Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Mark Sevalnev

Applying Queueing Theory to Computing Cluster Energy Optimization

Master's Thesis
Espoo, January 24, 2012

Supervisor: Professor Keijo Heljanko
Instructor: Professor Samuli Aalto and Dr. Tapio Niemi

Aalto University
School of Science

Aalto University
School of Science

ABSTRACT OF
MASTER'S THESIS

Author:	Mark Sevalnev
Title of thesis:	Applying Queueing Theory to Computing Cluster Energy Optimization
Date:	January 24, 2012 **Pages:** 9 + 105
Professorship:	Theoretical Computer Science **Code:** T-79
Supervisor:	Professor Keijo Heljanko
Instructor(s):	Professor Samuli Aalto and Dr. Tapio Niemi

This work examines the current state of green IT and introduces a new software solution for saving energy in computing clusters. The work includes a comprehensive literature review on green IT by offering solutions to both computing clusters to improve energy efficiency and, more generally, other information technology solutions targeted to minimize energy consumption. It is known that the information technology systems resulted in 2% of all carbon dioxide emissions in 2007. Green IT focuses on developing of solutions to improve energy efficiency. Most solutions are based on the idea that the computing equipment is used only occasionally, but the device still consumes electricity while in the waiting mode. This necessitates the identification of such idle periods and to switch the device into a power saving mode. The work will go fundamentally through the necessary theoretical background: queuing theory, control theory, and statistical testing, so that a solution can be understood without knowledge of background literature. The solution for computing cluster energy minimization was also experimented with in practice.

Keywords:	cloud computing, datacenter, energy-proportionality, green IT, Grid, power-performance trade-off, quality of service, queueing theory
Language:	English

Aalto-yliopisto
Perustieteiden korkeakoulu

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Mark Sevalnev

Työn nimi:
Jonoteorian soveltaminen laskentaklusterin energiakäytön optimoimiseksi

Päiväys: 24. tammikuuta 2012 **Sivumäärä:** 9 + 105
Professuuri: Tietojenkäsittelyteoria **Koodi:** T-79
Työn valvoja: Professori Keijo Heljanko
Työn ohjaaja: Professori Samuli Aalto ja FT Tapio Niemi

Tässä työssä tarkastellaan vihreän IT:n nykytilaa sekä esitetään uusi ohjelmistoratkaisu energian säästämiseen laskentaklustereissa. Työ sisältää laajan kirjallisuuskatsauksen vihreään IT:hen — siinä esitetään ratkaisuja sekä laskentaklustereiden energiatehokkuuden parantamiseksi että yleisemmin muita tietoteknisia ratkaisuja, joilla sähkökulutusta voidaan minimoida. Kirjallisuudesta tiedetään, että tietotekniset järjestelmät tuottivat 2% kaikista hiilidioksidipäästöistä vuonna 2007. Vihreässä IT:ssä kehitetään ratkaisuja energiatehokkuuden parantamiseksi. Useimmat ratkaisut perustuvat siihen havaintoon, että sähkölaitetta käytetään vain ajoittain, mutta laite silti kuluttaa sähköä ollessaan odotustilassa. Työssä kehitetyillä menetelmillä pyritään havaitsemaan tällaisia ajanjaksoja, joissa voidaan ohjata laite vähäenergiseen tilaan. Työssä käydään perusteellisesti läpi tarvittava teoreettinen tausta: jonoteoria, säätöteoria ja tilastollinen testaus, jotta oma ratkaisu voi ymmärtää ilman taustakirjallisuutta. Ratkaisun toimivuutta laskentaklusterin energian säästämiseksi on työssä kokeiltu myös käytännössä.

Avainsanat: energiaverrannollisuus, energia-tehokkuus-kompromissi, Grid, jonoteoria, palvelun laatu, pilvilaskenta, tietokonesali, vihreä IT
Kieli: englanti

Kiitokset

Haluan kiittää valvojaani Keijo Heljankoa, ohjaajaani ja työnantajaani Tapio Niemeä sekä ohjaajaani Samuli Aaltoa. Keijo Heljangolta sain diplomityöhöni erittäin paljon ajatuksia ja parannusehdotuksia. Tapio Niemi ohjasi hienosti työtäni CERN:ssa, ja myös työn ulkopuolella hän huolehti, että asumaan asettumiseni Ranskaan olisi mahdollisimman stressitöntä. Sain erittäin paljon motivaatiota Samuli Aallon opettamasta jonoteorian kurssista keväällä 2011. Hän myös selitti minulle vaikeita matemaattisia asioita niin, että lopulta osasin ne.

Työpaikalla CERN:ssä sain korvaamatonta apua Jukka Kommerilta klusterin rakentamiseen. Kiitän myös muita työtovereita leppoisasta ja motivoivasta työilmapiiristä, kiitos Henrille, Markolle, Eetulle, Jonille, Ollille ja Antille.

Diplomityön tarkistuksessa minua auttoivat Pia Lappalainen, Ilkka Ollakka ja Mikael Isoaho. Kiitos teille!

Diplomityötä tehdessäni, minulla oli ajoittain kysyttävää jostakin erityisalueesta, silloin marssin kyseisen alueen asiantuntijalle, ja sain vastauksia kysymyksiin. Suurkiitos Aalto yliopiston henkilökunnalle, joka auttoi minua näissä pulmissa, kiitos Pasi Lassilalle, Jukka K. Nurmiselle, Kai Zengerille, Jukka Parviaiselle, Jukka Mantereelle ja Vesa Hirvisalolle.

Viimeisimpänä mutta ei vähäisimpänä haluan kiittää rakasta perhettäni ja ihania ystäviäni, joita ilman en jaksaisi opiskella näin pitkälle puhumattakaan diplomityön kirjoittamisesta.

Minua auttoivat välittömästi tai välilliseti suuri joukko muita ihmisiä. Kaikkien nimiä en välttämättä edes muista, kiitos teillekin!

Espoo 24. tammikuuta 2012

Mark Sevalnev

Abbreviations and Acronyms

M/M/1-queue	the queue with Poisson arrivals and Exponential service times
M/G/1-queue	the queue with Poisson arrivals and general service times
λ	arrival rate
μ	service rate
ρ	utilization
π	state probability
AC	Alternating Current
CERN	(French) Conseil Européen pour la Recherche Nucléaire (European Council for Nuclear Research)
CPU	Central Processing Unit
CTwQA	Control Theory with Queue Adjusting
DC	Direct Current
DCiE	Datacenter Infrastructure Efficiency
DFS	Dynamic frequency scaling
DPPE	Datacenter Performance Per Energy
DVFS	Dynamic Voltage and Frequency Scaling
DVFS-WD	Dynamic Voltage and Frequency Scaling based on Workload Decomposition
EEE	Energy Efficient Ethernet
FIFO	First In First Out
FIR	Finite Impulse Response
GB	GigaByte
GHz	GigaHertz
HIP	Helsinki Institute of Physics
I/O	Input/Output
IaaS	Infrastructure as a Service
ICT	Information and Computer science
IIR	Infinite Impulse Response

LAN	Local Area Network
LCP	Lazy Capacity Provisioning
LHC	Large Hadron Collider
LIFO	Last In First Out
MMPP	Markov Modulated Poisson Process
OLDI	Online Data-Intensive
PaaS	Platform as a Service
PDU	Power Distribution Unit
PID	Proportional-Integral-Derivative
PMR	Peak-to-Mean Ratio
PMU	Performance Monitoring Unit
PS	Process Sharing
PUE	Power Usage Efficiency
QTwAF	Queueing Theory with Averaging Filter
QTwEF	Queueing Theory with Exponential Filter
RAILS	Redundant Array for Inexpensive Load Sharing
SGE	Sun Grid Engine
SLA	Service Level Agreement
SPC	Storage Performance Council
SPEC	Standard Performance Evaluation Corporation
SPUE	Server PUE
SaaS	Software as a Service
TCP	Transmission Control Protocol
TPC	Transaction Processing Performance Council
UPS	Uninterruptable Power Supply
i.i.d.	identically and independently distribution

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Nowadays Information and Communication Technology (ICT) starts to be as global and ubiquitous infrastructure such as electricity. Almost every data processing device has an access to Internet and probably initiates a great amount of computation, although this might be totally invisible to an end-user. The recent trend has been the switch towards cloud computing. All kind of computation such as web search indexing, data analyzing, business transactions, scientific simulations, which are either valuable in themselves or are executed as a byproduct of user-side actions, are performed in huge datacenters full of computing nodes. End-users' devices start to get the role of request senders and response receivers, and computation is transferring to datacenter side.

Datacenters vary in size from a single rack server room to several hundreds or even thousands servers containing halls. For instance, the most famous ICT company owning datacenters and providing cloud computing services — Google has 31 datacenters worldwide [1]. It was estimated that in 2007 the construction of a single datacenter cost for Google constituted $ 600 million, that includes investment for construction, infrastructure and servers [2]. But this is not the whole picture of expenses, namely datacenters of a such size are enormous electricity consumers, drawing continuously about 260 MWs [3]. Another example is Digital Realty Trust in Chicago has a 100 MW power feed and a datacenter campus in Ashburn, Virginia that is wired for 225 MW [4]. Assuming that such datacenter operates at its full capacity for an hour,

[1] http://royal.pingdom.com/2008/04/11/map-of-all-google-data-center-locations/

[2] http://www.datacenterknowledge.com/google-data-center-faq-part-2/

[3] http://www.nytimes.com/2011/09/09/technology/google-details-and-defends-its-use-of-electricity.html

[4] http://www.datacenterknowledge.com/google-data-center-faq-part-2/

it brings the expenditure of approximately $ 26000 (we take the cost of the electricity 0.117 $/kWh [5], thus 0.117 $/kWh * 225000 kWh = 26325 $). That is why it is not a surprise that datacenters are designed optimal energy use in mind and new solutions are sought continuously.

It was estimated in the SMART report that ICT is responsible for 2% of global carbon emissions in 2007 and it is estimated to go grow to 6% in 2020 (see Figure 1.1) [59]. Datacenters are not even the biggest consumers of electricity: from the same report we see that datacenters contribute to 14 % of the all ICT carbon emissions in 2007 [59]. Green IT is an emerging field which seeks solutions to energy optimization in ICT. Many achievements can be done just by increasing the knowledge about the current situation in ICT power consumption. For example the survey conducted by Fujitsu, which summarizes 1000 responses to questions about green IT policies, reveals that almost the half of the respondents either have never thought about the electricity bills in their companies or simply do not know how expensive electricity consumption could be. Further, they found out that there is a strong correlation between the visibility of the ICT power bill and the company's green IT efficiency [11].

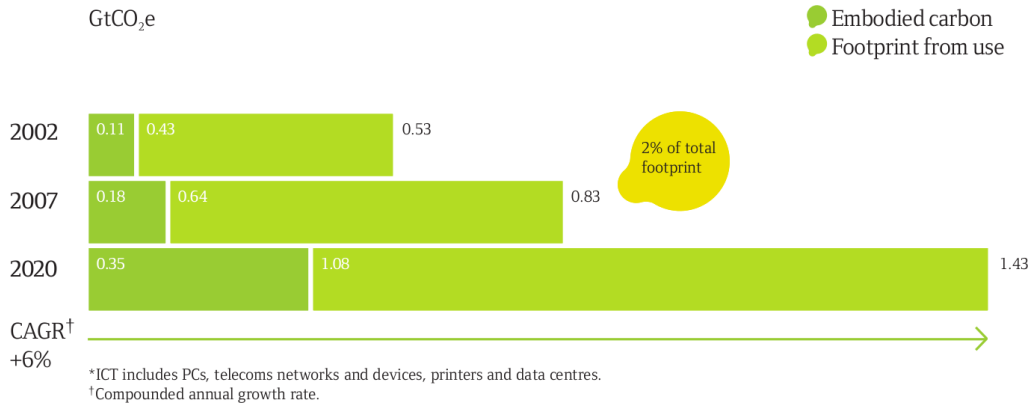

Figure 1.1: The contribution of ICT carbon emissions and its growth [59]

The aim of this Master's Thesis is twofold. First, we want to give an overview of green IT concentrating mainly on power efficiency of datacenters. We tell about the current state of solutions for improving power efficiency of datacenters and introduce academic research in the field. We also give the picture of the cloud computing ecosystem, which inevitably affects datacenter design. It is important to understand that power efficient datacenters are not the

[5] http://www.eia.gov/

ultimate goal. ICT companies are seeking for revenues by selling services provided by datacenters and by minimizing the cost caused by running those datacenters. Thus it is not enough to minimize only operating costs but instead it is worth to try to achieve a trade-off between the service availability and efficient datacenters design. Second, we propose our solution for improving datacenter power efficiency. We apply the state-of-the-art theory in performance analysis in communication systems — queueing theory.

