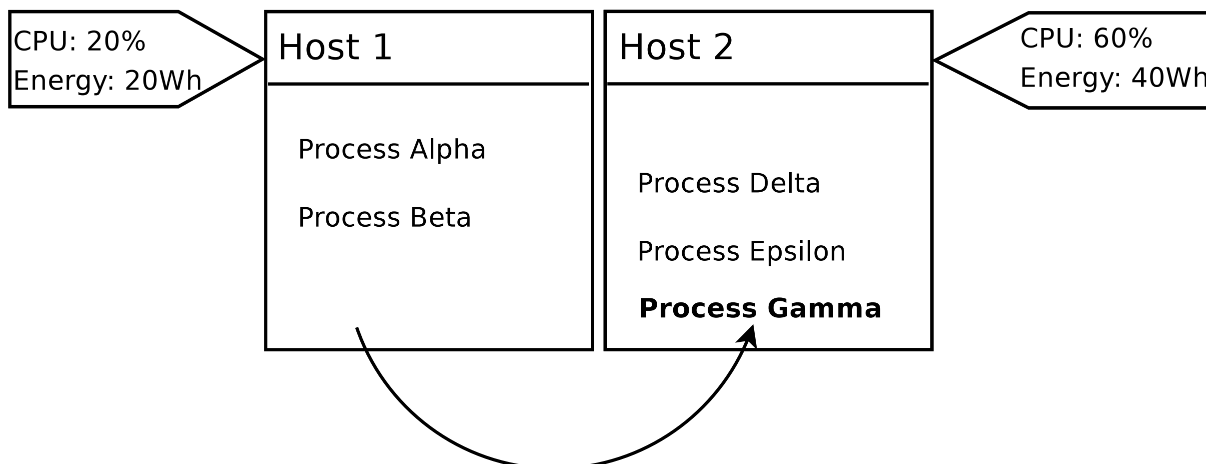

Figure 4.6: Process Gamma was moved from Host 1 to Host 2.

to colourise the processes according to their original host. If all host's processes were moved the host should be marked as "Empty".

4.7 Automatic Process Relocation

This section describes how the processes are relocated automatically.

4.7.1 Goal of Automatic Optimisation

In comparison to the open goal of manual Relocation 4.6.1 a very distinct goal must be formulated in order to create an algorithm. The most obvious goal is to move as many processes as possible from all hosts with low utilisation to hosts with higher utilisation.

This results in a subset of the BPP which is defined as "objects of different volumes must be packed into a finite number of bins or containers each of volume V in a way that minimizes the number of bins used". [Wik15a]. This is similar to the approach chosen in [SKZ08].

4.7.2 Used Algorithm

Due to simplicity the first fit algorithm is chosen ([Wik15a]) that does not yield to optimal solutions but should be sufficient for most use cases. The algorithm is shown in algorithm 1.

The algorithm has following properties:

- A process will only be moved to a TopHost if this action does not exceed the limit for a host's utilisation.
- A process will always be moved to some TopHost if possible. It will iterate through all topHosts.
- The algorithm produces different solutions depending on the sorting of the used data structures BottomHosts, TopHosts, BottomProcesses.

Algorithm 1 Auto Relocation Algorithm

```
mcu = maximum CPU Utilisation
for all BottomHosts do
    for each bP in BottomProcesses do
        repeat
            choose next TopHost = t
            if (t.utilisation + t.moved-utilisation + bP.utilisation) < mcu then
                report bP to be moved to t
                t.moved-utilisation += bp.Utilisation
            end if
        until bp reported for moving ∨ all TopHosts iterated
    end for
end for
apply movings
```

- The time complexity of the algorithm if n equals BottomHosts plus average BottomProcesses plus TopHosts is $O(n^3)$ because in every BottomHost every BottomProcess causes a request if it can be relocated to another TopHost.
- The space coomplexitiy of the algorithm is $O(1)$ because only possible movings have to be saved.

A different approach that is outlined by [MOS12] would be to use the paradigm of constraint programming. This way a VM or a process could be assigned to a machine by giving certain constraints per movable object and server. This solution is not followed but marked as a possible future extension in chapter 7.

After explaining the algorithm and listing its core properties the next section defines the printing functionality that is necessary for further passing the Consolidator's result.

4.8 Results of Process Relocation

Due to the fact that the Consolidator produces a new mapping of processes to hosts, and it is meant to be used by a data centre operator, an export function of this mapping must be defined. It should be easily readable and intuitive.

The proposed solution is to use a text file because the same output that is written to the console can be forwarded to a text file. The format of the file should be:

```
<name of process> <source host ID> -> <target host ID>
```

The following example

```
firefox 1 -> 2
```

denotes to a process called firefox being relocated from host 1 to 2.

4.9 Design of Consolidator

The previous sections explained the usage of and the arguments for the development of the Consolidator. This section outlines what rules the implementation should follow.

4.9.1 Most important Classes

The Consolidator's most important classes including a small explanation can be found in the following list:

- `PapillonConsolidator.java`: Holds Main - Function and most important functions. Is responsible for saving the hosts, analysing the reports, calculate process impact.
- `MainFrame.java`: Holds MainPane and most GUI functionalities. The initialisation method describes the panes and layout. Listeners implement the responses based on user input.
- `AnalyzeSystemThread.java`: Is a Thread-Class that is responsible for analysing the Hosts. This functionality is implemented as thread so that the GUI stays responsive.
- `Marshaller.java`: Marshalls and unmarshalls data objects that were either received or have to be sent via the network. Due to the fact that the Consolidator does not change any data all messages are requests - for example: report creation, report querying and process information querying.
- `HostReport.java`: Represents a host, holds general information of the host and is used in order to
- `ProcessGroup.java`: For every process that was recorded on a host one process group is created. They differ in parameters like name and original host.
- `ProcessEvent.java`: A process group has various static properties and in addition to that a recording per time unit (typically every minute). Those recordings are saved as process Events.
- `ConfigurationConsolidator.java`: Manages the configuration file. This file holds static information and has to be read at start and during execution.

This list is not exhaustive, more information can be found in chapter 5.

4.9.2 Class Diagram

In figure 4.7 a class diagram is shown that consists of the previously mentioned most important classes.

`ConfigurationConsolidator.java` as well as `AnalyzeSystemThread.java` are not shown in the diagram because `AnalyzeSystemThread.java` is a standalone Thread started by `MainFrame` `ConfigurationConsolidator.java` is addressed via Singleton Pattern meaning that every class has access to it.

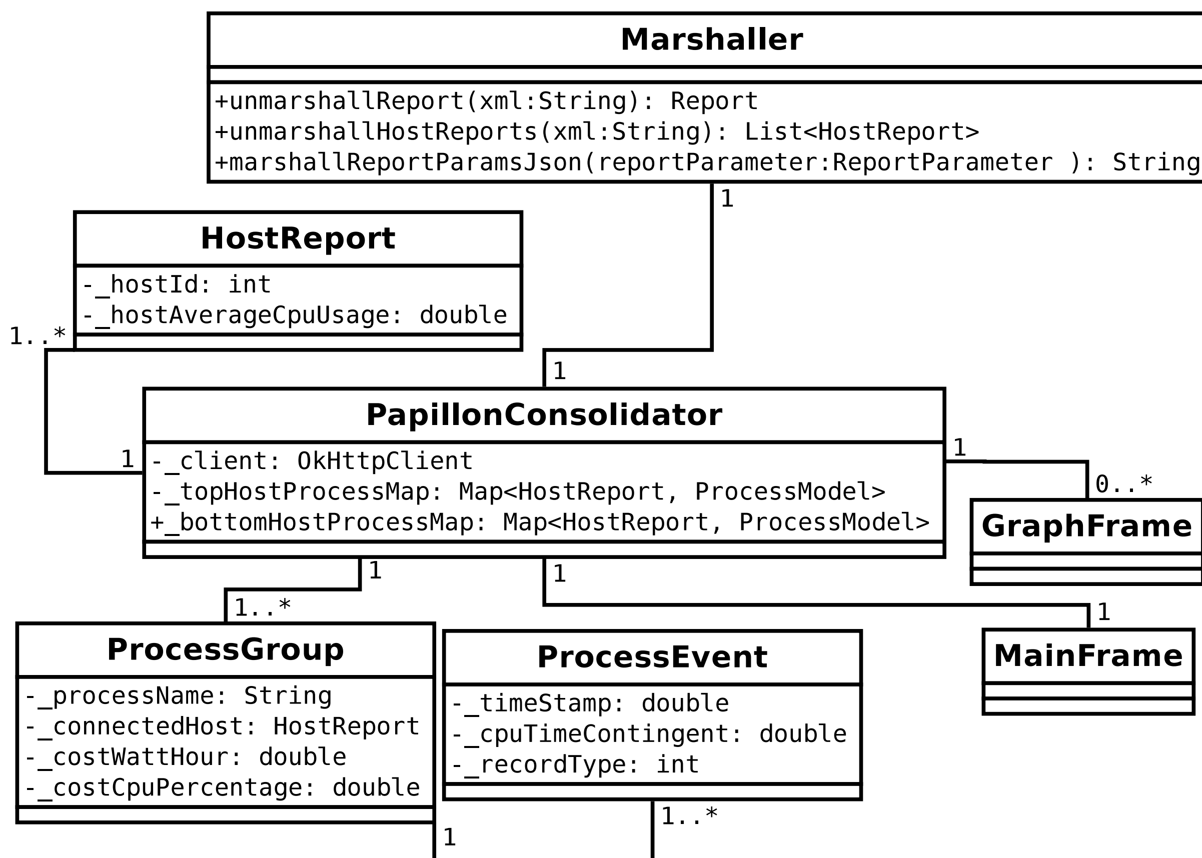

Figure 4.7: A basic class diagram of the Consolidator Software

4.10 Design of Server

Based on chapter 3.1.4 this section describes important aspects of the server's design in order to clarify the necessary modifications.

Jersey Framework: In order to offer a RESTful Web service the server uses the *Jersey* Framework [Cor15] which holds its own API that simplifies the development of such services. Jersey takes over the consuming and producing of exchange formats like XML and JSON. It maps Uniform Resource Locator (URL)s to functions and defines the appropriate HTTP request.

MyBatis Framework: In order to connect to the underlying MySql Database the *MyBatis* [myb15] framework is used. Instead of mapping data objects to rows of database tables this framework maps functions inside the source code to SQL queries.

MyBatis uses a number of factory patterns that have to be used in order to create a consistent system. This can be seen in the next section which shows an example API call.