Queueing theory has been extensively applied as a model for different telecommunication systems which provides service for randomly arriving customers with the specified quality of service. Queueing theory binds together the concepts of service capacity, customers' arrival intensity and a quality of service in an elegant and simple way such that their dependencies could be expressed with equations. Given two of the three parameters, the third can be derived of the two. Queueing theory has been mostly used in static environments, in which the customers' arriving intensity is assumed to have a constant value. This assumption leads to very convenient results, wherein we get the required service capacity when quality of service is fixed. But in reality an arrival distribution seldom remains unchanged. As long as the variations are small, a distribution with constant arrival rate is acceptable as an approximation. In computing clusters the workload could be very 'spiky'. That is why we introduce quite a new model, in which we let the customers' intensity to vary. Naturally, this not only captures better the behavior of the real customers but also brings us many new challenges, such as the increased uncertainty due to approximations and the difficulties to assess real values in changing settings. But regardless of the difficulties, our study makes possible to adjust computing resources to the level which is 'just enough' for the current customers' demand. By this approach we can significantly increase the utilization of the computing nodes in a datacenter and thus gain power savings. In order for our solution to be applied it is necessary to fulfill a number of conditions regarding the arrival and service time distribution. We analyzed data logs gathered from real computing clusters, performed statistical test on that data and argue how well the data fits into our models.

We implemented our approach with three algorithms written in Bash. The algorithms perform resource management in a data cluster by switching on or switching off computing nodes based on the current customers' arrival rate. We also tested our solution by executing long runs and measuring electricity consumption of servers. The test settings were as close to the real situation as possible — we used real resource management software (Sun Grid Engine, Bash-scripts), the jobs were CPU-intensive benchmarks and the customers' arrival was simulated realistically. Those preliminary test

runs show remarkable savings in electricity and thus the purpose of this work is fulfilled — it was attested that queueing theory is applicable to the problem. Because both the theory is large and the optimization targets are numerous, we could not investigate all the aspects of the approach in detail. We leave suggestions for further improvements to the future research.

This Thesis is constructed as follows. In Chapter 2 we give an overview of cloud computing. Cloud computing is not just a collection of datacenters full of servers spread all over the world. It is new concept in ICT business with its own advantages and challenges. Because this research was founded by Helsinki Institute of Physics at CERN, in the last Section of the Chapter we describe which connections CERN has to cloud computing.

In Chapter 3 we give an overview of green IT. ICT ranges from end-users' devices such as cell phones and PCs coming to data transmission infrastructure such as Internet, LANs and phones base stations and ending to large datacenters. It is important to remember that green IT is searching for energy efficient solutions on all those sides, although in our study we concentrate on datacenters. Also the Chapter is important because the motivation behind our solution is given there. It is very recommended to read at least Sections 3.2 and 3.3, in which we argue why the problem of servers poor utilization is worth to be considered.

In Chapter 4 we present work done in green IT concerning datacenter power efficiency improvements. Solutions are either hardware-driven or software-driven depending on which components are used to achieve power efficiency.

Chapter 5 familiarizes with the basics of queueing theory, control theory and statistical testing needed for understanding of our approach. In the Section describing queueing theory we have a short introduction to the modeling of computing systems with the aid of queueing theory. This could be understood as a related work but presented later because of the need to understand first the appropriate theoretical background.

In Chapter 6 we analyze a real world data and see how well it is applicable to our models.

In Chapter 7 we present our solution and the alternative ways to model the problem of improving datacenter utilization. The last Section is somewhat the summary of our investigations.

In last Chapters 8, 9 and 10 we give the concrete description of the implementation for our approach, then we present results for the test runs and conclude the thesis. Improvements and further investigations are presented there.

Chapter 2

Overview of cloud computing

In this Chapter we try to describe what cloud computing is and give four views at it. First, what kind of new business opportunities cloud computing brings. Second, what new technology cloud computing requires. We discuss the structure of datacenters and what challenges their use bring. Third, we look at scientific version of cloud computing — the Grid, which differs slightly in use from business cloud computing. Finally, we discuss why CERN requires Grid.

2.1 Cloud computing

In the recent years there has been a comprehensive movement towards cloud computing in the ICT field. Cloud computing is a concept which comprises both the economic and technical perspectives. Economic view comprises all policy, methodology and social changes in the way ICT companies offer their services and end-users perceive those services. Technical view describes the hardware and software architecture required to make the new services profitable for companies and suitable in cost for customers.

Let us examine the whole supply chain of cloud computing field and see what kind of new possibilities are arising from it for end-users and companies. We will employ the classification used in [15] and divide the supply chain into three parts — end-users or Software as a Service-users (SaaS-users), service providers (SaaS-providers), which are cloud users, and cloud computing providers (see Figure 2.1). The main stream literature introduces usually three layers of cloud computing business — Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) [44]. In

this work we combine PaaS- and IaaS-providers into a single actor and call it according to [15] a cloud computing provider. A company buying such services is then PaaS- or IaaS-user or alternatively SaaS-provider. End-users are called in all contexts SaaS-users. The terminology of the cloud computing have not yet become established and our distinction might not be the unique, so that a reader should be careful with their use. We will investigate what benefits cloud computing brings for each of those business groups.

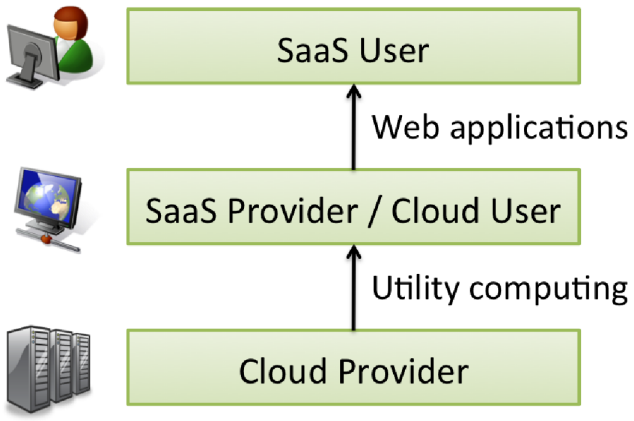

Figure 2.1: The flow of the cloud computing services [15]

Users get access to the applications and services regardless of a usage context. Internet services are changing from mostly informational to service-based applications. In cloud computing familiar services such as information retrieval, email, office applications, photo and video storage and social networking are offered ubiquitously and thus can be accessed from any device and in any place and time. From the end-user point of view cloud computing improves the service experience, as it reduces the management (no configuration or backups are needed), and eases the access as the service can be reached from any device (no matter which operation system it has, or whether it is desktop or portable device) [17].

Services used by end-users are run by companies, we call software providers (SaaS-provider) or service providers, whose software resides (usually) in an other company's cloud. Although some companies own both the datacenter and the service running on the top of it (Amazon, Google, Facebook), there is emerging a novel business trend, in which instead of purchasing a datacenter, a company actually buys a datacenter facilities for its service from an another company. Let us explain why such trend is possible. First, (especially) startup companies can outsource the IT-services to datacenter

providers. This brings the huge opportunities to company's visionary ideas that can be tested within a short time. In the old days, when a company considered about launching a new service, it had to invest to a datacenter, wait until it has been deployed (a building constructed, servers purchased and software installed) and only then see whether the idea was worth to implement. Nowadays, a company can buy any number of servers for any period of time. This means that a company can rent servers from a servers' provider, transfer its application and/or data to the servers for a paid period of time. If more computation is required the company can rent servers for longer time or alternatively rent more servers. Today, cloud computing supports total scalability, wherein renting one server for 1000 hours equals in price renting 1000 servers for one hour [15]. Second, renting servers from datacenter providers relieves cloud users of maintenance costs related to hardware. Problems with fault situations also disappear as the datacenter provider watch over them.

Now, let us take a look at datacenter providers. There are companies which construct datacenters containing hundreds, thousands or even tens of thousands servers. Renting servers from datacenters became a new profitable business with its own laws and peculiarities [60]. Below we mention some of them. Buying cloud computing utilities can be even cheaper than running workload on the own datacenter. The trick is that although the cost of cloud computing is more expensive than electricity bill of the own server, it pays nothing when not used. In other words buying a cloud computing service releases a company from the capital investment and the problems of its under-utilization. On the other hand a company providing such service can achieve a very high utilization, as it receives many requests with peaks and valley at different time moments, and thus can 'pack' workload very efficiently.

Cloud computing also benefits from economy on scale, wherein purchasing a large number of components at a time (servers, UPSs, PDUs) can cost less. This is possible because the fixed costs are then distributed over larger amount of machines. For instance James Hamilton reports in his presentation that in very large services the economy of scale brings 7.1, 5.7 and 7.1 times more savings in networking, storage and administration costs, respectively, than in small/middle-sized datacenters (numbers are collected in 2006) [37]. Server administration costs are also amortized among a bigger number of machines. An IT-specialist maintaining 10000 of servers costs less per-server price versus IT-specialist maintaining 100 servers. But of course a very high cost of an entry is required to make economy of scale substantial.

2.2 Grid

Grid computing is a term unifying the set of software and hardware technologies used in distributed computing into the paradigm, wherein computing resources are thought to be as ubiquitous and easy-to-use as electricity in the power grid. In the legendary book 'The Grid: Blueprint for a New Computing Infrastructure' the authors confirm this with the historical fact that not the invention of electricity made it so crucial and useful, but the spreading of electrical infrastructure — electrical power grids, which delivered electricity to everybody homes [41]. The same applies to the Grid, only the uniform standards of middleware bring grid computing capabilities to a mass consumer. The appealing idea behind the Grid is that instead of buying a computer, one buys the computing resources, the same way we buy electricity instead of constructing our own power plant. Concept of grid computing is very similar to cloud computing. Actually, besides tiny differences those two have a lot in common. Grid computing is aiming to the more tangible, standardized and secure services. For instance, wherein grid computing is providing storage and computing resources, cloud computing is providing services and more abstract resources. Grid computing aims to the standardization between sites to achieve universal protocols to request computing power. In grid computing security is safer but on the other hand more complicated. Instead, cloud computing resources are accessed simply via Internet. [31]

Grid and cloud computing have similar building blocks in software and hardware parts. Both aim to unify computing resources converting the underlying fabric to a pool of abstract resources. Cloud computing uses more virtualization and grid computing batch queueing systems. Thus, grid computing does not directly support interactive applications, because of expensive scheduling decisions and potentially long queues in the batch queueing systems. Both grid and cloud computing use programming interfaces such as MPI [14] and MapReduce [29]. Both also share the user's viewpoint of outsourcing the computer provisioning and maintenance to a third party. It is easy and fast to just use one's credit card to buy from a resources provider an on-demand access to thousands of servers all around the world. [31]

Are there then any reasons to choose local computing instead of grid or cloud computing? First, not everyone wants to send their private or very sensitive data to the third party. Second, some companies have very tight requirements to the availability — the service should be active even if the network is not functioning. One is ought to use grid computing whenever the jobs send to a cluster are stand-alone applications and do not have strict deadlines. Cloud

computing is appropriate to the business applications where the demand is changing fast and it is hard to predict it.

2.3 Warehouse datacenters structure

Most of nowadays computation is not running on end devices but in warehouse size datacenters. Those datacenters contain from hundreds to tens of thousands of servers. While running, servers consume electricity and produce heat, and this heat have to be removed by a cooling system requiring more electricity consumption and thus more heat. A datacenter is a matter of great investment, which are proportional to the amount of power delivered to the system. Typical construction cost is \$10-20/W [17]. Additionally, a datacenter consumes continuously electricity, which increases operational costs. That is why datacenters have to be designed very carefully. Careful datacenter design covers both the correct capital investment — appropriate for the current business situation number of servers and power efficient facilities for servers' execution support such as power system, fault control system and cooling. This introduces the totally new design solutions and methods to improve efficiency.

We briefly present the structure of a typical datacenter because a couple of solutions achieving power savings use the building blocks of a datacenter. The typical datacenter contains the following building blocks — power distribution units (PDUs), uninterrupted power supply (UPS) system, emergency diesel generators, electrical primary switchgear, individual computer cabinets and cooling infrastructure. Electricity enters from a transformer, and then it passes through a switch board to UPSs. An UPS is also connected to a diesel generator, which will be switched on in the case of power shortage. Because a generator will take 10-15 seconds to start, the UPS is needed to provide electricity during this time. UPS is basically small battery and it accomplishes AC-DC-AC conversions. The electricity coming from UPS is fed to a number of PDUs. Each PDU distributes power to 20-60 racks. Each rack is a circuit protected with its own breaker. Depending on the type of the server, each rack can contain 10-80 computing nodes (see Figure 2.2). [17]

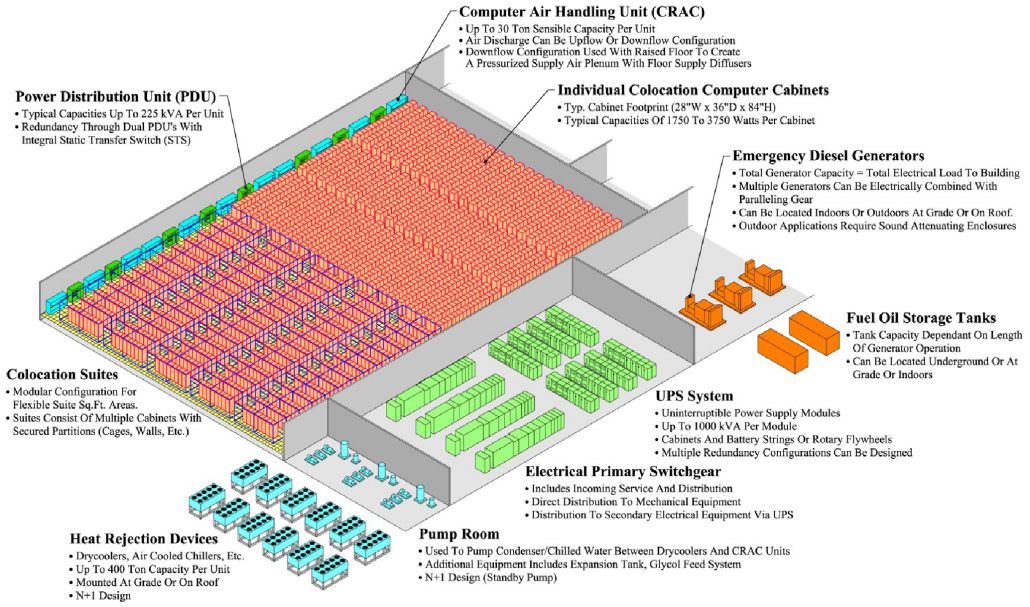

Figure 2.2: The graphical view of a typical datacenter [17]

2.4 Physics computing at CERN

CERN is a widely known nuclear particle research center. CERN was founded after Second World War in 1954 by 12 founding member states. Its original purpose was to recreate the European science and to unite European scientists. [1]

Nowadays, CERN is much more than just a research center. Within CERN objectives in addition to the primary research are: Creating and advancing the frontiers of technology, for instance World Wide Web was created at CERN by Tim Berners-Lee. Maintaining research community and bringing together nations through science. Training the scientists of tomorrow, for instance every summer there are around 200 young summer trainees visiting CERN to work and to study. [2]

Today, the most promising and breathtaking CERN project is Large Hadron Collider (LHC). LHC is the world's largest particle accelerator with the diameter 27 kilometers, situated 100 meters underground and spanning between France and Switzerland. [3] It was started up in 2008 and with its aid the researchers try to answer fundamental questions about the nature of our universe — what gives matter its mass, what is the dark matter and energy comprising 96% of the Universe, why there are more matter then antimatter

and what are the first instants of the Universe's existence. [4] [46]

The major question, which will be possibly answered with the aid of LHC, is the existence of so called Higgs boson. Higgs boson can be thought as the mathematical correction in nowadays physics to explain the presence of mass in force-carrying particles. Researchers accelerate particles to the near light speed to produce high energy collisions and thus conditions in which theoretical Higgs boson can be registered. [5] In the LHC experiment a great amount of empirical data is collected and compared to the results obtained 'theoretically' — conditions and results which are deduced and computed through physical simulations, wherein the existing physical laws are represented. If the results of empirical and theoretical approaches match, the existence of Higgs boson would be proved.

The Grid is the innovation, which is used and developed in parallel with LHC-project. The enormous amount of data produced by LHC (about 15 petabytes of data in a year) is stored and distributed through the Grid enabling physicists to work with the experiment remotely. The Grid provides also computational and communicational resources. Finland participates to the LHC-project through NorduGrid [1].

[1] http://www.nordugrid.org/about.html

Chapter 3

Overview of green IT

In the current state of IT the power expenses associated with maintenance of IT facilities are unproportionally large compared to what they could ideally be. Green IT is an emerging field aiming to decrease the wastage of power resources consumed in IT field. There are two properties we want to achieve. First, a computing device should consume as little energy as possible, let us call this power (energy) efficiency. Second, a computing device should consume the amount of energy (electricity) proportional to the useful work it does, let us call this energy-proportionality. However, both are far from the the ideal situation, computing and computing-aided components such as servers, power supply units, hard disks and coolers are not ideally power efficient nor energy-proportional in the sense that they consume more power than useful computations they accomplish. Additionally, a computer produces heat as a byproduct, which has to be removed requiring more energy. Inefficient hardware is only part of the problem, poor utilization of computational resources is also wastage of energy. Even if a server is very power efficient (do not mix up with energy-proportionality), 'doing nothing' or idling might be 'the useful computation' from a server's point view, whereas we regard it as a wastage of energy. Bringing the problem to its highest degree — even if a server is both ideally power efficient and energy-proportional (such a server do not yet exist nowadays), if there is no enough workload for it, the investment into the server were done inefficiently. In this Chapter we will look at the current state of green IT, first we give a couple of examples of green IT applied elsewhere than in datacenters just to avoid the impression that green IT and efficient datacenters are synonyms. Then we present metrics used to assess the datacenter's power efficiency generally and the metrics we will use extensively in our study. We conclude the Chapter with the challenges in front of green IT in datacenters. The concrete exam-

ples of solutions developed for improving datacenters power efficiency are left to the next chapter.

3.1 Green IT

If we are talking about office buildings, there is a research on reducing plug load. A "plug load is the energy consumed by any electronic device that is plugged into a socket." [58] In office buildings those devices are coffee makers, water coolers, refrigerators, routers, computer desktops, projectors and servers. Nowadays, many of such devices have stand by modes, in which electricity consumption is less than in active mode. Still, stand by modes consume some energy, which accumulates to a great amount over long periods of time. In [28] authors argue that the promising new technology would be occupancy sensors, which shut the gadgets totally when the users are away and do not need them for sure.

In the field of data transmission there is a research on energy efficient hardware. In [55] authors describe IEEE 802.3az Energy Efficient Ethernet (EEE) standard approved in September 2010. In this standard is used low power mode, which uses less electricity if the data transfer is also low. The standard covers the most common Ethernet physical layers such as 100BASE-TX, 1000BASE-T and 10GBASE-T. This is an important achievement because networks are very lightly utilized, 1%-5% is a typical utilization [25].

In [40] authors evaluate the energy consumption of Internet. They split Internet structure to three logical layers — access network, metro network and network core. They present the typical devices that operate in those layers such as optical line terminals in an access network, routers in a metro network and multi-shelf core routers in core network. Then, they estimate their power consumption taking into account such factors as number of homes, peak access rate and the over-subscription rate. Thus, the author concludes that the approximate power consumption of the Internet is 1% of the total electricity consumption in broadband enabled countries. This is an example of those rare studies aiming to increase the knowledge about ICT power consumption.

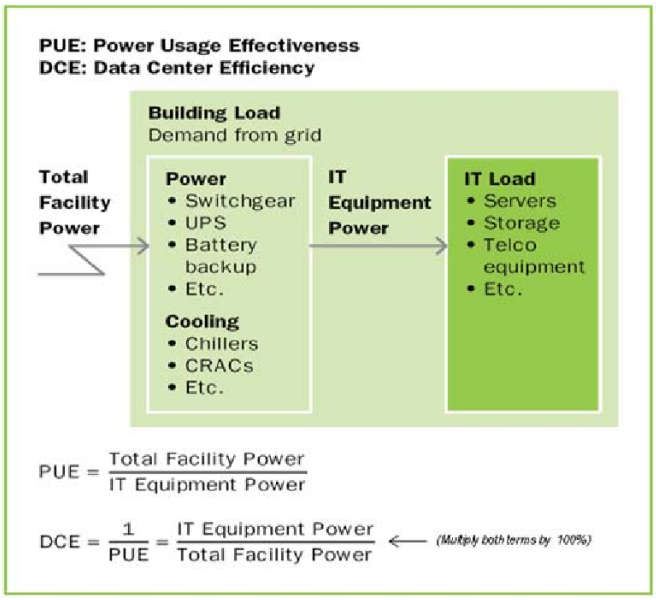

Figure 3.1: Calculation of PUE and DCiE [35]

3.2 Green IT in Cloud Computing

Energy efficiency in datacenters is defined as the useful work performed divided by the energy required for that [17]. Green IT communities have developed metrics which help designers to assess their datacenters power efficiency. Power usage efficiency (PUE) is a such widely adopted metric [7]. PUE reflects the energy efficiency of datacenter buildings in relation to IT resources. As we saw in the previous chapter, the datacenter building is more than just a collection of servers, the datacenter consists of a large infrastructure of equipment such as power supplies, UPSs, coolings, etc. PUE tells how much energy is spent in datacenter buildings' infrastructure in addition to the pure maintainance of an IT equipment (PUE = total facility power / IT equipment power) [35]. For example, the PUE value of 3 means that every watt going to an IT equipment needs 2 more watts to maintain infrastructure. Ideally, we wish the PUE value equaling 1, but according to the several studies average PUE is around 2.0-3.0 [17]. A good PUE value can be thought to be as large as 1.5 according to James Hamilton [37] and 1.6 according to The Green Grid consortium [35]. Google reported of having its PUE equal to 1.12 [6]. There is also a derivative metric of PUE — Datacenter Infrastructure Efficiency (DCiE), which is calculated as follows: DCiE = 1 / PUE. A visualization of those metrics is presented on Figure 3.1. The main

contributor of datacenters' inefficiencies are claimed to be cooling, after that comes an IT equipment itself (servers) and Uninterruptible Power Supplies (UPSs).

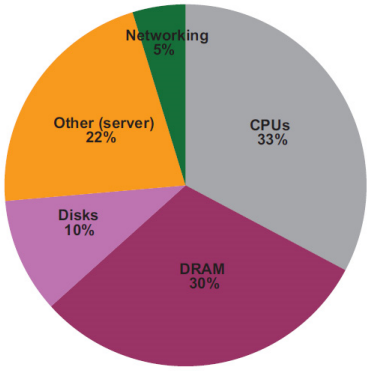

Figure 3.2: Energy consumption per a server component [17]

PUE is not alone enough when we want to improve efficiency of the IT equipment itself [34]. For instance, Figure 3.2 shows the proportion of electricity consumption each server component is responsible for. The metric server PUE (SPUE) tells how efficient the IT components which are involved into computation — CPU, memory, disks, motherboards. Normal SPUE values are around 1.6-1.8 [17]. Green IT Promotion Council suggested its own metric — Datacenter Performance Per Energy (DPPE), which function of four other metrics including PUE. DPPE captures more aspects of datacenter's energy efficiency than PUE [34].

There are a couple of industry standard organizations such as Transaction Processing Performance Council (TPC), Standard Performance Evaluation Corporation (SPEC) and Storage Performance Council (SPC) that have developed tests for measuring energy consumption of computing systems. Most of servers' developers are members of those consortia. Their test can work as a uniform measure of computer performance. Normally, they express servers' performance in terms of overall work performed per overall energy consumed, in units it can be $\frac{W*m}{T}$ (electricity consumed $[W]$ per transaction $[T]$ per time unit $[m]$) for TPC-C[1], $\frac{T}{W}$ (web transactions $[T]$ per electricity consumed $[W]$) for SPECweb2009[2] or $\frac{T}{s*W}$ (input/output operations $[T]$ per second $[s]$ and

[1]http://www.tpc.org/tpcc/
[2]http://www.spec.org/web2009/

per electricity consumed [W]) for SPC-1C/E[3]. As we can see from those example metrics, those benchmarks usually test commercial systems in terms of business transactions and differ from each other in terms of what they test [57]. For instance TPC-C is the TPC's benchmark, which is claimed to be "accepted in the industry as the most credible transaction processing benchmark" [56]. TPC-C operates in the following way, it generates a 'realistic' mix of transactions, where the proportion of each transaction is the same as in the real workload, and load the system with the workload to achieve 100%-utilization. When the system is stabilized into steady state, TPC-C measures its energy consumtion in time windows, for a minimum 120 minutes. The workload also tries to load all conponents of the server — CPU, memory, disks. TPC-C tests the whole system including clients and server enclosures (see Figure 3.3). [57] [56]

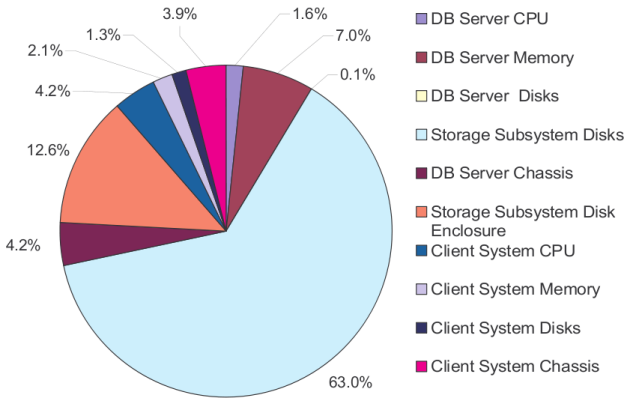

Figure 3.3: Average power consumtion of major parts used in TPC-C [56]

In our research we concentrate on the third aspect of energy efficiency measure — utilization, which we try to maximize. Utilization can be defined as a ratio of time spent on useful computation divided by the time of the whole measuring period. Naturally, it cannot be greater than one and usually it is below one because of the lack of the workload. Nowadays servers run at a very low (approximately 30%) utilization (see Figure 3.4) [17] [16]. It makes interesting for us that maximizing utilization may involve software solutions. As we will see in the next chapter, the research performed on utilization maximization have been primarily concentrated on putting server components into low-power modes during low activity periods. Such approaches need of course support from the hardware — there should be such power saving states and entering and leaving them have not to take long

[3]http://www.storageperformance.org/results/benchmark_results_spc1c

time. So, software solutions alone without hardware assistance cannot solve the under-utilization problem. Still, there are many unseen improvements on the software side that can exploit the existing hardware possibilities. Already at this point, we want to mention that due to queueing theoretical reasons, we will describe later on, in a long run the utilization cannot reach 100% (if it could, the response time would go to infinity). But of course during short time intervals we can see a system working at 100% utilization.