Sequence Diagram of an API-Call: In order to limit the descriptions only to subcomponents that are critical to the newly created API call explained in section 4.5.5 this part shows the call hierarchy of the host's API call that is used in order to gain information from it. Basic explanation on how to use these calls can be found in the Stratergia document [Str14]. In order to gain information about a registered host following URL needs to be addressed:

`/datacenters/{datacenterId}/floors/{floorId}/racks/{rackId}/hosts/{hostId}`

The called subroutines are illustrated in figure 4.8. The actor calls the server via the given address. The request is converted via Jersey and the call is passed on to different classes until the mapping between a Java function and a MySql query takes place. Then the resulting data is passed back to the actor and again converted by Jersey to the chosen data format.

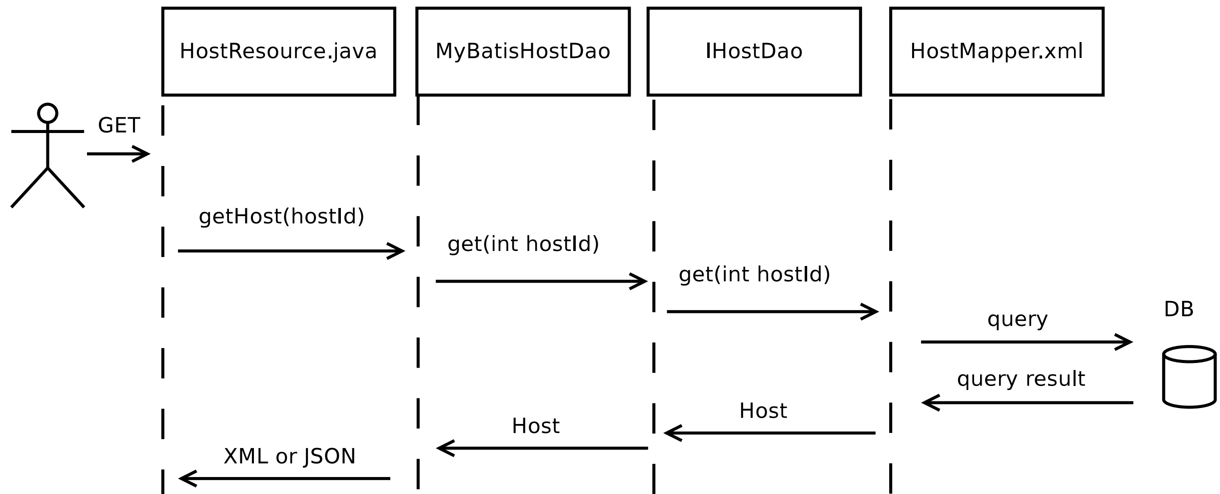

Figure 4.8: An API call example: get information about a host.

The description of the basic design of the server concludes the design chapter. The next chapter focuses on the implementation task.

5

Implementation

This chapter documents the implementation process - it gives detailed instructions about the used tools and coding details. The chapter starts by listing the used environments for development and testing and moves over to the modifications of the existing software packages Papillon Server and Client and ends in describing the implementation of the new software package Consolidator.

5.1 Environment

This section describes the used development and runtime environments for Client, Server and Consolidator.

5.1.1 Development Environment

All programs were compiled on an Arch Linux OS with 64bit and a Kernel with a version >= 3.10. The Papillon environment setup document [Str11] notes under section 1.1.5 that Eclipse for Java EE developers should be the used for development. Accordingly Eclipse for Java EE developers Luna Service Release 1 (4.4.1) was used. In order to have a better overview of Swing's design elements Eclipse was extended by the *Windowbuilder* plugin, developed by Google [Goo15]. The plugin was used to create the layout and the main elements of the GUI. Given the fact that all software components are based on Java 1.7 no other development environment was used.

5.1.2 Runtime Environment

There were two basic runtime environments. Primarily the development system was used as Client, Server and Consolidator. For this case a Python script was used that populated the database with artificial entries. It produced data which timestamps spanned throughout two days in the past. These entries were created for every minute, there were always four process entries per record. Every process had one quarter of CPU share. The average CPU load varied between ten and ninety percent. A list defined the amount of artificial hosts - it held the names of the hosts and based on their name the CPU level was chosen. More information regarding that script can be found in section 9.2.

In order to test the tools on different hardware following machines were activated as secondary environment:

- Server and Client 1:

  - Ubuntu 14.04.1 LTS 64 bit
  - Quad-Core AMD Opteron(tm) Processor 2356 (4 Cpus) - 4 threads possible
- Client 2:
  - Windows Server 2008 R2 64 bit
  - Dual AMD Opteron 285 (2 processor) 2 threads possible
- Client 3:
  - Windows 7 32 bit
  - Xeon (2 Cpus) - 4 threads possible
- Client 4:
  - Win 2008 R2 64 bit
  - Xeon e5410 (2 Cpus) - 8 threads possible

The usage of the latter machines is documented in chapter 6.

5.1.3 Workload Generator

A workload generator is a tool that creates artificial load in order to make sure that a process can be monitored. This is helpful during development because arbitrary software would not be reliable enough to produce a constant load. Two different generators were used on Linux and Windows OS.

Linux - stress-ng: A tool called *stress-ng* [Kin14] was used. This tool is derived from the standard *stress* program that is part of most Linux installations but more advanced. It gives more possibilities in capping the maximum load and has many different stressing techniques. *stress* had troubles utilising multiple threads at the same time and keeping the utilisation below a certain level - for single threaded mode the program *cpulimit* [Mar15] made sure that *stress* used less than 100% load.

Windows - Prime95: Although the software *Prime95* [Res15] was originally developed for finding huge prime numbers this tool can also be used as a stress program. Although it does not give many stressing options it proved to be sufficient for basic testing on windows machines. In order to keep the utilisation below a threshold the number of threads and the maximum amount of RAM was decreased until a sufficient result was produced.

5.2 Server Modification

Based on section 4.10 this section describes the necessary changes on the server code. These modifications can be split up into three parts:

- The separation of records depending on their record reason specified in section 4.3.1. Here not only the server and the database have to differ between them also the Client needs to send that very information.
- A new API call named *processes* that gives detailed information regarding the process' recordings of the past. This is due to the fact that there has not been a need for such an API call.

- The modifications for the report so that it contains rack and floor information. Without that information a host's processes cannot be queried.

5.2.1 Record Reason

In order to differentiate between different record reasons the database needed to be modified. This was done via the original script that was responsible for the configuration of the database. The table *app* was extended by another column named *recordType*. There are three possible values for this new column:

- `0`: Top process - so far this was the default and only top three applications were reported.
- `1`: Recorded because of name - if the configuration file of the Client held the name of the application this would be the case.
- `2`: Recorded because of tag - the process held a value for the EV PAPILLON_TAG that is listed inside the Client's configuration file.

Due to the fact that it is much easier to use meaningful variables instead of arbitrary integers a new Java-Enum called `RecordType` was created on both the Client and the server. Naturally during transmission only integers were able to be sent - this is solved by converting the integers to Enum at receiving and the other way round while sending data objects.

Besides changing the data base schema following changes had to be applied:

- **ActivityMapper.xml:** `ActivityMapper.xml` is responsible for mapping calls for the activities of a host to a SQL query. These queries had to be modified so that they took the new column into account.
- **App.java**: The class that stands for a process needed modifications so that it saved the new member.

5.2.2 API Call "processes"

The new API call should carry the properties of a process according to the list from section 4.5.5:

- **Process name**: The name of the recorded process.
- **Relative CPU share**: This is calculated according to (4.1) and stands for the share of CPU utilisation in % for the recorded process.
- **Energy share**: This is calculated according to (4.2) and stands for the share of energy in Wh for the recorded process.

Although it is possible to hide all process events and only give the mentioned values for that process all events are delivered. This way the Consolidator can check if a data point per minute is available (thus full coverage and no missing points) and charts can be drawn which was part of the original goals. Otherwise the necessary information would be missing.

The new call is called *processes* and concerning the URL address has to be sorted after deciding for a host. If a host with data centre = one, floor = two, rack = three and hostId = four, following URL gives access to the call:

```
/datacenters/1/floors/2/racks/3/hosts/4/processes
```

The call has to be requested by a GET call - no other requests are needed.

MyBatis classes: There are some classes that needed to be created to follow the MyBatis guidelines. Their creation follows the previous example from figure 4.8: A new function called `getHostProcess` is created inside `HostResource.java` that takes following arguments:

- `process name`: Optional, all processes are shown if no name is given.
- `startTime`, `endTime`: A time frame which limits the number of results. Recorded processes have to occur between those times. Both times are in epoch format [1]. If no time is given then all processes for the last 24 hours are queried.

A new data access object (DAO) class called `MyBatisProcessDao` is created that implements a new Interface called `IProcessDao` - methods that are defined here can be implemented by MyBatis. This class therefore holds two empty methods, similar to the previous ones:

- `getProcessEventsAsList`: Gets process' information if no name is given.
- `getProcessEventsAsListByName`: Gets process' information if a name was given.

Those two methods' signature occur in a another class called `IProcessMapper` which represents a place holder for the mappings that take place in a file called `ProcessMapper.xml`. The mapper XML file implements the methods listed in `IProcessMapper` through MyBatis Framework with database query commands. The content of the XML file consists out of two simple SQL queries. In order to receive a whole set of entries a result map called `resultProcessEvent` was created in order to receive a whole set of process events and not only one. The arrangement of new classes concerning the described MyBatis functionality is depicted in figure 5.1.

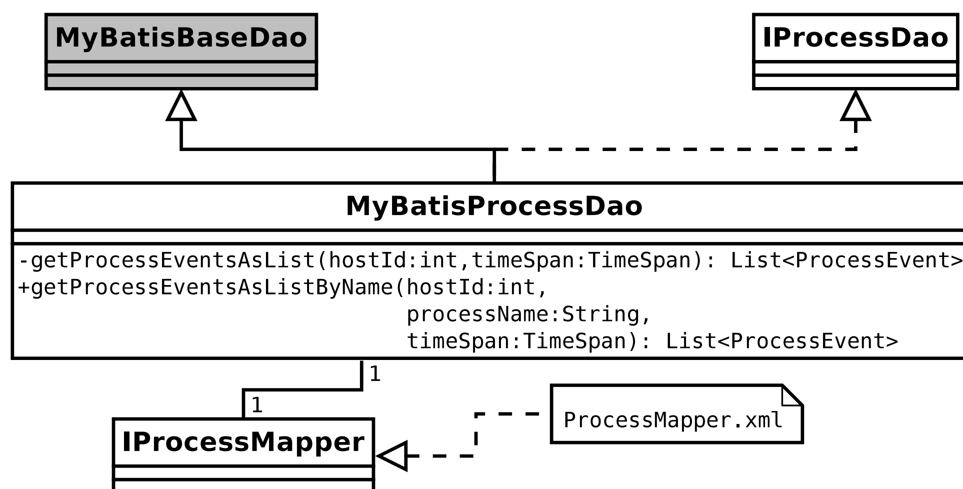

Figure 5.1: Modifications in code concerning MyBatis functionality. Classes with white background were newly created.