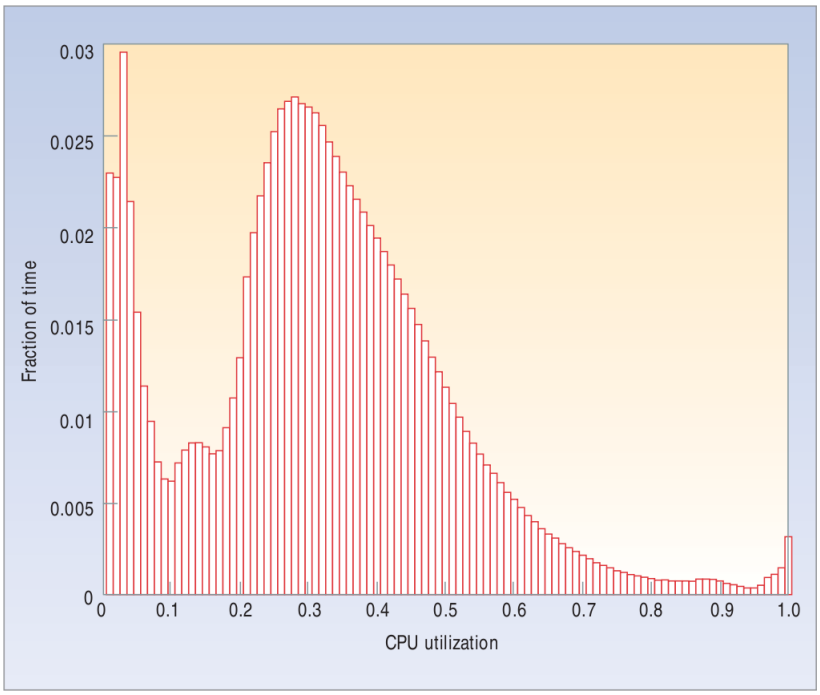

Figure 3.4: "Average CPU utilization of more than 5,000 servers during a six-month period. Servers are rarely completely idle and seldom operate near their maximum utilization, instead operating most of the time at between 10 and 50 percent of their maximum utilization levels." [16]

Here we also want to stress that the additional desire to improve utilization is driven by the poor energy-proportionality of server components, i.e. while idling, servers still consume too much energy (see Figure 3.5). Energy-proportionality could be improved either by software solutions (scheduling and buffering workload to maintain a pool of jobs) or with hardware solutions, wherein with less workload servers spend proportionally less energy (and ideally while idling they do not spend energy at all). Currently, efforts are mostly performed on software side. The reason is that to improve hardware the designer should optimize several server components in their whole

workload spectrum. As we see from the Figure 3.2 a CPU is not the only consumer of electricity.

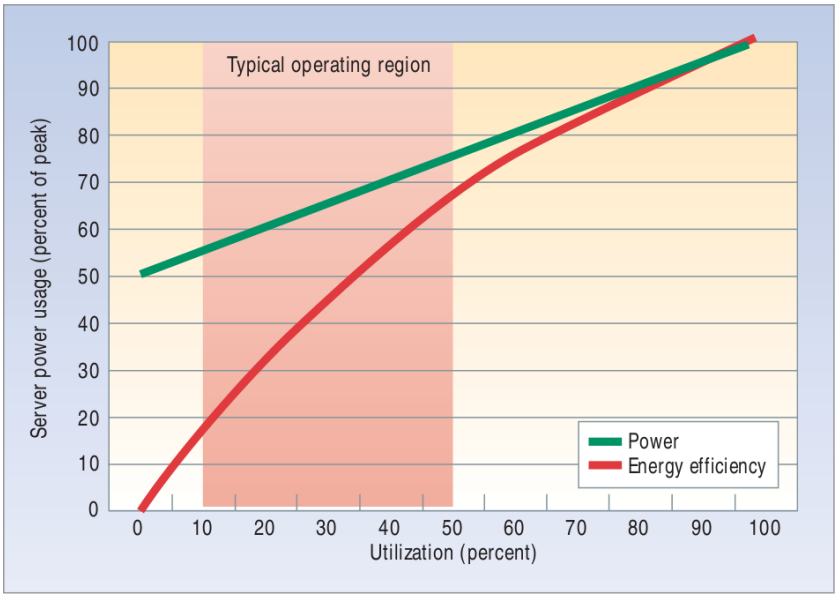

Figure 3.5: "Server power usage and energy efficiency at varying utilization levels, from idle to peak performance. Even an energy-efficient server still consumes about half its full power when doing virtually no work." [16]

3.3 Important processor related metrics

In this Section we will briefly discuss important energy consumption metrics, whose nature and characteristics explain how to achieve better utilization by balancing between less energy consumption and increased response time. Very first it is important to understand the difference between concepts power and energy, which are often wrongly thought to be interchangeable. Energy is the metric which determines the final bill and usually expressed in *(kilo)watt-hours*. Energy is ability to do work. The power of an electrical device is the rate at which it consumes energy and it is expressed in *watts*. Energy is what and additionally how much one delivers and power is the rate at which one delivers it.

The first metric is processor frequency, which defines how efficient the processor is. Processor frequency is expressed in number of cycles performed in

a time period [38]. The frequency is very closely related to power. The frequency and voltage scaling techniques such as Intel's SpeedStep and AMD's Cool'n'Quiet use this relation to decrease power at the expense of reduced frequency. The equation binding those metrics is the following $P = C*V^2*F$, where C is the capacitance, V is the voltage and F is the frequency [27].

The second metric is power efficiency with unit *jobs/watt-hour*. It defines the rate at which a computer processes an amount of fixed workload (here we assume that all jobs have the same size) for every watt-hour of energy used. Power efficiency is calculated in the following way: the workload delivered (expressed in a number of jobs) is divided by the energy consumed, and the bigger this ratio is the more power efficient computation we get. Earlier studies [54] have shown that power efficiency increases when several jobs are run simultaneously on a single core. This way they are run in parallel, CPU and memory switching between a number of different processes. In [54] the authors argue that this effect is explained by increased utilization, i.e. when we have enough jobs running in parallel on a single core, we achieve a better control to interchange between jobs and fill the process idling gaps. As computer electricity consumption during idling is almost as big as during working phase, it is desirable that a computer remains always fully-utilized. In Figure 3.6 you can see one possible scenario how processes could be run simultaneously instead of sequentially to achieve better power consumption.

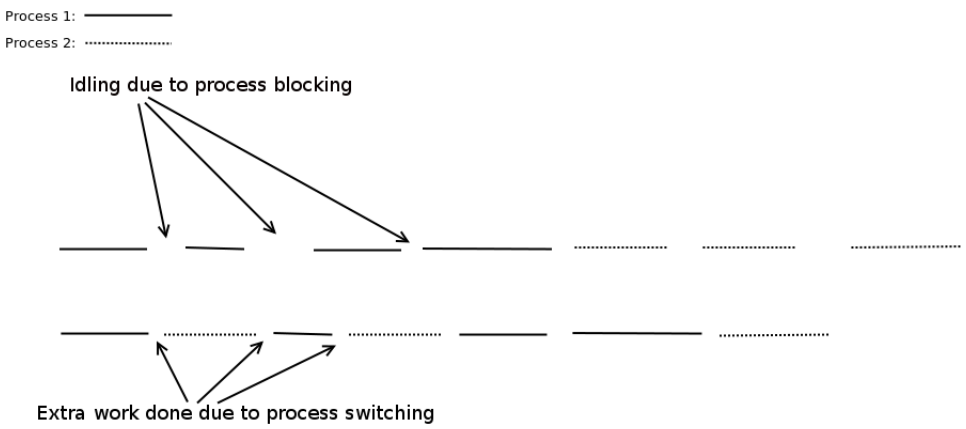

Figure 3.6: The processes consuming CPU when run in serial one after another and simultaneously

Now we demonstrate the use of those metrics by examples. Their purpose is to motivate the reader towards our solution (see Section 7.2). The examples

explain why in our solution we aim to get a few computing nodes fully-utilized and other off instead of all nodes running at moderate utilization. Assume we have a computer running at 1 watt power level and it should compute 10 similar jobs. To accomplish that it takes 10 hours. What is the energy consumed and how efficient is this solution? As the computer ran at 1 watt during 10 hours, we get

$$1 \text{ watt} * 10 \text{ hours} = 10 \text{ watt-hours}$$
$$10 \text{ jobs } 10 \text{ watt-hours} = 1 \text{ jobs/watt-hour}$$

Now imagine that we use two computers similar to the previous to accomplish the same task. How much more efficient/inefficient is this solution to the previous one? Both computers run at 1 watt power level but because the workload is divided equally between them, each computer will get 5 jobs, so it will take 5 hours to accomplish all jobs. We get for the energy spent:

$$1 \text{ watt} * 5 \text{ hours} + 1 \text{ watt} * 5 \text{ hours} = 10 \text{ watt-hours}$$

and for the energy efficiency:

$$10 \text{ jobs} / 10 \text{ watts-hour} = 1 \text{ jobs/watt-hour}$$

So the solution is as power-efficient as the first one. A few points to remark are that the computation lasted only a half of the first one. This is of course a positive outcome, but during the computation we needed at the same time twice as much power as in the first example. This could cause problems if we have only a limited amount of power. This is not a totally artificial situation as in the field of power generation it is extremely desirable not to produce power peaks.

Now let us imagine a situation, where computers are not fully utilized. Suppose we have three computers to run 6 jobs (2 jobs per computer). As before each computer consumes 1 watt of electricity, but now it takes 3 hours to run all the jobs for each computer. The energy consumed is now:

$$(1 \text{ watt} * 3 \text{ hours}) + (1 \text{ watt} * 3 \text{ hours}) + (1 \text{ watt} * 3 \text{ hours}) = 9 \text{ watt-hours}$$

and energy efficiency is:

$$6 \text{ jobs} / 9 \text{ watt-hours} = 0.667 \text{ jobs/watt-hour}$$

which is less than in above cases. Now the reader might wonder, how it is possible that a computer executes jobs slower when there are less jobs. In the first example we had 10 jobs / 10 hours = 1 jobs/watt-hour and in the last 2 jobs / 3 hours = 0.667 jobs/watt-hour. The trick is that in the first example a computer executes all jobs simultaneously (in parallel) and thus the spent time is spread among all jobs, although that one job completion takes still fewer time than it would take when executed alone. The previous study [54] implies that this is due to poor utilization, in which case a job might block waiting for a memory access and thus a computer spend much of its time idling but still spending energy. In contrast, when fully utilized, while some job is blocked, another could be executed and thus consume energy that otherwise would be wasted. This leads us to a very important observation—to optimize energy efficiency we should provide computers with enough workload.

3.4 Challenges

We can think there are two main classes of workload sent to datacenter servers — on-line requests and off-line batch jobs. On-line workload (such as web page surfing, on-line translation, video watching and so on) is interactive and thus requires response from the server to the users' requests. Such workload exhibits a high sensitivity to the response time — if it exceeds the limit, the users start to suffer from degraded service. It can have disastrous consequences if the service provider has agreed on the *service level agreement* (SLA) (the synonym of this term is quality of service, which we will use in queueing theory), which is often the case. When service level is violated, the provider loses revenue as some of the customers cease to use its service. The on-line workload is also problematic because of the hardness to achieve perfect utilization. Although the on-line workload is highly interactive, it has a number of idle periods, for instance those time intervals when the server is waiting for the user's response. It is hard to fill such intervals because they are quite short and cannot be extended as quality of service may suffer.

Off-line batch jobs are a type of workload requiring large resources — CPU, memory, disk space and they usually do not have strict time constrains. For datacenter computation optimization this brings great opportunities — by using different scheduling approaches we can improve the energy-proportionality just slightly affecting performance. We will see in the next Chapter 4 how it could be done and our approach is an example of this type of energy improvement.

Another important aspect related to optimizing datacenter computation is workload prediction. It is well known that workload intensity is varying. But not only workload, business requirements, electricity prices and datacenter capacity are seldom constant values.

When we start to explain related solutions and our solution, it is worth to keep in mind these challenges. These challenges explain why the solutions are such as they are. For instance, one cannot forget users' perspective while designing energy efficient solutions.

Chapter 4

Related work

One of the main obstacles in front of efficient energy management is a varying workload. There have been different strategies for improving energy management with the commonality that they introduce trade-offs, for instance achieving better power efficiency at the expense of degraded performance. We can divide those techniques into two categories — those affecting performance and those affecting users' behavior. The former has been studied more than the latter. In researches it has been proposed to deny the service during rush hours (TCP protocol's scale window) [48], use execution throttling and frequency scaling techniques, and spreading the workload over time and space. The latter are slowly coming into use in the datacenter community, although the users' behavior shaping has been used in other fields. For instance in electrical power companies it is quite common that the company charges less for energy during less active periods (i.e. nights) to make the workload as constant as possible. We will look in this Chapter into few examples of the first type of workload management — reducing performance to achieve better energy efficiency.