Process Event A new class called `ProcessEvent` was created. For every process and time frame there is a certain amount of process events. If a process was recorded throughout the monitoring of the machine then there should be a process event for every minute. A process event has following attributes:

- `name`: The name of the process.

[1] The Epoch Format counts the seconds that went by since the first of January in 1970 UTC [Wik15b]

- `timeStamp`: The time stamp of the recording.
- `cpuTimeContingent`: The CPU share of the process at the time of recording.
- `power`: The power value of the host in Wh at the time of recording.
- `recordType`: The record reason.

The same class is used for exchange and can therefore be found in the both client code and the Consolidator code.

Result of API call: The XML guidelines of the API call can be found in listing 5.1.

```
1  <processEvents>
2  <processEvent>
3    <name>Name of Process</name>
4    <timestamp>TimeStamp of Recording</timestamp>
5    <cpuTimeContingent>CPU Share</cpuTimeContingent>
6    <power>Power Value of Host</power>
7    <cpuStat>CPU Percentage</cpuStat>
8    <recordType>Why was this Process recorded</recordType>
9  </processEvent>
10 ...
11 </processEvents>
```

Listing 5.1: The XML structure of the newly created processes API call

This call delivers all necessary information in order to support the Consolidator.

5.2.3 Floor and Rack inside Report API call

In order to extend the report call by floor and rack information the class `HostData` was extended by two new members including their getters and setters. The file `ReportMapper.xml` that held the SQL query was changed so that the floor and rack was queried as well. The *XMLType* information was altered for `HostReport` in order to tell Jersey to serialise the two new members. The function `createHostReport` inside `AnalylisForStorageGenerator` was modified so that the created report was supplied with the rack and floor information from the SQL query.

5.3 Client Modification

This section describes all changes that were necessary in order to work together with the server smoothly. Kriechbaumer [Kri15] applied clean-ups and some fixes to the existing Papillon Client source code during the thesis. The modifications of this section reflect only changes that happened after his refittings. The changes can be split into three groups:

- Fix processes with same names.
- Track process via name or tag.
- Adapt the client according to previously mentioned server modifications.

Before the actual implementation some classes were newly created and extended. A new class `ApplicationMonitor` decides what processes are selected. It holds all necessary functions in order to follow the proposed work flow from figure 4.2.

The class `ApplicationEntry` is extended by some members. They save the necessary information of a process before it is sent to the server. Besides mainly informational properties like PID and name there are specific variables that help calculate the importance of the process or support tracking.

Every process, at the time of the measurement, has an absolute CPU time - the time the process used the CPU in absolute terms. This is not very helpful because this does not reflect the process' influence since the last measurement. Therefore a variable called `_diffCpuTime` is used that holds the CPU time that was used only during the last time slot.

The new member `_tags` saves the content of the *PAPILLON_TAG* variable as a list, if available. If this environment variable does not exist the list is empty.

5.3.1 Fix Processes with same names

According to section 3.2.1.2 the old system had troubles in computing processes with the same name. The solution, portrayed in section 4.2.2, is to use a hash function that is applied on the arguments of the process.

The `ApplicationEntry` class is responsible for holding information regarding the recorded process. It was extended by several new members. The ones that are necessary for solving the same name problem are:

- `args`: The arguments of the process.
- `uniqueName`: The unique Name of the process.

Unique names are formed by concatenating the name, the # sign and a hash on the arguments. There are many different hash algorithms. Java implements a very simple hash functionality for every class that is derived from Object - which is the case for the String class. According to Java's documentation [Ora15b] the hash function of a string s with size n is defined as:

$$h(s) = s_0 \cdot 31^{(n-1)} + s_1 \cdot 31^{(n-2)} + ... + s_{n-1} \quad (5.1)$$

Given the fact that 32 bit Integer additions are used the resulting value ranges from $-(2^{31})$ to $+(2^{31})$. This function is used because no additional libraries are needed. In order to improve the readability a minus is replaced by a zero. The Client is then adapted only to send the processes' unique names instead of their original names that could occur multiple times.

5.3.2 Track Process via Name

Tracking processes via their name is part of the new work flow deciding what programs are reported.

5.3.2.1 Implementation of the new Work Flow

The following list repeats the work flow from section 4.4 and lists the needed modifications:

- **wait for first batch**: `DataCollectorThread` saves all processes into a variable called `appInfoOld`. This is done via a function called `getRunningProcesses` that uses the SIGAR library. The old data is then always overwritten in the next time slot by the new data in an infinite loop until the Thread is shut down.
- **get all processes**: The function `getApplicationList` in the `ApplicationMonitor` class is executed and supplied both with the old and the current process list.

- **delete idle processes**: The function `calculateDiffsAndFilter` looks at the old and new process information. If a process was created recently (it is not part of the old data) then its difference time of the cpu equals its absolute CPU time. If the process was already started on the previous time slot then its difference time is the subtraction of the two absolute values. If the CPU time is zero the entry is dropped.
- **sum up same processes** the function `addIfUniqueName` is executed on every remaining process and sums up processes with the same unique name.
- **find tracked names**: The function `findTrackedNames` is used in order to search through all processes for the tracked names. Furthermore the method sets the record reason accordingly.
- **find top processes**: The list of all processes is sorted after CPU time and if a process was not already found through name search it will be added. The record reason will be set to top.
- **find tagged processes**: To the existing list tagged processes are added - this is explained in the next section.
- **announce selected processes**: The list is returned and sent by the `DataCollectorThread` to the server.

5.3.2.2 Configuration of tracked Names

Desired names are given via the configuration file `papillonclient.properties`. An example line would be

```
trackedApplications=process1, process2, process3
```

which yields to tracking three processes named *process1*, *process2* and *process3*.

5.3.3 Track Processes via Tag

The environment variable that is considered is called `PAPILLON_TAG`. The content of this variable can be an arbitrary string. It could also contain multiple entries that are comma or space separated.

The filtering of the tagged processes is done after top three and tracked processes via name were added. The function `findTrackedTags` is responsible for looking through all processes and selects only the ones with the desired tags. A new class `TagEntry` was created that is derived from `ApplicationEntry` - this is because a tagged entry differs in some way from a named or top three entry:

- **name or unique name**: A tagged entry could consist of multiple processes. Therefore the unique name of a tag entry is

  ```
  'TAGENTRY' + '#' + <tag name>
  ```

- **included processes**: The class `TagEntry` has a member `_includedProcesses` that holds all the processes that contain the desired Tag. Although this information is not transmitted to the server it is printed to the screen. This helps assuring the correcttfunctionality.

- **CPU time**: In comparison to a named or top three entry where the time is always a value on its own a tagged entry carries the sum of all tagged processes. In order to count every CPU cycle only once the following approach is chosen: multiple tags result in a division of the cpu time for this process when a `TagEntry` is created.

5.3.3.1 Configuration of tracked Tags

Desired values are given via the configuration file *papillonclient.properties*. An example line would be

```
trackedTag=TESTTAG1 TESTTAG2
```

which tells the client to track all processes that have a variable *PAPILLON_TAG* that holds the values *TESTTAG1* or *TESTTAG2*. If a process contains multiple configured tags it is recorded and its CPU time will be divided accordingly.

5.3.3.2 Proper Start of tagged Processes

In order to make sure that a tagged process actually carries the correct tag following procedure has to be followed:

Portable Operating System Interface (POSIX) compatible systems: On a POSIX compatible OS the program has to be started via console:

```
PAPILLON_TAG=TAG1 process1
```

process1 will hold the environment variable *PAPILLON_TAG* with value *TAG1*.

```
PAPILLON_TAG=TAG1,TAG2 process2
PAPILLON_TAG="TAG1 TAG2" process3
```

Like the upper example but both processes will hold the values TAG1 and TAG2.

Windows system A batch file has to be created that starts the program. Inside that script the *PAPILLON_TAG* is set, afterwards the program can be started by using the `start` command. The running process will contain the correct tag. In listing 5.2 an example batch file is shown that starts the previously mentioned Prime95 as tagged program.

```
1 @echo off
2 set PAPILLON_TAG=MYTAG
3 start prime95.exe
```

Listing 5.2: Proper start for Windows OS in order to supply the process prime95.exe with the correct EV PAPILLON_TAG with value *MYTAG*

5.4 Consolidator

This section describes the code of the Consolidator - it follows section 4.9. The Consolidator is a prototype which main function is the handling of the visualisation of the manual and automatic relocations of the recorded processes. In addition to that it gives host and process information. The following sections describe the implementation based on the Consolidator's features.

5.4.1 Configuration

Similar to server and client the configuration is done via a file that is managed by a Singleton class called `ConfigurationConsolidator`.

The properties saved in this file are:

- Master IP address and master port: used in order to connect to the server. Of course the user has to make sure that the Server is accessible from the machine that executes the Consolidator.
- Data centre ID and data centre name: Needed for selecting the correct data centre. This was put into a configuration file and not part of GUI because it tends to change seldom.
- Relocation CPU limit: This is the limit the auto relocation takes into account when moving processes. If moving a process manually results in violating this threshold a warning is printed, automatic relocation will always propose relocations that do not violate that threshold.
- Maximum utilisation of bottom hosts and minimum utilisation of top hosts: these thresholds are used in order to check the found hosts for validity. This is due to the mentioned fact that the threshold should be checked again at the Consolidator because changing the thresholds directly at the server has a few disadvantages, listed in section 4.5.3.

5.4.2 Graphical User Interface

The GUI helps the user viewing entries, receiving additional host and process information and controlling the relocation process.

Swing Layout: The Swing library supports different layouts. In order to keep a consistent but variable layout the *Gridbag* layout was chosen. This layout expects the elements to be sorted in columns and rows without fixating the height and width of every cell.

Overview: A screen shot of the GUI can be found in the appendix in the Consolidator's user manual in figure 9.1. Four JList Objects of the Swing library contain top hosts, bottom hosts and the corresponding processes (top processes and bottom processes). Below those areas are two buttons that relocate processes manually and show their graphs. One logical group of text fields shows process and host information. Another group is responsible for setting the desired time frame and asking the server for report and analysing it.

Code Structure: The constructor of the `MainFrame` Class is responsible for creating and initialising the window and its graphical elements. The interactions were done using listeners. Every item that is used for interaction between the user and the system holds a listener that starts the correct function if the action was performed. On buttons this would mean that the user activated it, on lists this yields to a new selection of an entry.

5.4.3 Create Report

A report needs to be created on the server side in order to find less and more utilized hosts. The report has to be created with certain parameters. These are:

- start and end of the time frame in which the report should be calculated
- data centre ID and name

Data centre information is saved inside the configuration file due to the fact that these properties change very seldom. However the time frame is likely to change quite often. Therefore the times are given via two text fields that take dates via a human readable format and convert them to epoch form via a helper class called `TimeConverter`. It is important to use a Coordinated Universal Time (UTC) format before conversion in order to omit problems that result out of different time zone settings on the computer.

The report parameters are then marshalled to JSON which is the only occasion the Consolidator does not use XML. Although the server should support both formats only this one worked flawlessly. After marshalling the parameters are sent via POST request to the correct URL. If the response is empty the creation on the server side was successful. Otherwise the response holds an error message.

Assuming that the report was created the Consolidator will notify the user and wait for additional actions. Most likely the user will now try to analyse the report.

5.4.4 Analyse Report

In case a report can be found on the server it can be analysed in order to find the announced hosts and ask for their processes. The times that were used from report creation are reused because for each host the server needs to be asked what processes were recorded. For this reason the API call from section 5.2.2 is used. Following work flow is started:

Start of new thread: After checking the connectivity of the server a new thread called `AnalyzeSystemThread` is spawned and deals with the report and all linked tasks. During that the GUI is reset to a passive state and functional. The new thread, however, asks the server for all reports and uses the most current one. That report is requested from the server and analysed if not empty. During that process a progress bar is shown that tells the user how many hosts were queried so far and how many are still missing.

Choose hosts: The thread then takes all bottom and top hosts from the report and checks if they are within the given limits from the configuration file. If a host is within its specified limit it is either added to `_bottomHostModel` or `_topHostModel`. These models are from type `HostModel` and are needed in order to fill `JList` elements which are part of the Swing library.

Colourise hosts: In order to visualise the difference between hosts the background colour of these entries have a distinct colour. The colour is saved as a Red Green Blue (RGB) variable. This variable is calculated after the amount of total hosts is clear. The algorithm for calculating the colour is shown below in algorithm 2. The start and step values are based on experiments that resulted in best visual appearances. Based on the host's colours the original processes that are linked to it have the same colour and keep them, even after moving to an other host. This way the user can find out the original host of a process quite easily.

Algorithm 2 Colour Calculation Algorithm

```
i = 0
redStartValue = 100
greenStartValue = 100
stepRed = 220 / hostCount
stepGreen = 80 / hostCount
stepBlue = 220 / hostCount
for all host in Hosts do
    host.red = redStartValue + i * stepRed
    host.green = greenStartValue + i * stepGreen,
    host.blue = i * stepBlue
    i++
end for
```

Query processes: After deciding which hosts are selected the corresponding processes can be queried. This is done via the newly created API call from section 5.2.2. A class `ProcessModel` is created per host and saved in a map that takes a `HostReport` as key and a `ProcessModel` as value. This way a connection between those pairs can be saved. The `ProcessModel` is created by asking each host via a GET call for the recorded processes within the given time frame. All processes that were recorded due to the fact that they were one of the top three processes are deleted. This happens so that there are no gaps in between their process events. Otherwise any metrics that are deduced from the data points would not be reliable. Therefore a process should have been monitored throughout the given time frame. All remaining processes, those that were added because they were either tagged or tracked because of their name, are added to the list of processes. There are two classes that are responsible for storing process information: `ProcessGroup` that is created only once per process and `ProcessEvent` that holds all data points for the process.

Calculate costs: After every Host object was assigned with its process objects the costs of every process can be evaluated. The two metrics *CPU share* and *energy share* are calculated according to section 4.3.2. The function `calculateCost` inside every `ProcessGroup` is responsible for this functionality. The average CPU load of every host is part of the report information and can be seen as given - all needed parameters are available and the metrics can be calculated.

Reset: If the button is clicked after the analysis finished the hosts are queried again. Before this happens all lists are emptied.

5.4.5 Move Processes

Process events are always moved inside their `ProcessGroup` class - this way the events that belong to a process stay together. This class holds a member named `originalHost` that is shown to the user in order to keep the link to the original host. In addition to that a process entry in a list element always has the same colour as its original host entry.

If a process entry, a destination host and a source host are selected the move functionality is available. The relocation starts by checking if it would result in violating the given limit of average CPU load. As stated in section 2.6 two hosts are treated equally regarding their CPU loads. Naturally not only the CPU loads but the other two metrics are modified - energy share and CPU load. Additions and subtractions are calculated by saving positive or negative

modifications of these values into two variables. At creation time these variables are zero but via moving processes around they are altered.

If a host was modified its name signals that with a text that contains the new CPU modification in percentage. This is implemented by changing the result of the `toString` function of every `HostReport` according to their modification variables. A host with zero CPU load (all of its processes were moved) is marked with the word `EMPTY`.

5.4.6 Visual Feedback regarding the Process

There are not only the two previously mentioned metrics regarding every process in order to view its impact. There is also a visual feedback implemented with JFreeChart. The resulting charts give more insight into single data points by showing their power, host CPU loads and process CPU share at the time of measurement.

If a process entry is selected and the *graph* button is clicked the function `createNewGraph` inside `PapillonConsolidator` is executed. The function starts by searching through the *_graph-Frame* map - this map holds every frame including the associated chart if it has been created already. If that is the case the Frame is made visible. If, however, the chart was not created an instance of `GraphFrame` is newly constructed. Afterwards the datasets for power, CPU load and CPU share are fetched from the selected process. The x axis consists out of time data points. In order to give a better scalability multiple y axis' for power value of whole system in Wh, CPU load in % (ranging from 0 to number of cores times 100) and the fraction of CPU load of this process in % (ranging from 0 to 100) are used. This is implemented by using certain JFreeChart functions. First a `XYDataset` for all three curves is created - this is a dataset that contains the time on the x axis and the appropriate value on the y axis. In order to view the cpu contingent values as percentage they are multiplied by factor 100. Afterwards a `timeSeriesChart` is created that is made to contain time on the x axis. On this chart all datasets are defined. Afterwards new axis' are inserted by first creating a new object of type `NumberAxis` and setting it via the function `setRangeAxis`. In order to create different shapes for every curves new renderer have to be created and mapped to the correct datasets. Finally a `ChartPanel` has to be created that contains the `timeSeriesChart`.

5.4.7 Auto Relocation

The auto relocation algorithm is implemented as stated in section 4.7.2, the responsible function is `autoRelocation`. Bottom processes are marked for relocation by using a map named `toMove` that has a `HostReport` as key (the destination host) and a list of objects with type `ProcessGroup` as entry (the processes that should be moved). This way the algorithm holds all processes that should be moved and can keep track of the resulting changes in CPU utilisation of destination and source hosts. After all bottom hosts are iterated the algorithm loops through `toMove` and actually uses the relocation function on the entries. For results of this algorithm see chapter 6.

5.4.8 Print Function

The print Function `printRelocation` is implemented in agreement with section 4.8. It prints every relocation between hosts and saves the output to a file called `relocation_summary.txt` where the Consolidator is executed.

After describing the implementation of the Consolidator and the modifications of existing code in client and server, the next chapter shows results and verifications.

6

Experimental Results

This chapter describes findings and results that were gained throughout as well as after finishing the implementation part. Most of the results of this thesis are feature based. This is due to the fact that the majority of features were not available in the previous version of the Papillon system - see section 3.2 for further explanation. Therefore a comparison is limited. Because of that his section lists all shortcomings and explains how the problems were solved including results, if possible.

6.1 Verification

The implementation was tested on two environments - the first environment was an artificial database, the second environment consisted of five virtual computers. Both environments were explained in section 5.1.2.

The client experienced most changes and therefore was tested with several new Junit tests that included following test cases:

- **Correct difference times:** Check if the new work flow of the client does not change the correct times of processes. For this two maps of artificial processes were created - one with past values and one with current values. The function `getApplicationList` that is responsible for filtering the existing applications was supplied with these maps and the result was compared with the correct differential times.
- **Top three applications still found:** Check if top three applications are still found. Without any intervention the work flow should still report the top three processes depending on CPU cycles of the OS.
- **Duplicates:** This test checks if duplicate names are not counted twice but merged together. This tests, on one hand, if duplicates are found and if at most three applications are recorded as top three.
- **Total CPU time**: The routines in this test case checks if the total amount of CPU cycles are counted correctly. 50 process objects are created and passed to `ApplicationMonitor`, the actual CPU time is compared to the expected CPU time of all objects.
- **Tracked Processes**: This test case creates several processes and tracks two of them by name. First the configuration is modified so that it holds the desired names and then the usual work flow is started. If the two processes are not found the test fails.

- **Multiple processes, single tags**: This test case checks if processes are found if they carry a tag that can be found in the configuration. The test fails if one of the tagged processes is missing.
- **Single/multiple processes, multiple tags**: This test validates the division of multiple tags on processes. In order to prevent to count the CPU cycles twice of process with multiple tags its CPU time is divided by the amount of tags it carries. The test is done for single as well for multiple processes.

The existing tests on the server software were sufficient to test the small modifications of the server code.

6.2 Process Tracking

The original process monitoring problem that did not distinguish between processes with the same name was solved by a using a mechanism that differs between processes based on hashing their arguments. This solved some serious problems concerning their calculated CPU time for processes with the same name and gave the possibility to differ between them.

In order to solve the problem of unwanted gaps of process information in monitoring sessions two more tracking methods were implemented that helped monitoring processes without interruption. Firstly processes can now be tracked by their name, secondly they can be tagged with an environment variable. These changes were applied to the Papillon Client as well as to the Papillon Server.

6.3 Process Grouping

The absence of grouping processes and forming structures of some kind in the original software was solved by using environment variables - so called tags. Single or multiple processes could have a tag that can now be tracked explicitly. Due to their inheritance property it is safe to say that any spawned process also receives the same tag as its parent process. This was tested on both Windows as well as Linux OS:

Linux verification: The environment variables of a process can be viewed by looking at the file

```
/proc/<PID>/environ
```

This file holds the whole environment and is therefore appropriate for comparison between the new client's results and the actual situation.

Windows verification: On Windows machine a 3rd party tool called *Process Explorer* [Rus15] was used that gave the possibility to view a process' environment variables.