In [50] the authors claim that current datacenter servers run on average with 20-30% utilization. Still, such poorly utilized servers consume about 60% of the peak power. The paper comes up with the solution called PowerNap suggesting turning off unused machines to save energy and resulting in 23% energy savings. The advantage of this approach is that hardware designers can concentrate on optimizing the energy efficiency of the only two states in a computer — fully utilized active and near zero-power passive. There are other approaches in which one or several computer components are set into a power saving state. But then it is desirable that the components functioning in such states are not less efficient than they are in active states, which

imposes serious challenges for hardware designers. According to the authors the disadvantage of this approach is that despite of idling more than 60% of the time the typical workload of some interactive server applications exhibits idle periods of less than one second, which are too short to be exploited by the PowerNap mechanism. The other problem resides in an inefficiency of datacenter power supply units (PSUs), whose efficiency degrades as output voltage decreases. To overcome this, the authors introduce a new power provisioning approach — Redundant Array for Inexpensive Load Sharing (RAILS).

Although, the idea of PowerNap is elegant and simple, it has a limitation. In their workload analysis of datacenter servers they do not distinguish between workload of separate machines. This leads to the situation in which the decisions are done based on the knowledge of averages, which does not lead to perfectly efficient solutions. For instance they present a table with data collected from seven departmental IT servers. According to their analysis, an idle period is on average 1.8 seconds long in a 600-node cluster. Although, the length of idle periods can vary largely on different machines of the same cluster.

In [30] similarly as in [50] the authors found that in a big cluster of thousands of servers there is a 7-16% gap between achieved peak power and theoretical peak power reported by a manufacturer (so called nameplate value). The peak grows to almost 40% in the whole datacenter. They found out that computers' nameplate value is higher than a server can actually achieve in its real use. They also found that, as the number of servers grows, it becomes more unlikely that different machines have their peak power at the same moment due to statistical effects. They present cumulative distribution function curves of rack's, PDU's and cluster's utilization level to visualize the effect of under-utilization. For example a cluster runs at most at 0.923 utilization (normalized to the peak value), on the other hand the rack can achieve 0.96 utilization. The under-utilization of the datacenter facility is transferred into total cost of the datacenter acquisition. As a solution, the authors suggest hosting additional machines under the existing power budget. They also suggest using a more heterogeneous workload to smooth workload activity peaks. This is because a heterogeneous workload more likely does not have correlated activity peaks. The authors suggest the use of CPU voltage and frequency scaling (DVS) as a technique to preserve the cluster from the exceeding an allowable cluster power limit. If the current workload will threaten to grow over the existing power supply limits, the machines could be slowed down until the workload decreases again.

Paper [30] presents a pair of simple methods to improve PSU usage efficiency. Here is important to stress that those methods are aiming to PSU full utilization. By adding more servers and keeping the PSUs capacity unchanged, we can process more workload, but workload variation still remains. Thus, we still face the problem of an inefficient power usage during less active periods. The usage of DVS in case of too tall activity peaks is a useful idea, but as the authors said no analyses were made on how it can affect the system performance.

In [51] the authors describe several techniques to switch server hardware components into active low-power modes during the low workload. They argue that PowerNap [50] is not applicable to the workload class called On-line Data-Intensive (OLDI). OLDI is a very interactive workload as it requires responsiveness in the sub-second time scale. Large search products, on-line advertising and machine translation are examples of OLDI. Although, the OLDI workload there could exhibit periods of low activity, it is very rarely fully idle. Thus, instead of turning machines off, their components performance can be reduced to achieve power-performance trade-offs. The authors present activity graphs, whose purpose is to succinctly describe how different workload activity causes the utilization level changes in different hardware elements such as CPU, memory and disks. The authors also present the characteristics of components idle modes such as the duration of transition to the mode and how much energy it saves related to the full active mode. By combining those two pieces of information, the authors state, which low-power mode can be applied in which workload. Their ultimate goal is to create a framework for energy-proportionality, wherein the server component spends as much energy as there is useful computation available.

Although, in [51] the very large spectrum of idle and low-power modes is presented as well as their applicability to different workload levels, their approach suffers from the criticism presented in [50] — it is much harder to optimize several server components to operate efficiently at all activity levels than to optimize just two — fully-active mode and fully-passive mode. Some of their results are not correct in all contexts. For instance they demonstrate how to read activity graphs — when the CPU utilization is 80% then CPU can be switched into active low-power mode that incurs a 25% slowdown ($1/1.25 = 80\%$). This is not always true, because low utilization is not necessarily a result of the lack of computation. For instance the process can wait for a constant time I/O-event and thus the CPU slowdown will not increase the utilization (but it is true that it might decrease energy consumption during the waiting time).

In [33] the authors propose to exploit the datacenter uninterrupted power supplies (UPSs) during the increased workload to 'shave' power peaks. The authors remind that minimizing energy consumed in a datacenter is not necessary the same as minimizing the electricity bill. They argue that most electricity providers charge a greater price for intensity peaks, which forces us to concentrate on them. Their insight is that in datacenters there is an unused power source — UPS, whose primary task is to guarantee uninterrupted power supplement during electricity breaks. The authors formed a framework for power peak reduction. Their algorithm knows the day's workload in advance and then with the help of UPSs and voltage scaling techniques tries to smooth the peaks starting with the tallest one. The authors report that using the algorithm can reduce 15-45% of peak power which translates into 6-18% savings in an electricity bill.

Although, the idea in [33] is very fresh, it suffers from several limitations. First, the shortages reported by authors themselves — lower UPS lifetime due to frequent charge/discharge cycles and reduced datacenter availability due to discharging UPSs undermines their usefulness in power break situations. Second, the algorithm makes too severe assumption without a justification — knowing the workload for a day in advance.

In Figure 4.1 we summarize the ideas presented in the first three papers and compare them to our solution, which was briefly described in Introduction and in details in Section 7.2. We sketched the original workload with a red color and assume that it correlates with electricity consumption. Then, from the same graph with blue, we can see what happened to electricity consumption after applying one of the earlier presented techniques to the original workload (marked with red).

In [45] provisioning of datacenter servers for power savings is modeled as an optimization problem. The authors present a general formula which captures the expenditure of energy consumption, revenue lost due to an increased delay and costs related to switching servers in and out of a power saving mode. The generality of the formula provides wide possibilities to apply any model appropriate for the current situation. For instance, the choice of revenue losses can be expressed by any equation, as well the delay can be modeled either with the queueing theory average delay or the 99%-bound delay. The optimization procedure is then deduced from this formula. The basic idea is to achieve a trade-off between energy costs and revenue losses due to degraded quality of service. The new algorithm — Lazy Capacity Provisioning (LCP(w)) operates by predicting the incoming traffic for the next time period w, and based on that outputs the optimal number of servers able to process

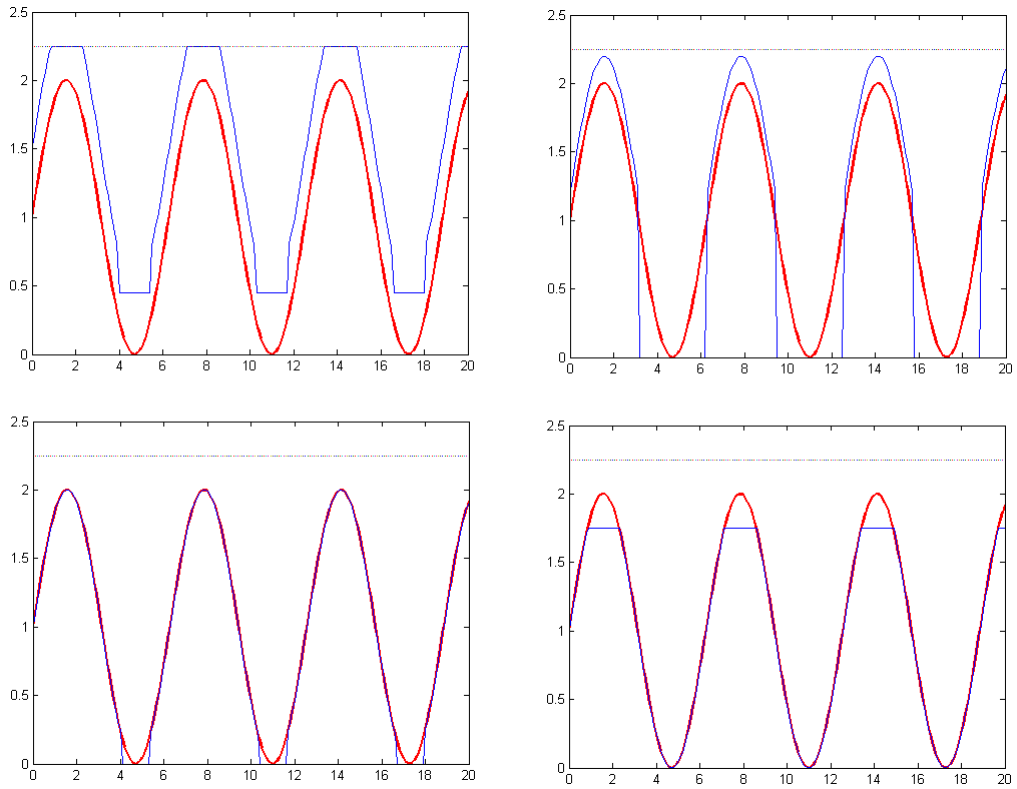

Figure 4.1: How techniques affect workload (from left to right, from top to bottom): Google approach [30], our approach described earlier in the introduction, PoweNap [50] and approach using UPSs [33]

the workload. Although, for the optimal solution the whole future should be known in advance, the 'myopic' algorithm LCP(0) is still 3-competitive, which means that it is less than 3 times as efficient as the optimal algorithm for all problem instances (all convex cost functions and finite incoming traffic rate). The authors state that as large as 50% energy savings and 40% cost savings can be achieved with the algorithm if the workload peak-to-mean ratio (PMR) equals 5. The authors also consider the impact of datacenter related aspects on the efficiency of the algorithm. They present how PMR, the energy cost function, delay cost function and switching costs affect the savings gained with the algorithm. To our knowledge and opinion [45] gives the most wide analysis of such aspects and their solution is provides significant savings.

In [24] the technique named Dynamic Voltage and Frequency Scaling based

on Workload Decomposition (DVFS-WD) is introduced. Energy savings are achieved through the use of program workload analysis and the use of DVFS in the light of the obtained information about the current workload. The program workload is decomposed into on-chip and off-chip based on statistics reported by a performance monitoring unit (PMU). The insight of the approach is that during the memory accesses the CPU is idling — consuming energy but performing no useful work. In [24] the authors switch the processor with DVFS into a lower frequency mode. In the lower frequency mode CPU consumes less energy at the expense of degraded efficiency, but efficiency is not crucial then because it is reduced during the execution off-chip periods. The authors report that energy savings of 20-40% are achieved with 10-30% performance loss for CPU-bound applications. The authors also state that because the latency for switching the CPU voltage/frequency is 500 μseconds the scaling should happen at least in two or three orders of magnitude larger periods of time. The idea of [24] is very close to [51], DVFS-WD is just one technique to slow down the processor during less active workload periods.

In [32] authors consider how to distribute available power among datacenter servers so as to minimize mean response time. The authors argue that datacenters usually have fixed peak power budget and interesting question what is the optimal allocation of power to servers. For instance the available power can be distributed in the way that as many servers are running as possible at their minimum speed (frequency) or in the other way a few servers are running at their maximum speed. The third possibility is to run intermediate amount of servers at some speed between maximum and minimum. Generally, authors state that it is not always optimal to run servers at their maximum power levels as it was introduced in some research [16]. With the use of voltage and frequency scaling mechanisms such as DFS, DVFS and DVFS+DFS the server can consume less power at the expense of its performance. The authors deduce the power-to-frequency curves for those techniques. Then they present a formula which relates the used power of the server to its performance. The sum of running servers' power must of course equal to the fixed power budget and the total performance of all servers is matter of maximization. The answers to the equations to be maximized are derived. It came out that the optimal allocation depends on such factors as power-to-frequency relationship in the processors, the arrival rate of jobs, the maximum server frequency. The study experiments show that the use of the optimal power allocation can reduce mean response time even as much as a factor of 5.

Chapter 5

Theoretical background

5.1 Queueing theory

In this Section we will define basic concepts we will need later on. We start from the definition of *stochastic process* and later, as stochastic process is a very broad concept, introduce its subclasses, which fulfill our needs. The concept of Markov chains is needed to understand how the equations of queueing systems are derived. The general idea of queueing theory modeling is presented in Section 5.1.6. If the reader is already familiar with queueing theory basics, we urge to go directly to Sections 5.1.9 and 5.1.10, which give a basis for our solution. The materials used in this Chapter are taken from queueing theory textbooks [19] and [36]. We also used a material presented on 'Introduction to Performance Analysis'-course of Department of Communications and Networking [1].