6.4 Process Visualisation

The missing visualisation of processes over a period of time with more information was solved by using the JFreeChart library in the newly created Papillon Consolidator. This way customers can view the history of their servers and receive a graph that helps them getting an overall picture. Multiple y axis' were used that show CPU utilisation of the system in %, CPU share

of the selected process ranging from 0 to 100% and power value of the system in Wh. The library offers an export function that is helpful if users want to share the produced charts with colleagues. Two examples of these exports from each working environment are shown below:

Real Test Environment: The processes were actually started and the data was monitored by the client. Figure 6.1 shows a CPU intensive program named *stress*. In contrast figure 6.2 depicts a program called *top* that is quite lightweight.

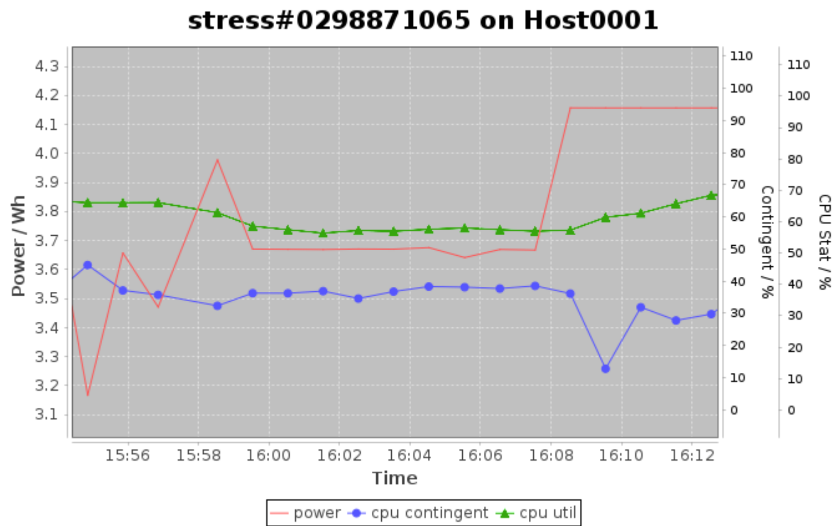

Figure 6.1: A chart from a process that was quite cost intensive. This can be seen in the CPU contingent value.

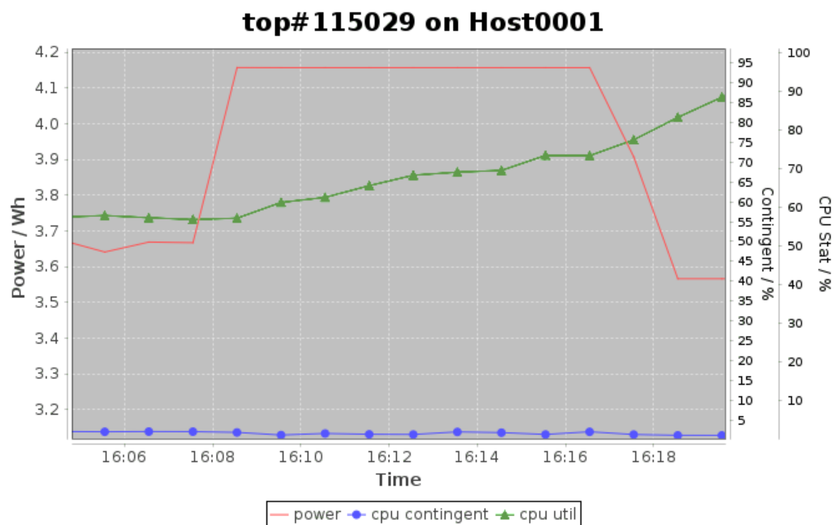

Figure 6.2: A chart from a process that produces less load than the one in figure 6.1.

Artificial Environment: The data base entries were created by a python script that supported early development.

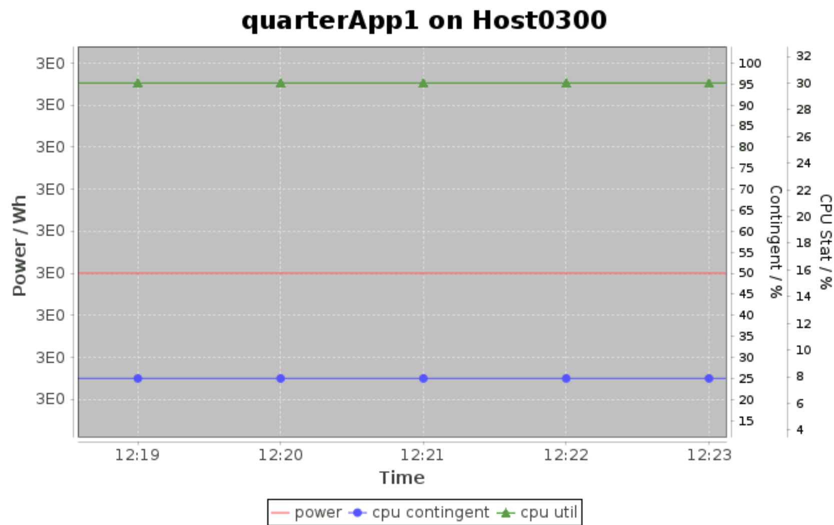

Figure 6.3: A chart from an artificial process entry. On host 300 the CPU load is always 30%.

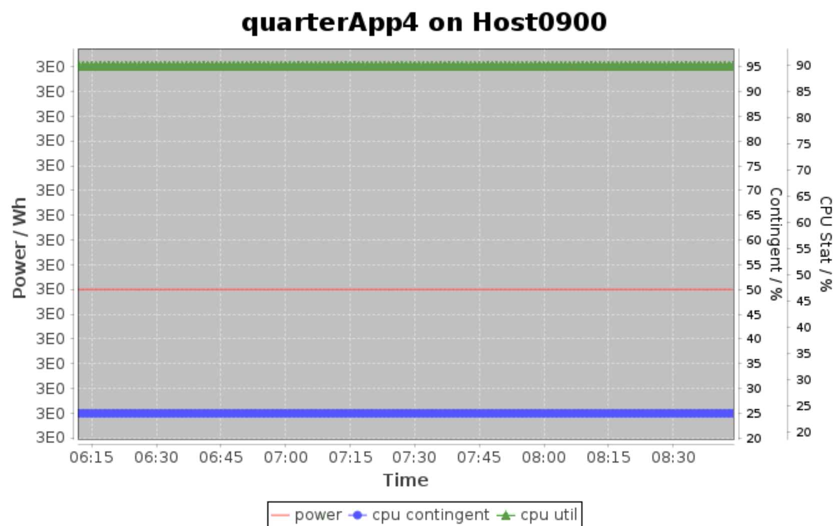

Figure 6.4: A chart from an artificial process entry. On host 900 the CPU load is always 90%.

6.5 Process Costs

Two metrics were introduced: CPU share and energy share. Both values of every tracked process can be viewed with the new Consolidator. Figure 6.5 shows a small example where host 100 with an average CPU of 10% demanded 8640 Wh over the queried time frame and the selected process is responsible for 2.5% and 2160 Wh of the whole host. This was one of the artificial data base entries where every process was responsible for one quarter of the load.

6.6 Consolidation of less utilised servers

Less and more utilised Servers can now be found through the use of the Consolidator. The thresholds are defined at the server and rechecked by the Consolidator. It uses the already

Host Name	Host100
Average Cpu	10.00
WattHours	8640.00
Process Name	quarterApp1
Original Host	0100
Relative CPU	2.50
Total Energy	2160.00

Figure 6.5: The two newly introduced process metrics: *CPU share* (Relative CPU) and *energy share* (Total Energy). The upper lines show host information whereas the lower lines show process information.

existing *report* and a newly introduced *processes* API call to find hosts and their recorded processes. The new API was necessary in order to keep all needed information in one call - existing calls tried to minimise the amount of sent data. Two relocation methods are available: manual and automatic.

Manual relocation: Can be used to observe the impact of processes on their original and other hosts. Processes can be relocated on both less and more utilised hosts. Figure 6.6 shows a screen shot in which multiple processes were moved to different hosts. This can be seen by the colours that are used. If a process cell has different colour than its parent host it was relocated. Furthermore if a CPU load was changed by adding or removing a process the parent host has a note including the difference in percentage. Host 900 for example has 31.5% more average CPU load, the load of host 2 was decreased by 2.5%. A host with no remaining processes is marked with the word *EMPTY* - visible on host 100.

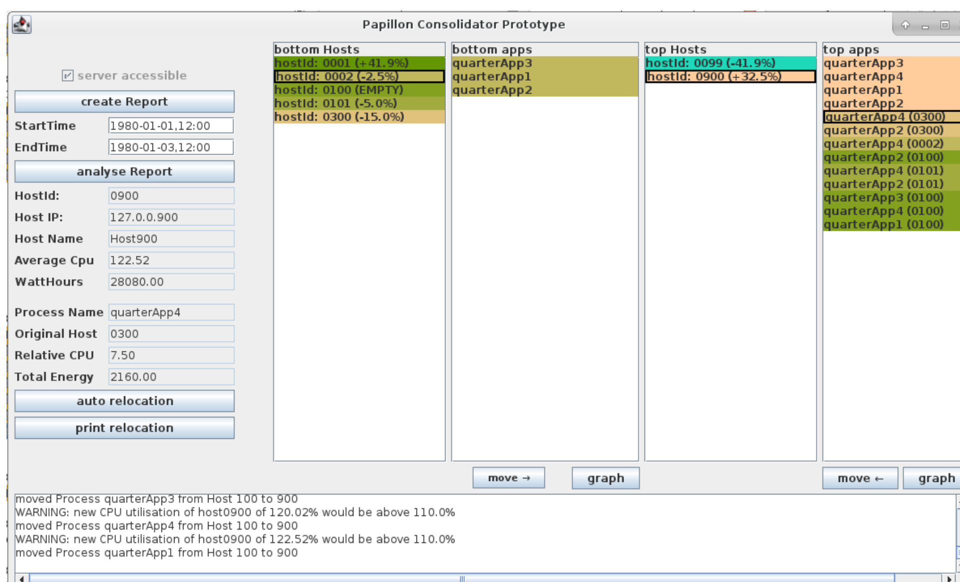

Figure 6.6: Manual relocation of different processes.

Automatic relocation: This feature uses the manual relocation in a way that as many as possible less utilised hosts are freed from processes. Figure 6.7 shows an example where four of five bottom hosts are marked as empty. As it was described in section 4.7 the algorithm does

not yield the same solutions every time it is executed. The maximum CPU load is taken into account by the algorithm. In the example the limit was 110 - host 99 has a load of 107,45%, host 900 108,83%. Both values are below the given limit.