5.1.1 Renewal sequence

In probability theory, a stochastic process is a sequence of random variables, each of which is indexed with numbers from a set N and takes values in a common range from a set M. Varying the characteristics of the index set and the random variable range set, we get four types of stochastic processes — discrete time, discrete space; continuous time, discrete space; discrete time, discrete space and time ordering, continuous space. In our study we will use discrete ordering, discrete space and discrete ordering, continuous space. With these we can represent for instance the number of jobs in a queue at the

[1] https://noppa.aalto.fi/noppa/kurssi/s-38.1146/luennot

specified time moment or the interarrival time between successive customers. Also, we assume that random variables in the sequence are independent and identically distributed (i.i.d.). When such a property is true, the process is called *renewal process*. Formally:

$$\{X_n\}_{n=1}^{\infty}$$

$$F(x) = P(X_n \leq x),\ n = 1, 2, ...$$

We can think about the sequence $\{X_n\}_{n=1}^{\infty}$ that it is an ordered set of random variables, each of which follows the same probability distribution and independent of others, also in a realization — when each random variable takes a value, we get an ordered set of values. Depending on the probability distribution of X_i:s, we can get different types of processes. In our study we will need the following distributions: deterministic (D) and exponential (Poisson process, M).

5.1.2 Deterministic process

Deterministic distribution can be defined as follows:

$$\begin{aligned} F(x) &= 0,\ x \neq b, \\ F(x) &= 1,\ x = b. \end{aligned} \tag{5.1}$$

Deterministic distributions are used when for example service or waiting time is known to be a constant. However, it is seldom the case, because of small fluctuations caused by other uncontrollable factors. So, a better approach might be the use of other distributions.

As we explained in Section 5.1.1, the random process gets its realization according to its probability distribution. Now, we can think that for example a server service time is deterministically distributed, thus the sequence $\{X_n\}_{n=1}^{\infty}$ can get the realization $5, 5, 5, 5, ...$ ($b = 5$ in (5.1)).

5.1.3 Poisson process

Poisson process can be defined in three interchangeable ways:

1. Poisson process is a pure birth process. In an infinitely short time period only one event can occur. This happens with the probability λdt and independently of previous events, or formally:

$$A.\ \text{The process has independent increments}$$

$$B.\ \text{The process has homogeneous increments}$$

$$C.\ Pr[N(t+dt) - N(t)] \begin{cases} = 0] = 1 - \lambda dt \\ = 1] = \lambda dt \\ > 1] = 0 \end{cases}$$

   where $N(t)$ is the counting process — the function representing the cumulative number of events happened thus far, λ is the arrival rate and dt is a differential sized time interval.

2. The number of arrivals in a finite interval t follows the Poisson distribution: $P\{N(t) = n\} = \frac{(\lambda * t)^n}{n!} * e^{-\lambda * t}$. In addition, the number of arrivals in non-overlapping intervals is independent of each other.

3. The interarrival times are independent and follow the exponential distribution: $P\{interarrivaltime < t\} = 1 - e^{-\lambda t}$.

The very important property of a Poisson process is that Poisson Arrivals See Time Averages (PASTA). This property means that the distribution of the system at a randomly chosen time moments will be the same as the system state distribution the Poisson arrivals observes. In other words, if we make infinitely many observations of system state at random time moments and there are infinitely many Poisson arrivals, which find the system in one of its states, those two ways will result the same distribution.

Another important property is the random split property. It says that if a Poisson process with the intensity λ is split into two processes with probabilities p_1 and p_2, such that $p_1 + p_2 = 1$ then the resulting processes are also Poisson processes with intensities $p_1\lambda$ and $p_2\lambda$. This result can be generalized to a bigger number of processes.

5.1.4 Markov chains

Markov chain is a mathematical model, which has a number of applications in machine learning, economics, biology, etc. Markov chains are well studied and lead to nice and simple equations, which express the nature of the underlying system. We will need Markov chains to model the behavior of

servers and especially server queues. First, we will look at discrete time Markov chains, which are based on the concept of state transition probabilities. Then, we will look at continuous time Markov chains, which are based on the concept of transition rates. In our study we will need continuous time Markov chains, but we will present also discrete time Markov chains, because the derivation of transition rates is easier through the transition probabilities.

Formally, a Markov chain (with a discrete state space) is defined as follows [19]. If there is a random process $\{X(t),\ t \in T\}$ that satisfies the following properties:

$$P\{X(t_{n+1}) = x_{n+1} \mid X(t_1) = x_1, ..., X(t_n) = x_n\} = \\ = P\{X(t_{n+1}) = x_{n+1} \mid X(t_n) = x_n\}$$

then the process is called Markov chain. This condition says that the probability to transfer to the state x_{n+1} in the next time interval t_{n+1} depends only on the current state x_n and not on the whole history. We can think that the random process realization of x_n (the value random variable X takes at the time moment n) defines which probability distribution x_{n+1} will follow.

Markov chains are often depicted as directed graphs with nodes as states, arcs between states as transitions and weights on the arcs as transition probabilities. Mathematical way representing Markov chains is a matrix, where the rows and columns represent states. The matrix entry p_{ij} tells the probability to transfer from state i to state j, that it:

$$p_{ij} = P\{X_{n+1} = j \mid X_n = i\}.$$

We will call such p_{ij} as a state transition probability.

Another important property of the Markov chain is time-homogeneity. Markov chain $X(t)$ is time-homogeneous if

$$P\{X(t+h) = y \mid X(t) = x\} = P\{X(h) = y \mid X(0) = x\} \\ \forall t, h \geq 0 \ \wedge \ x, y \in S.$$

In other words the probability to transfer into the next state depends only on the difference of time moments in successive states and not on the absolute

time. The other way to describe it is to say that the state probabilities remain the same over time.

If we use Markov chain with transition probabilities, we call it discrete time Markov chain. Now we will define state transition rates q_{ij}. If the transition rates are used instead of transition probabilities, we are dealing with a continuous time Markov chain.

$$q_{ij} := \lim_{h \to 0} 1/h * p_{ij}(h) = \lim_{h \to 0} 1/h * P\{X(h) = j \mid X(0) = i\}$$

If the Markov process is in the state i, the conditional probability that it transfers to the state j during a short time interval h is $q_{ij}h + o(h)$. Note, that the probability does not depend on the other time intervals. Let q_i denote the total transition rate out of the state i:

$$q_i := \sum_{j \neq i} q_{ij}$$

In other words, the conditional probability the Markov process will transfer from state i to any other state adjacent to it is given by $q_i h + o(h)$. As the transition is independent of the other time intervals, the condition satisfies the memoryless property. Thus, the time the process stays in the state i before transferring anywhere else is exponentially distributed with the intensity q_i (an exponential probability density function $f(x; q_i)$ equals $q_i e^{-q_i x}$ if $x \geq 0$, and it equals 0 otherwise):

$$T_i \sim Exp(q_i)$$

Similarly, the time the process stays in the state i before transferring to the state j is exponentially distributed with the intensity q_{ij}:

$$T_{ij} \sim Exp(q_{ij})$$

Let us think about those in the light of what we told in Section 5.1.3. If the service times are exponentially distributed and thus the sequence of the service times forms Poisson process (see definition 3 in Section 5.1.3), the Markov chain can be used to model the server.

A state transition probability is then defined with state transition rates:

$$p_{ij} = P\{T_j = T_{ij}\} = q_{ij}/q_i$$

In addition to state transition rates and probabilities, it is useful to know the probabilities of the states, i.e. the probability with which the process visits the state at the random time moment — $\pi = (\pi_i \mid \pi_i \geq 0, i \in S)$. We call the set of such probabilities as an equilibrium distribution, which tells us what is the probability to reside in the specific state. We can calculate the equilibrium distribution from the state transition rates and so called normalization condition. The normalization condition says that the sum of state probabilities must sum up to one (as the typical distribution does):

$$\sum_{i \in S} \pi_i = 1 \tag{5.2}$$

The global balance equations set the dependencies between the state in and out transitions. For every state it must hold that outgoing flow sum up to the sum of the ingoing flow:

$$\sum_{j \neq i} \pi_i q_{ij} = \sum_{j \neq i} \pi_j q_{ji} \tag{5.3}$$

If normalization condition and global balance equations are satisfied, we achieve the equilibrium distribution of the Markov process. The unique equilibrium distribution always exists if the Markov chain is finite — it has the finite number of states and from each state there is a path to all other states. Markov chains we use in our study usually have a unique equilibrium distribution.

5.1.5 Birth-death process

Here is presented a class of the Markov chains we will use extensively throughout the study. The Markov chain which has an infinite (or finite) number of states, whose states can be arranged in a sequence and whose transitions are possible only between the adjacent states is called *birth-death process*. The idea of the birth-death process is better understood through an example. Let

us assume, we have some service, for instance a shop's cash desk. The cash desk serves customers at the speed of 3 customers in an hour, and customers arrive to it with the speed of 2 customers in an hour. If we want to model the cash desk as a Markov chain, we can represent the number of customers in the service as states — if there is no customers, we are in the state 0, if there is one, we are in the state 1, if there are two (one is served by the seller and another is waiting for the service), we are in the state 2 and so on. At a single time moment only one customer could be served (and thus leave the service) or ask for the service (thus being served or joining the queue if it was not empty). The adjacent states are thus connected with arcs weighted by birth and death rates — λ_i and μ_i respectively. The birth rate in our case is the speed at which new customers arrive and thus 2, the death rate is the speed at which they are served and thus leave the service, which is 3. Finally, we get a Markov chain, which looks like in Figure 5.1.

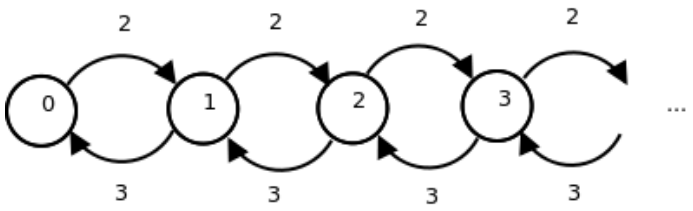

Figure 5.1: Example of a birth-death process, customers arrive at the rate 2 and are served at the rate 3, numbers on the states signifies the number of waiting customers (customers in the queue).

Birth-death processes are important because first we can model our systems with them, and second, their theory — equilibrium distribution and others are very easy to calculate. When we set the condition (5.2) and the conditions (5.3) for every state and solve them, we obtain:

$$\pi_i = \pi_0 \prod_{j=1}^{i} \lambda_{j-1}/\mu_j \tag{5.4}$$

$$\pi_0 = (1 + \sum_{i=1}^{\infty} \prod_{j=1}^{i} \lambda_{j-1}/\mu_j)^{-1} \tag{5.5}$$

5.1.6 Queueing systems

Now we use the theory developed thus far to model data communications. We can think that there are infinite or a very large number of customers (if there are only a few customers, their Poisson arrival property might be violated (see Section 5.1.3)) and the system as shown on the Figure 5.2. The system provides a service to customers, which request the service by arriving at the random time moments to the system. The duration of the service is not known in advance. Although customers' arrivals and their service time are random, their probability distributions are known, and based on that, we can draw general conclusions about the system behavior in a long run. Thus, we assume that we know the probability distribution of the time t_a to the next customer arrival and their arrival rate λ. Also, we assume we know the probability distribution of the service time t_s and similarly its rate μ.

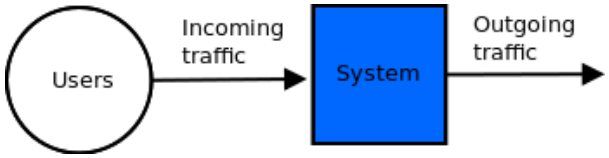

Figure 5.2: The setting of the data communication system: customer arrive to the system, processed by the system and leave the system

When we express the system and the customers in the mathematical terms using queueing theory, we obtain the relationships that bind together properties of the service, the system and the traffic. Generally, we can freeze two aspects to obtain the third one, i.e. when we set the values of two the values of the third is derivable from those. Thus we can quantitatively determine how, for instance, the quality of service (service) will react to an increase of customers arrival rate (traffic) when the system capacity (system) remains the same. We can get answers to the questions like "Given the system and the traffic, what is the quality of service experienced by the customers?", "Given the traffic and the quality of service, what should be the capacity of the system?" and "Given the system, with a predefined capacity and required quality of service, what is the maximum possible traffic?". Note that quality of service can be defined in many different ways and can reflect either customers' or system's point of view, it can be the delay experienced by a customer, packet loss rate, utilization, etc.

The system can be modeled in many different ways. The choice of the model depends on the nature of real system. For instance, the way the system be-

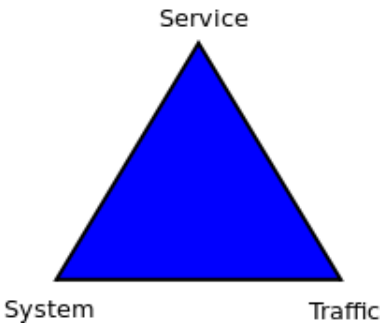

Figure 5.3: The quality of service triangle: by freezing two aspects makes the third derivable of the first two

haves when too many customers are arrived to it (a newly arrived customer might be lost or a newly arrived customer will be forced to wait or alternatively a newly arrived customer is accepted to the service but the service rate will degrade) determines which model would be a more faithful representation of reality. We will need in our study a so called pure queueing system model, in which there are a finite number of servers ($n < \infty$), which provide a service to customers, and an infinite number of waiting places ($m = \infty$). If all servers are occupied when a customer arrives, it goes to the waiting place and waits until some server will be released.