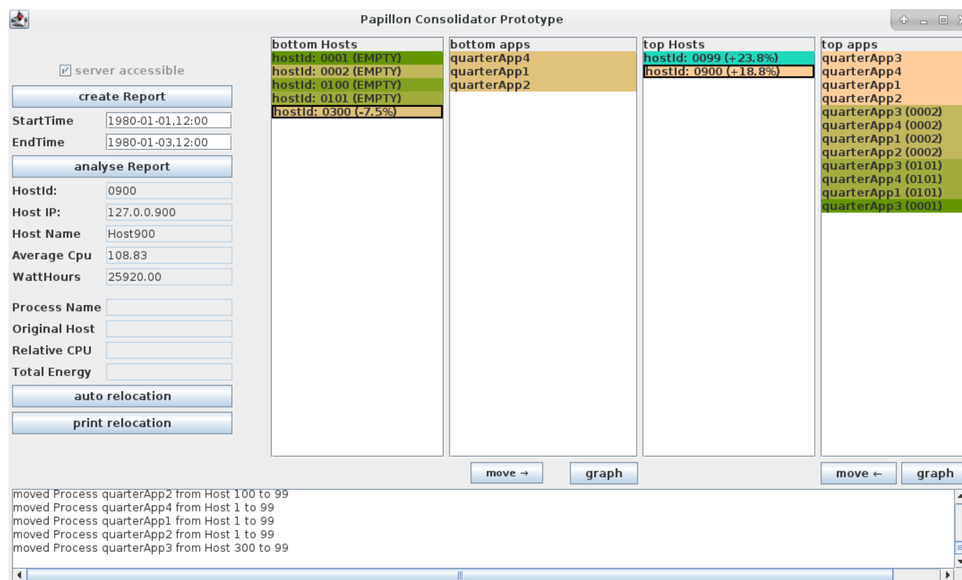

Figure 6.7: Automatic relocation of processes.

6.7 Additional Improvements

Besides the main work on the three main components Client, Server and Consolidator there are two other projects that helped developing and testing the source code: A new build system and a faster server deployment. Both sub projects were part of a preceding topic of this thesis before the actual topic was chosen.

6.7.1 Build System

A new automatic build system was created that compiles the Client, Server and Consolidator software automatically. Due to the fact that all projects are Java based *Ant* [Fou15b], a standard Java build system, was chosen to improve the build process. On each top level directory a file called *build.xml* was created. Executing the command `ant` in that folder results in compiled versions of the software.

Ant build files are XML structured files that, similar to *make* files, address different build targets. All files offer the possibility to compile and bundle the resulting executables with necessary libraries. For the Client and the Server it furthermore implemented an improvement of the existing revision system that looks at two specific source files and names the resulting programs according to the version number inside the source code. Before that the developer had to keep the new version number in mind, the new build system, however, names its executables automatically.

6.7.2 Server Deployment

This is a script bundle that helped installing the server faster. It depends on the previously mentioned build system that can generate binaries by using an automated build system.

It consists of various scripts and archives that contain used software on the server. The structure of the tool set is depicted in figure 6.8. The main script *createPackage.sh* iterates over the software code of all components (Server, Client and Dashboard) and rebuilds them if possible (Dashboard is Javascript code that is interpreted, not compiled). Then it packs the resulting binaries with an install script and two external software projects that are needed on the server (*tomcat* for serving data and *yajsw* for starting the software as a service). The install script *installPapillonserver.sh* is executed on the server and takes an archive as input that holds all necessary files to install the server. It is meant to be run on an Ubuntu Linux OS and therefore uses OS specific install commands. The basic steps of the install script are:

1. **check requirements**: Software packages like *mysql* and *openjdk* are installed if necessary.
2. **data base preparation**: The database is created and filled with some test data.
3. **tomcat**: tomcat is installed and configuration files are changed according to the new paths.
4. **copy Server, Dashboard**: Server and Dashboard are copied, the Client is transferred optionally.
5. **start as a service**: yajsw is used in order to automate the start of the Server.

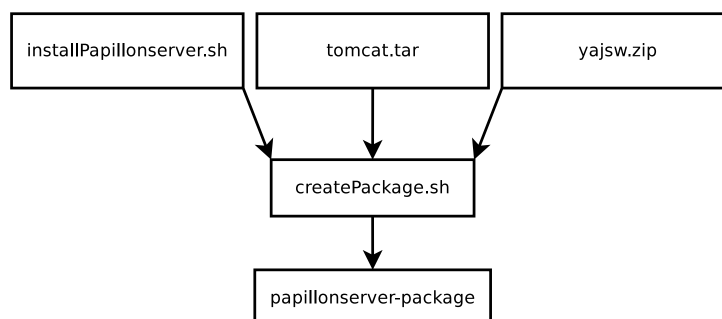

Figure 6.8: The structure of the deployment tool set of Server. The main script takes three components and produces a compressed package that can be executed on the server.

Besides helping Stratergia's developers the deployment tool was successfully used during a practice lecture that was held in cooperation with Stratergia, it supported students installing their version of Papillon on their own servers. The feedback from the students helped putting the source code to its final form.

The next chapter describes future work projects that could be of importance and interest.

7

Future Work

This chapter discusses project ideas that could be of value to Papillon, Stratergia or in general.

7.1 Actual Relocation

The shown relocation functionality gives only statistics of a theoretical situation. A great advancement would be to use a configuration management tool that actually changes configuration recipes and relocates processes between the hosts. This could be implemented inside the Consolidator as a subcomponent.

7.2 Process Grouping into logical Application

Although this thesis showed that a group of processes can be formed by using environment variables there is still a lack of a real process hierarchy throughout the system. This can probably be achieved by focussing on a Linux OS, thus developing a strategy that uses OS specific functionality.

7.3 CPU Load Normalization

The report API call gives information regarding the utilization of attached hosts to the server. Average CPU-loads range from 0 to the amount of installed processors times 100. Due to the fact that the report does not give information regarding the installed processors a user is not aware of the maximum value. What could be done in the future is to change the API-call in order to give normalized values from 0 to 100 or include that information in the report.

7.4 Refine Auto Relocation of Consolidator

The auto relocation feature looks at the less utilized hosts in an arbitrary order and tries to relocate all processes of that host to other, more utilized hosts. There could be additional refinement on how the auto relocation should take place. For example it could be decided that the relocation focusses on certain hosts or tries to move a certain process with higher priority. In addition to that an advanced auto relocation could calculate multiple solutions and give the user the best one. The implemented algorithm only gives the first solution that is adequate. Additional inspiration for refinement can also be found in [MOS12] that uses

constraint programming in order to limit the found constellations to specific constraints for a very similar problem.

7.5 Different Process Costs

Different metrics should be considered when it comes to calculating the impact of a process on the costs. Although this work used CPU based metrics in order to keep its openness there are various alternatives. For instance the financial costs that are used in [MD12] or expanding the possibilities in using modified OSs as it was done in [FS99] and [Zen+02]. Multiple metrics could be evaluated, built into the system and possibly offered as alternative orderings.

7.6 Security Countermeasures

A topic that was not addressed so far by Stratergia is the security of Papillon. Unfortunately there is no responsible layer that could take over certain tasks like encryption of statistics or authentication on the server. Although there was no need for these features data centre operators would definitely appreciate the software that is meant to run on their machines to be highly secured against various attack vectors.

7.7 Intelligent Process Tracking

So far the data centre operator still has to put quite some thinking into tracking single processes or application groups. The operator has to find out on his own what the most interesting processes are to him. Of course the monitoring of the top three applications is of some help but there is room for improvement. Clients could send all processes on demand and these processes are then analysed for a certain period of time. Different metrics can be applied that result in marking several processes as interesting. If a client can accept commands that tell them to track certain processes Papillon could build up a network of clients which processes are monitored automatically.

7.8 Remote Tracking of Processes

Based on the previous project an interesting feature of Papillon would be to track processes remotely. Clients would have to accept commands from other machines that tell them to track certain processes. This would result in major modifications of the existing code base due to the fact that the client software is at the time of writing only sending to the server and not receiving from him.

8

Concluding Remarks

In this thesis I modified and extended parts of the Papillon DCIM by Stratergia. The main work was to find problems in the existing system and solve them in an appropriate way. The main focus was to enable a data centre operator to get more insight into the process' influence that are executed on his machines - this was made possible by changing the existing components and introducing a new component called Consolidator. The system benefited in several ways: It is possible to track single processes and calculate its energy requirements and therefore the majority of its costs. Furthermore the system can now form logical groups of applications that are set by the user who knows what processes belong together. After assigning the same tag to them the system is able to monitor all included processes as a whole unit. And finally a data centre operator received a visual help of his servers that maps processes to machines and helps relocating those mappings. The relocations' impact is presented and if the result is acceptable the new mappings can be exported and passed on.

After giving an introduction to the need of these extensions the reader received some necessary fundamentals in chapter 2. After that State of the Art was defined by explaining the existing Papillon system and other related research projects in chapter 3. In the same chapter the system was analysed and typical shortcomings were listed. In the next chapter the design of all necessary modifications and metrics were given. Subsequently these decisions were put into practice in chapter 5 which described the implementation process. Chapter 6 shows results of the solutions to the previously listed shortcomings. The thesis is concluded by possible future projects and an user manual of the Consolidator next to the explanation of the used data base script for artificial data in the appendix.

9

Appendix

9.1 Consolidator User's Manual

This section describes how the Consolidator is used and what information can be gained from it. This part is given to Stratergia as a standalone document.

9.1.1 Functionality

The Consolidator fulfills following functionalities:

- View hosts with high and low average utilisation.
- Give information about the running processes on hosts.
- Relocate processes manually and visualize the estimated impact.
- Relocate processes automatically in order to free hosts from recorded processes.
- Give costs of processes in terms of Watt Hours and share of CPU utilization.

9.1.2 Requirements and Shipping

The development folder of papillonconsolidator holds a build.xml ant file. Opening a Linux compatible console there and typing `build` should create a shippable zip-folder inside the newly created folder dist. The zip folder contains the properties file, all needed libraries and the jar-file. It can be executed by typing `java -jar papillonconsolidator.jar`

The user has to make sure that the information supplied in the properties file are correct and the papillonserver can be reached via the network.

9.1.3 Configuration of Consolidator

Following Parameters need to be specified in the properties file:

- `masterIp`,`masterPort`: The IP address and port number of the Papillon server.
- `datacenterId`,`datacenterName`: The identifier and the name of the data centre that should be analysed
- `relocationCpuLimit`: The maximum CPU load for automatic relocation. If a user does a manual relocation and exceeds this limit a warning will be given.

- `maxUtilizationBottomhost`: The maximum average utilization that is needed in order to qualify for being a less utilized host(called bottom host). The minimum is zero, the used unit is percentage.
- `minUtilizationTophost`: The minimum average utilization that is needed in order to qualify for being a more utilized host (called topHost). The maximum is not specified because the range of this number depends on the amount of installed processors. A four core system has a maximum value of 400%.

9.1.4 Server Configuration

One has to make sure that the number of reported hosts on the Papillon server are configured correctly. If only two hosts are reported this will result in two bottom and two top hosts. Another thing the user has to keep in mind is that the Server properties file holds its own thresholds for bottom and top hosts. It is better to keep those values as tolerant as possible so that Consolidator receives as many hosts as possible. As it was explained previously the Consolidator's limits can be changed quickly without causing any trouble whereas the Server's limits are only activated after a restart.

9.1.5 GUI Overview

In figure 9.1 one can see the GUI of the Consolidator.

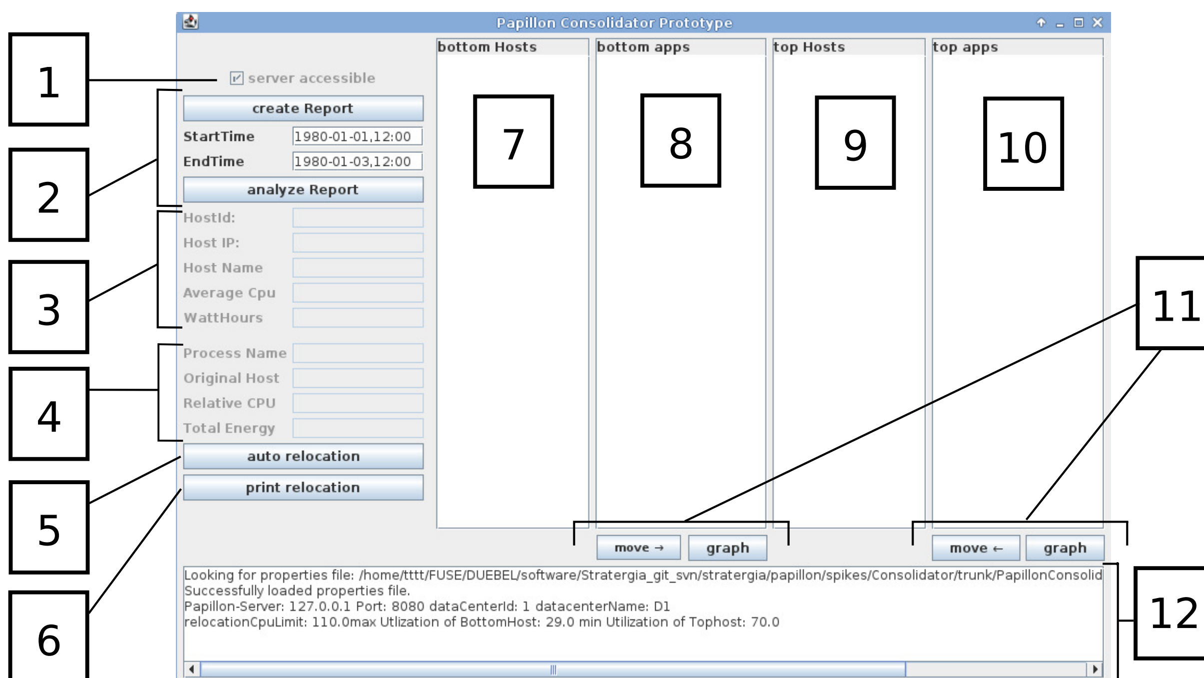

Figure 9.1: Overview of Papillon GUI

It consists of following elements, the numbers represent annotations in the figure:

1. Server Check
    - This check box is ticked if the Papillon server, specified by the properties file is accessible.

- It will be ticked if the /rest/information - page is reachable without any Java exception thrown.

2. Create and analyse report over given time
   - Reports hold the utilisation of hosts and therefore aid the decision which hosts are to be analysed.
   - In order to prevent the creation of too many Reports (they are saved on the server's hard disk) a report has only to be created once.
   - If the analyse button is clicked the most recent report is chosen.
   - If analyse button is pressed again after all processes are queried then all List-Items are emptied and analysis of report starts again.
   - The start Time and end Time represent the analysed time period. It is expected that every host was reporting its processes throughout this time with one sample each minute.

3. Host information
   - This part gives information regarding the chosen host. ID, IP Address and Name are self explanatory.
   - Average CPU: gives the average CPU load over the specified amount of time. This value is reported from the report and ranges between 0 and number of cores times 100.
   - Watt Hours: the accumulated Watt hours that were reported by the report.

4. Process information
   - This part is similar to the host Information but analyses the selected process. Process name is self explanatory.
   - The original host field shows the host that actually ran that process during the time of recording. Because relocations of processes between hosts are possible the original host has to be saved in order to summarize the relocations.
   - Relative CPU: The share of the selected process' in terms of CPU usage. Example: Let the average Utilisation of a host over a certain amount of time be 10%, if four applications are running for this time and all of them have the same share (25%) then the cpu contingent of every process is 10%/4 = 2.5%.

     It is assumed that all servers have roughly the same specifications in such a way that x% load on host A are equal to x% on host B.
   - Total energy: That value is the energy consumed by that process. On every sample point the total Energy consumption by that host is summed up according to the process's share of CPU. The value's unit is Watt Hours.

5. Auto relocation
   - Auto Relocation helps the user relocate the processes.
   - It is a greedy algorithm that takes the first bottom host (less utilized) in the list representation and tries to move all of its process entries to be relocated to a top host. If no more processes of that host can be relocated the algorithm moves on to the next host up until all less utilized hosts are handled.
   - Auto Relocation will not take place if a relocation would result in exceeding the given limit from the properties file.

6. Print relocation
   - prints the relocations that were done either automatically or manually to the console and a file called relocation_summary.txt
7. Bottom hosts
   - This List holds all hosts that were reported by the Reports-API-call and within the limit specified by the properties - file.
   - If a host was reported with low utilisation but its utilisation is higher than specified a warning will be written to the Console-Log (12).
   - If a host is selected its information is displayed on the host information part (3).
   - In addition to that the bottomProcesses List (8) will be updated according to the selected host.
   - If the list is clicked with a right click the selection will be cleared. This is helpful if only one host is available.
8. Bottom apps
   - In this list all the processes are listed that were recorded throughout the given time period.
   - Only processes that were either tracked via name or tagged via a environment variable.
9. Top Hosts
   - Similar to (7) but lists higher utilised hosts.
   - Minimum load is specified in properties file.
10. Top apps
    - similar to (8).
11. *move* and *graph*
    - The button *graph* shows a visualisation of the recorded Applications, for more information see section 9.1.6.
    - The button *move* will relocate the process to a new host and compute the new CPU loads and power values. More Information on relocation of processes can be found at section 9.1.7.
12. Console
    - The console shows the current status and gives the user additional information about progress and relocations.

9.1.6 Visualisation as Chart of Processes

If a process is selected the *graph* button below the list creates a new panel that holds a chart which gives additional information regarding the process events that were monitored for that process. The chart is controllable by right and left clicks of the mouse and its view can be decreased or increased by using the mouse wheel. The chart holds a multiple y axis that consists out of three axis:

- **power**: The power value of the whole host at the moment of recording in Wh.

- **cpu contingent**: The fraction of CPU load this process is responsible for in % ranging from 0 - 100.
- **cpu stat**: The CPU load of the host in % ranging from 0 - number of cores times 100.

9.1.7 Manual Relocation of Processes

In order to relocate processes between hosts a source host, a target host and the process have to be selected. Then the *move* button transfers the process from source host to target host. The host entries are marked with the change in CPU load automatically.

9.2 Population of database

During the thesis a script was created that populated the database with artificial entries. This script is explained in this section.

The basic structure is depicted in figure 9.2. An existing SQL script, that was used during installation of the Server, is used as template. This file holds all tables and fills them with some test data. A Python script creates temporary files with artificial data entries which are then used by a Bash script that replaces the old data entries in the template SQL script with the artificial entries. The product of this replacement is saved as a new SQL script that can be executed on the server. The resulting file is about 10 megabytes in size - automatic copying is therefore important because most text editors can't handle such big files without problems.

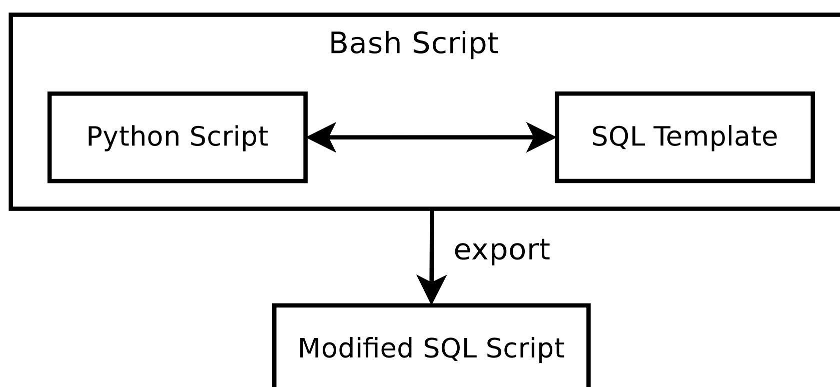

Figure 9.2: The basic structure for the scripts that populated the database with artificial values.

The python script includes a list of host numbers which are three digit numbers, average loads can be calculated by multiplying the highest digit with the number 10. For example a host with ID 900 should have an average load of 90%. The list of host numbers are iterated and for every host following table entries are created:

- **Host**: Every host needs an entry in the database according to its name, this includes name, ID, rack, floor and processor count. The IP address is assigned according to their ID, the other static values that do not depend on the host ID are set to one due to the fact that they are not used by the Consolidator.
- **Activity**: One activity entry stands for one record that should occur once a minute. In order to separate real from artificial entries a past start date was chosen. From that point enough entries were created to fill two full days. For every day there is an entry for every

minute - therefore $24 * 60 * 2 = 2880$ entries are created. The python script calculates the correct monitoring times according to the first date and the average CPU based on the host ID. The sum of all CPU cycles used by the reported applications is set to 1000.

- **App**: For very activity entry there is at least one entry that lists the recorded applications. For the artificial database every activity is linked to four A app entries. Each of them has the same share of CPU cycles which is in that case 250. It is assumed that these entries are tracked by name - the entry receives the according property.