When all system servers are occupied and there are multiple customers in the queue, it is interesting which customer will be taken into service next. This is called the scheduling policy. There are many different policies each of which affects the properties of the system such as average waiting time, average response time, etc. We will use only two of them — First In First Out (FIFO) and Process Sharing (PS), but a reader must understand that those are not the only choices. In FIFO the customer arrived the earliest will be served the first. In PS all customers in a queue are served simultaneously, this can be thought either that each customer is given a time slice during which it performs some computation at the server's full speed and then goes back to the queue or that the server capacity is shared between all customers in the queue such that each customer is served at a degraded speed.

There is an agreed notation, called Kendall notation, for various customers' arrival distributions, service duration distributions, number of servers and scheduling policies. For arrival and service times we assume that they are identically and independently distributed (i.i.d.). M denotes the exponential (memoryless) distribution, D denotes deterministic times and G denotes general (any possible) distribution. The Kendall notation is of the form

$A/B/n/p/k - P$, where A refers to the arrival process, B refers to service times, n refers to the number of parallel servers, p refers to the number of waiting places, k refers to the size of the customer population (p and k are usually infinite — $p = \infty,\ k = \infty$ and thus omitted) and P refers to the scheduling policy. For example, $M/G/1 - FIFO$ stands for the configuration in which customers arrive according to the Poisson process, service times follow some general (possible unknown) distribution, there is one service place in the system and waiting customers are taken into system according to the FIFO-policy.

5.1.7 Little's formula

Before we start to describe properties of different systems, let us introduce the very important formula which plays a crucial role in deducing general results about queueing theory systems. When we construct a system, we usually make a lot of simplifying assumptions about it — the arrival or service process is Poisson, etc. When those assumptions do not hold, the resulting formula for the system behavior become either too intractable or even impossible to deduce. Still, there is a general formula, which is true for all stable systems.

If the system is stable i.e. it becomes empty at eventual time intervals — arrival rate equals departing rate, we can derive *Little's formula*:

$$\overline{N} = \lambda * \overline{T} \tag{5.6}$$

where $\overline{N}$ is the average number of the customers in the system and $\overline{T}$ is the average time a customer spends in the system. As the total waiting time in the system consists of waiting time in the queue $\overline{T_Q}$ and in the service $\overline{T_S}$ — $\overline{T} = \overline{T_Q} + \overline{T_S}$ (the same is true for the number of customers — $\overline{N} = \overline{N_Q} + \overline{N_S}$), Little's formula has also forms for queue and service waiting times:

$$\overline{N_Q} = \lambda * \overline{T_Q}$$
$$\overline{N_S} = \lambda * \overline{T_S}$$

We want to stress here again that Little's formula is true for all system types and it is used in derivition of many results.

5.1.8 $M/M/1$ queue

Now, we will look at the properties of $M/M/1 - FIFO$-queue more closely. As we described in Section 5.1.6, $M/M/1 - FIFO$-queue is the system, wherein customers arrive according to Poisson process with the rate λ (or in other words, interarrival times follow exponential distribution), service times follow exponential distribution with the mean $\frac{1}{\mu}$, and there is one server processing customers in first in first out manner. Let $X(t)$ denote the number of customers in the system at time moment t. During a short time interval $(t, t+h]$ a new customer could arrive with a probability $\lambda h + o(h)$ (with a state transition $i \to i+1$) and a customer leaves the system with a probability $\mu h + o(h)$ (with a state transition $i \to i-1$), where $o(h)$ is any function for which holds $\lim_{h\to 0} o(h)/h = 0$. Thus $X(t)$ forms a Markov chain and $X(t)$ is also a birth-death process with an infinite state space $S = \{0, 1, 2, ...\}$ (see Section 5.1.5).

By forming global balance equations and a normalizing condition described in Section 5.1.4, we can derive the formula for state probabilities (equilibrium distribution). If we substitute $\frac{\lambda}{\mu}$ with utilization ρ in equations (5.4), we obtain the second and the third formula in equation (5.7). The mean number of customers $E[X]$ in the system is computed by using the second and the third formula in (5.7) as the state probabilities that work as the weights for the number of customers in the appropriate state. The variance $D^2[X]$ is computed in the similar way.

$$\begin{aligned} \rho &= \frac{\lambda}{\mu} \\ \pi_i &= \rho^i \pi_0, \ i = 0, 1, 2, ... \\ \pi_0 &= 1 - \rho \\ E[X] &= \frac{\frac{\lambda}{\mu}}{1 - \frac{\lambda}{\mu}} = \frac{\rho}{1-\rho} \\ D^2[X] &= \frac{\rho}{(1-\rho)^2} \end{aligned} \tag{5.7}$$

If the distribution of either arrival or service time is not exponential and the scheduling policy is still FIFO, the above results do not hold any more. However, for PS scheduling the formula remain the same although the service time distribution will change, i.e. the above formula also hold for $M/G/1 - PS$-queues.

The total delay D of the system contains both waiting W and the service S times: $D = W + S$. The mean delay can be derived from the Little's formula (see Section 5.1.7) and it holds the same for all work-conserving scheduling policies (FIFO, PS, LIFO, ...). In the case of $M/M/1 - FIFO$-queue, the mean delay is derived from 5.6 and 5.7:

$$E[D] = \frac{E[X]}{\lambda} = \frac{1}{\lambda} * \frac{\rho}{1-\rho} = \frac{1}{\mu - \lambda} \tag{5.8}$$

Waiting time of a customer is given by: $W = D - S$

$$E[W] = E[D] - E[S] = \frac{1}{\mu} * \frac{1}{1-\rho} - \frac{1}{\mu} = \frac{1}{\mu} * \frac{\rho}{1-\rho}$$

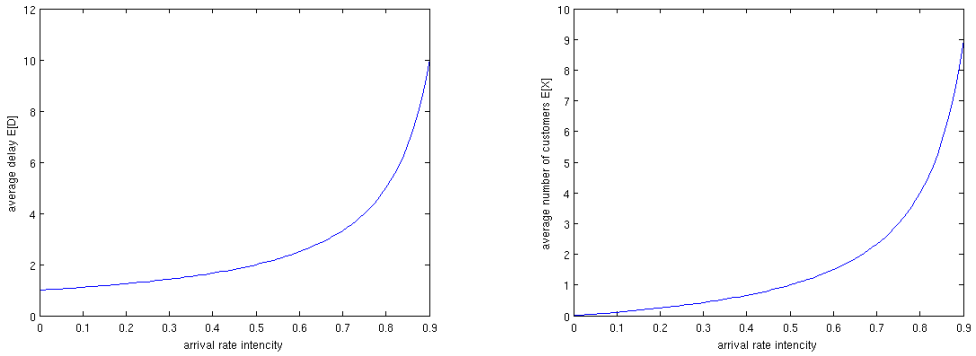

Figure 5.4: Left: The mean delay as the function of the arrival rate. Right: The mean number of customers as the function of the arrival rate

From Figure 5.4 we can see how the average delay and the average customers number depends on the arrival rate.

5.1.9 n $M/M/1$ queues

By the configuration of n $M/M/1$ queues, we will call the setting in which there are n queues connected in parallel, and each of the queue is $M/M/1$. Such configuration is reasonable when there is a Poisson arrival process which is then split according to a general probability function into n Poisson processes (splitting Poisson process results in several Poisson processes as was

shown in Section 5.1.3). Each of newly created processes 'generates' customers, which enter their own queue to get service. Such configuration could be, for example, thought as a representation of the shop's checkouts.

The rate of service time is, as before, μ and the rate of arrival time is λ. Because we have n queues and the traffic is (as it is possible equally) split between them, each queue is experiencing only $\frac{\lambda}{n}$ of the whole traffic. A single customer sees the system as it would be a $M/M/1$ queue with a traffic intensity $\frac{\lambda}{n}$. From that fact we can derive all property formula for n $M/M/1$ queue by just substituting λ with $\frac{\lambda}{n}$ in $M/M/1$ queue formula (5.7) and (5.8). We get:

$$E[X] = \frac{\frac{\lambda/n}{\mu}}{1 - \frac{\lambda/n}{\mu}} \tag{5.9}$$

$$E[D] = \frac{1}{\mu - \lambda/n} \tag{5.10}$$

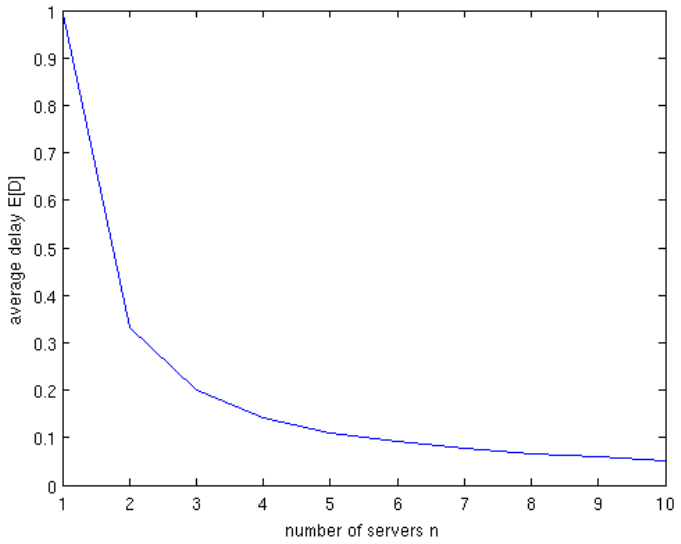

Figure 5.5: The mean delay as a function of the number of servers with fixed λ and μ

From the Figures 5.5 and 5.6 we can see how the average delay decreases when we increase the number of servers as well as how many servers we need

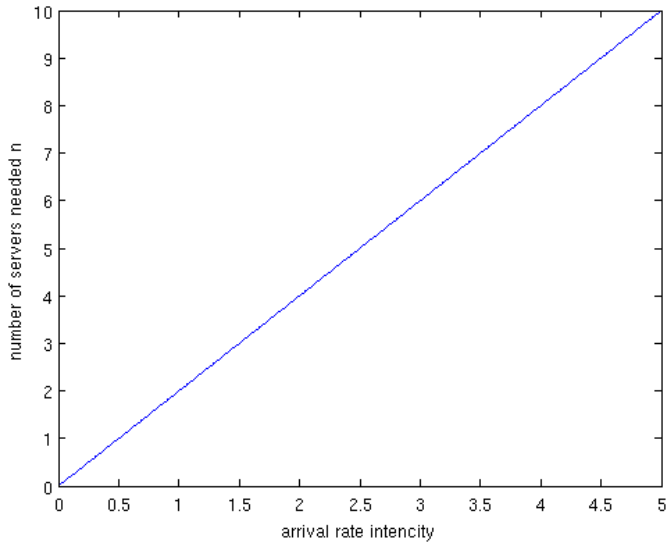

Figure 5.6: The required number of servers as the function of customers' arrival rate when mean delay is fixed

to add to the system to maintain the pre-defined average delay if the arrival rate increases.

5.1.10 Non-homogeneous Poisson process

Although Poisson process is a common process class to represent real phenomena, it is not always a totally accurate model because it considers situations wherein the incoming intensity stays constant. This is rarely the case in the real life as the intensity rather changes over time and may even possess a cyclic behavior. For example, in the public traffic there are rush hours during 7-8 and 16-18, which actually is seen in the more dense transportation's time schedules. Airline flights are used more on weekdays, which is seen in the increased flight tickets. Shops experiences rushes, for example, during pre-Christmas periods.

Those and many others are phenomena that can be modeled with a *non-homogeneous (non-stationary) Poisson* process. Non-homogeneous Poisson process is a Poisson process whose intensity is not a constant any more but a function of time. Formally, a non-homogeneous Poisson process can be defined as follows:

$$A.\ \text{The process has independent increments}$$
$$B.\ Pr[N(t+dt)-N(t)] \begin{cases} = 0] = 1-\lambda(t)dt \\ = 1] = \lambda(t)dt \\ > 1] = 0 \end{cases}$$

where $N(t)$ is the counting process, $\lambda(t)$ is the arrival rate at time t and dt is a differential sized time interval.

So the non-homogeneous Poisson process is defined in a simpler way than the homogeneous Poisson process (see Section 5.1.3) except that the incoming intensity does not vary in time. It is important to remember that because the interarrival times of non-homogeneous Poisson process are not exponentially distributed, the process does not possess the memoryless property. This leads to the problem — we cannot use the steady state formulas deduced earlier. Still, [36] states that the results deduced assuming the homogeneous Poisson process are applicable in non-homogeneous case as long as $\rho(t)$ — the absolute utilization at time t, is less than 1 for all t and the rate function varies slowly. Or as the author has stated: "the steady-state approximation can only be accurate if the change in $\rho(t)$ during one relaxation time is small in comparison to the average queue length". The relaxation time defines the time after which the system 'forgets' its initial state. Mathematically the approximation is valid if:

$$\Delta = \left[\frac{1}{\mu(t)}\right]\left[\frac{1}{[1-\rho(t)]^3}\right]\left[\frac{d\rho(t)}{dt}\right],\ \rho(t) < 1$$
$$\Delta << 1$$

Interestingly, there are different opinions about the predictability of the arrival rate function. [36] claims that the varying value can be well estimated and predicted. The author even says that: "If the arrival pattern is not predictable, then it should not be modeled as a non-stationary Poisson process. The essence of the non-stationary Poisson process is predictable variability in the arrival process." On the other side of spectrum is study conducted by [20] on the predictability of arrival calls rate to a small banking call center. They performed a series of statistical tests on data gathered and tried to prove that the number of calls arrived during the same day and the same time over a year (say every Monday between 8 and 9) was drawn from the identically distributed Poisson distribution. Almost in all cases the null hypothesis on the identical distribution was refuted. We will return to the statistical test in the later sections.