After the completion of the Python script the Bash script replaces the newly created values with place holders inside the SQL template file. The product is a script that can be executed on the server and results in a clean and working data base that can be used for development and testing.

Abbreviations

API Application Programming Interface. iii, vi, 8–11, 24–26, 31, 34–37, 42, 43, 47, 49, 56

ARP Address Resolution Protocol. 16

BPP Bin Packing Problem. 16, 28

BSD Berkeley Software Distribution. 14

CPU Central Processing Unit. i, iv, v, 3, 6, 8, 10, 12, 14–16, 18, 21–23, 25–27, 33, 36–40, 43–50, 53, 55, 56

DAO data access object. 35

DCIM Data Center Infrastructure Management. 8

DVFS Dynamic Voltage and Frequency Scaling. i, 4

EVs Environment Variables. 5

GUI Graphical User Interface. iv, 24, 30, 33, 41, 42, 54

HMM Hidden Markov Model. 17

HTTP Hypertext Transfer Protocol. 8, 24, 31

IDE Integrated Development Environment. 20

IO Input/Output. 14

IP Internet Protocol. 10, 16, 26, 40, 53, 55

JSON JavaScript Object Notation. 9, 25, 31, 41

LAN Local Area Network. 16

MAC Media Access Control. 16

NAS network attached storage. 16

NIC Network Interface Controller. 15, 16

OS Operating System. 5, 9, 14, 15, 20, 22, 24, 33, 34, 40, 46, 49, 50

PAPILLON Profiling and Auditing of Power Information in Large and Local Organisational Networks. 7

PID Process Identifier. 14, 20, 21, 37

POSIX Portable Operating System Interface. 40

PUE Power Usage Effectiveness. 26

REST Representational State Transfer. 8, 31

RGB Red Green Blue. 42

SI International System of Units. 3

SQL Structured Query Language. 9, 10, 31, 36, 37

URL Uniform Resource Locator. 31, 32, 35, 42

UTC Coordinated Universal Time. 41

VM Virtual Machine. 15–17

WAN Wide Area Network. 15

XML Extensible Markup Language. vi, 9, 25, 31, 36, 37, 41

Bibliography

[App15] Apple. *Environment Variables Mac OSX*. Mar. 2015. https://developer.apple.com/library/mac/documentation/MacOSX/Conceptual/BPRuntimeConfig/Articles/EnvironmentVars.html (cited on page 7).

[BH07] Luiz André Barroso and Urs Hölzle. "The case for energy-proportional computing". In: *IEEE computer* 40.12 (2007), pages 33–37 (cited on page 1).

[BR04] Ricardo Bianchini and Ramakrishnan Rajamony. "Power and energy management for server systems". In: *Computer* 37.11 (2004), pages 68–76 (cited on pages 5, 6).

[Cla+05] Christopher Clark et al. "Live Migration of virtual Machines". In: *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association. 2005, pages 273–286 (cited on page 19).

[Cor15] Oracle Corporation. *Jersey Website*. Mar. 2015. https://jersey.java.net/ (cited on page 36).

[EKR03] Mootaz Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. "Energy Conservation Policies for Web Servers". In: *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*. USITS'03. USENIX Association, 2003, pages 8–8 (cited on page 6).

[FS99] Jason Flinn and Mahadev 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.* IEEE. 1999, pages 2–10 (cited on pages 17, 58).

[Fou15a] Linux Foundation. *Xen Project*. Apr. 2015. http://xenproject.org/ (cited on page 19).

[Fou15b] The Apache Software Foundation. *Apache Ant Website*. May 2015. https://ant.apache.org/ (cited on page 55).

[Goo15] Google. *Window Builder Website*. Apr. 2015. https://www.eclipse.org/windowbuilder/ (cited on page 38).

[Gre+08] Albert Greenberg et al. "The cost of a cloud: research problems in data center networks". In: *ACM SIGCOMM computer communication review* 39.1 (2008), pages 68–73 (cited on pages 1, 5).

[Ham15] Uni Hamburg. *Definition of Joule*. Mar. 2015. http://www.sign-lang.uni-hamburg.de/tlex/lemmata/l3/l334.htm (cited on page 4).

[Hyp14] Hyperic. *Sigar Project Website*. Dec. 2014. https://support.hyperic.com/display/SIGAR/Home (cited on page 23).

[Kin14] Colin King. *Stress-ng Website*. Dec. 2014. `http://kernel.ubuntu.com/~cking/stress-ng/` (cited on page 39).

[KVN10] Ricardo Koller, Akshat Verma, and Anindya Neogi. "WattApp: An Application Aware Power Meter for Shared Data Centers". In: *Proceedings of the 7th International Conference on Autonomic Computing*. ICAC '10. ACM, 2010, pages 31–40 (cited on pages 16, 17).

[Kri15] Thomas Kriechbaumer. "Optimisation and Analysis of Full-System Power Models". In: (2015) (cited on pages 9, 42).

[Lim15] Object Refinery Limited. *JFreeChart Website*. Mar. 2015. `http://www.jfree.org/jfreechart/` (cited on page 31).

[Mar15] Angelo Marletta. *cpulimit Website*. May 2015. `http://cpulimit.sourceforge.net/` (cited on page 39).

[MBP11] Moreno Marzolla, Ozalp Babaoglu, and Fabio Panzieri. "Server consolidation in clouds through gossiping". In: *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a*. IEEE. 2011, pages 1–6 (cited on page 21).

[MD12] Michele Mazzucco and Dmytro Dyachuk. "Optimizing Cloud Providers Revenues via Energy efficient Server Allocation". In: *Sustainable Computing: Informatics and Systems* 2.1 (2012), pages 1–12 (cited on page 58).

[MOS12] Deepak Mehta, Barry O'Sullivan, and Helmut Simonis. "Comparing solution Methods for the Machine Reassignment Problem". In: *Principles and Practice of Constraint Programming*. Springer. 2012, pages 782–797 (cited on pages 34, 57).

[myb15] mybatis.org. *MyBatis Website*. Mar. 2015. `mybatis.org` (cited on page 36).

[Ora15a] Oracle. *About Java*. Mar. 2015. `https://www.java.com/en/about/` (cited on page 28).

[Ora15b] Oracle. *Java Documentation String*. Apr. 2015. `https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#hashCode()` (cited on page 43).

[OLG10] A-C Orgerie, Laurent Lefevre, and J-P Gelas. "Demystifying Energy Consumption in Grids and Clouds". In: *Green Computing Conference, 2010 International*. IEEE. 2010, pages 335–342 (cited on page 6).

[Pro15a] Apache Tomcat Project. *Apache Tomcat Website*. Mar. 2015. `https://tomcat.apache.org` (cited on page 11).

[Pro15b] XStream Project. *XStream Website*. Apr. 2015. `http://xstream.codehaus.org/` (cited on page 28).

[Res15] Mersenne Research. *Prime 95 Website*. Apr. 2015. `http://www.mersenne.org/download/` (cited on page 39).

[Rus15] Mark Russinovich. *Process Explorer Website*. Apr. 2015. `https://technet.microsoft.com/en-us/sysinternals/bb896653` (cited on page 51).

[Rut09] Stephen Ruth. "Green it more than a three percent solution?" In: *Internet Computing, IEEE* 13.4 (2009), pages 74–78 (cited on page 1).

[Sca06] Jed Scaramella. "Worldwide server power and cooling expense 2006-2010 forecast". In: *International Data Corporation (IDC)* (2006) (cited on page 1).

[SPH07] David C Snowdon, Stefan M Petters, and Gernot Heiser. "Accurate on-line prediction of processor and memory energy usage under voltage scaling". In: *Proceedings of the 7th ACM & IEEE international conference on Embedded software*. ACM. 2007, pages 84–93 (cited on page 18).

[Sno+09] David C Snowdon et al. "Koala: A Platform for OS-level power management". In: *Proceedings of the 4th ACM European conference on Computer systems*. ACM. 2009, pages 289–302 (cited on page 18).

[Squ15] Square. *okhttp Website*. Apr. 2015. https://square.github.io/okhttp/ (cited on page 28).

[SKZ08] Shekhar Srikantaiah, Aman Kansal, and Feng Zhao. "Energy Aware Consolidation for Cloud Computing". In: *Proceedings of the 2008 Conference on Power Aware Computing and Systems*. HotPower'08. USENIX Association, 2008, pages 10–10 (cited on pages 20, 33).

[SS615] SS64.com. *Windows Environment Variables*. Mar. 2015. http://ss64.com/nt/syntax-variables.html (cited on page 7).

[Str11] Stratergia. "Papillon Environment Setup". 2011 (cited on page 38).

[Str14] Stratergia. "Papillon API Reference". 2014 (cited on page 36).

[Tra+06] Franco Travostino et al. "Seamless live Migration of virtual Machines over the MAN/WAN". In: *Future Generation Computer Systems* 22.8 (2006), pages 901–907 (cited on page 19).

[Upt15] Uptime-Institute. *Data Center Industry Survey 2013*. May 2015. http://uptimeinstitute.com (cited on pages 5, 6).

[Ver+10] Akshat Verma et al. "Brownmap: Enforcing Power Budget in shared Data Centers". In: *Middleware 2010*. Springer, 2010, pages 42–63 (cited on page 1).

[Wik15a] The Free Encyclopedia Wikipedia. *Bin packing problem*. Mar. 2015. https://en.wikipedia.org/wiki/Bin_packing_problem (cited on page 33).

[Wik15b] The Free Encyclopedia Wikipedia. *Epoch Time*. Mar. 2015. https://en.wikipedia.org/wiki/Unix_time (cited on page 41).

[Wik15c] The Free Encyclopedia Wikipedia. *Representational state transfer*. May 2015. https://en.wikipedia.org/wiki/Representational_state_transfer (cited on page 7).

[ZEL05] Heng Zeng, Carla Schlatter Ellis, and Alvin R Lebeck. "Experiences in managing Energy with ecosystem". In: *Pervasive Computing, IEEE* 4.1 (2005), pages 62–68 (cited on page 18).

[Zen+02] Heng Zeng et al. "ECOSystem: Managing Energy as a first Class Operating System Resource". In: *ACM SIGPLAN Notices*. Volume 37. 10. ACM. 2002, pages 123–132 (cited on pages 18, 58).

[ZA10] Jiedan Zhu Qian and Gagan Agrawal. "Power-Aware Consolidation of Scientific Workflows in Virtualized Environments". In: *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '10. IEEE Computer Society, 2010, pages 1–12 (cited on page 21).