5.1.11 Queueing theory models

We have presented central queueing theory results that we will use for modeling the problem of our interest. Still, it is instructive to investigate of which extent queueing theory has been used in industry and in the academic world. Here we will briefly describe a few queueing theory models used in other researchers' articles.

In [21] the $M/G/1/K * PS$ queue was used to model a single web server. The authors assumed that the customers arrive according to Poisson process. The server can process at most K customers, if more customers arrives when all K places are reserved, the customer is blocked. The server service time follows general distribution and additionally operates in process sharing fashion. This nicely justifies the use of the results of $M/M/1/K$ queue, which is simpler than general $M/G/1/K$ queue. The authors stress that compared to the other models their approach is a beneficial because of its simplicity, although it may lack the validity in the server overloaded work region. The authors also applied a maximum likelihood estimation to obtain model parameters — average service time $\overline{x}$ and the number of service places K. They preset the maximum number of TCP-connections their web-server could process and generated a random workload. Then they measured the average response time experienced by the virtual customers for several data sets. Since the average response time is a sample average, it is due to central limit theorem approximately normal. Several estimates for the average response time are substituted into normal probability function. The functions values are multiplied together, and the result is tried to be maximized through varying the values of $\overline{x}$ and K. The authors applied a brute force optimization algorithm to derive those. The study of [21] is advantageous because of its simplicity but criticism can be introduced about the justice to use Poisson process as the representation of arrivals, as no real data analysis was made on it.

In [13], written by the same authors as [21], a better model for an overload region is presented. The authors use a two-state Markov Modulated Poisson Process (MMPP) in order to simulate bursty traffic with random peaks. A two-state MMPP is simple Markov chain with two states, in the first state the Markov chain outputs the incoming customers with the intensity λ_1 and in the second with the intensity λ_2 ($\lambda_1 << \lambda_2$), and the Markov chain changes between those two states with intensities r_1 and r_2. As the model of the service is the same as in [21], so the whole system is $MMPP/G/1/K * PS$. The maximum likelihood method is used again to obtain the estimates for the mean service time and number of server places. Although, the approach

of [13] is more realistic than in [21] because of the use of MMPP, the authors did not compare the resulted traffic generated by MMPP with some real server traffic.

In [47] the authors model a 3-tier web-server as a closed queueing network. Traditionally web-servers consist of three layers — a web server processing the customers' requests, an application server implementing business logic and a database server storing important records. The authors model each of those as a service with its own queue connected in series, such that the customer served by the web server is proceeded to the application server. The authors made a couple of simplifications compared to the real setting, for instance the service time is assumed to be workload independent and follow an exponential distribution. By using mean-value analysis, the queueing network parameters, such as mean delay and throughput, are computed. The authors ran tests in the described setting, measured the system parameters and then compared them to the mathematically deduced ones. The authors report that the error between the empirical and theoretical values is small, which says that the model represent the real setting accurately. Still, as in the two first studies, [47] lacks the investigation about the real traffic and service. The authors use TPC-W benchmark to generate the workload but did not give justifications behind that.

5.1.12 Mathematical modeling with queueing theory

If we want to model the problem with queueing theory, first we should test whether the data coming from the problem's setting fulfill the set of conditions. In the case those conditions do not hold, it is important to understand the reasons for that. Usually when we discover settings breaking the conditions, it is easier to justify immediately whether queueing theory is applicable for the problem, instead of running the large amount of statistical tests.

Models are simplifications of real situations, such that it is rare luck if all model's conditions are satisfied. The more realistic case is that the data from a real world phenomenon does not pass all tests. Thus we are faced with a choice of how the phenomenon should be modeled. There are four general solutions to the situation when the data coming from the real world problem and the model conditions mismatch:

1. When the reasons that break the assumptions are revealed, they can be taken into account in the theory. Usually, this brings us a more complicated model. For instance, if in a queueing system customers arrive in

groups, the queueing theory extension named bulk arrival can be used. Still, the use of more complicated models should be considered properly, because the resulting equations might be intractable that benefits of using the theory diminish greatly.

2. We can find the data from the similar real world phenomenon, which pass the tests. Thus the model is not universal for the current problem but solves only a subset of it. Again, the reasons behind the inconsistency should be found such that we are able to argue why different datasets from the same phenomenon have divergent properties. For example, if in a queueing system Poisson arrivals are assumed, the customers' population should be truly large. There could be similar settings such as high throughput scientific clusters, but the incoming traffic does not seem to come from Poisson distribution in one of them. The reason could be simply that there are too few customers in one of the those settings compared to the another.

3. If the data failed to pass the tests, the attractiveness of model simplicity can drive to use a model as an approximation. If the model is used as an approximation, it is desired to check how large is the error between the model predictions and actual system parameters.

4. The last possibility is to reject the model as inappropriate. Again, the reasons have to be mentioned and the alternative model is presented.

In Figure 5.7 we present the path from the real phenomenon data to the model capable to describe it. In addition to cases when the data does not fulfill the conditions of the model, we took into account the possibilities to choose simpler models at the expense of accuracy. The first condition that has to be true in all queueing theory models is that arrivals must be independent of each other. If they do not (and as we see in Section 6), we can represent the group of dependent arrivals as a single multi-job arrival. This of course requires both more complicated pre-processing as interdependent jobs has to be identified and resulting formula for the such model (called bulk arrival). Nevertheless both models need to pass Poisson distribution test. If this is not the case either, there is still the model for general arrival distributions but its shortage is that it provide just few closed-form formulas. In such situations we prefer to use simpler models (such as $M/G/1$) although they are not necessarily 100 % accurate. In Figure 5.7 we marked the path for our model choice as a dotted line for HIP's cluster data and as a dashed line for NorduGrid data.

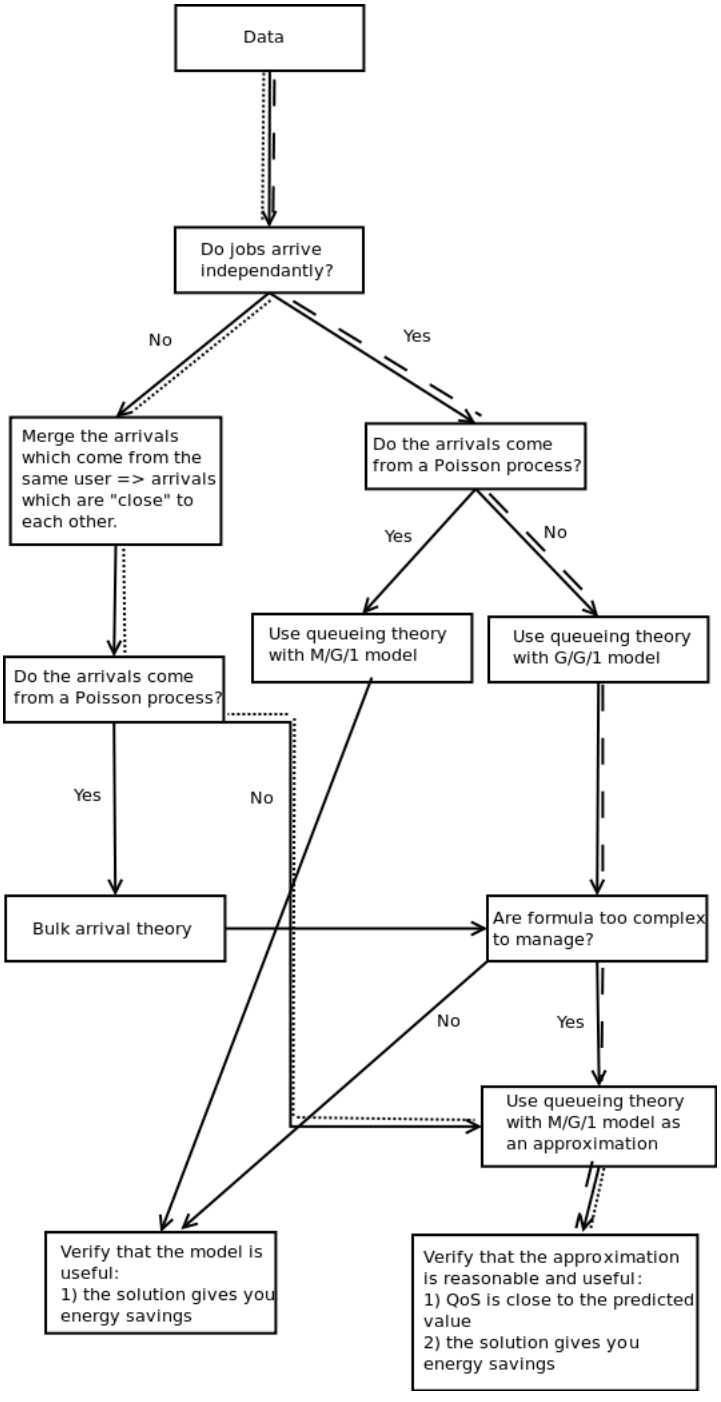

Figure 5.7: Possible paths from the raw data to the model representing the data

5.2 Control theory

5.2.1 Control theory basics

Control theory provides mathematical calculus for automation technology field. In an automation there is always a system we wish to control, and it is called a process. The process (system) has its own specific dynamics, which means that it responds to an input in the way which is governed by the equations particular to this process. There is also a process variable or process value, which can be thought as a process output we are interested in. We control the process by changing the value of a process manipulated variable, which can be thought as an input to the process. We also have a set-point, which is the desired or 'correct' value of the process variable. Process control happens through measuring the value of process variable, comparing it to the set-point, and based on the difference and knowledge of the process dynamics, changing the process variable. In some situations there is also a disturbance, which is the second input variable (in addition to the process manipulated variable). The disturbance cannot be controlled, so that we after the process achieved the set-point, we should not leave the process alone, because a new disturbance can eventually interfere it. Control theory provides mathematical tools that can bind all those terms nicely together, such that we are able to see the characteristics of the system dynamics and the impact of the control on it. [43]

Let us now illustrate those concepts with a simple and practical example. We want to take a bath and fill the basin with water coming from a tap. We wish water in the basin to be of the specified temperature, i.e. the basin water is the set-point. But the water temperature is what it is and can take any values, and it actually forms the process variable. Still we can control the temperature of the incoming water by adjusting the valve, such that the valve position acts as the manipulated variable. The process is the system of the tape and the basin, it can be, for example, expressed in mathematical formula how fast the tape water temperature actually reaches the value set by the valve. We can also add a disturbance to our model, which can be the phenomenon that the water in the basin cools down as time passes. The purpose of the control is to provide us with the positions of the valve along the time such that at the end we obtain the basin filled with a right-temperature-water.

Control systems are characterized into two main classes — feedback - and open loop-systems. Feedback systems contain a feedback loop, which gives

possibility to observe values of the process variable. When there is an observation of the process variable, it is easier to adjust the manipulated variable to the set-point. In our water basin example, the possibility to measure basin water temperature plays the role of the feedback loop. In contrast, in open loop-systems there is no knowledge of the process variable value, and all what can be done is to adjust the manipulated variable and hope it will be close enough to the set-point. Control systems are usually depicted with block diagrams, in Figures 5.8 and 5.9 is shown block diagrams of a water basin process with a feedback- and open loop-control.

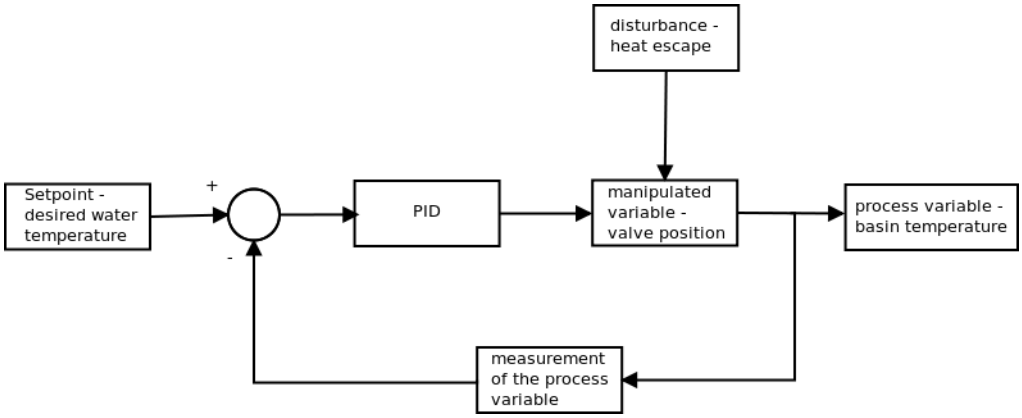

Figure 5.8: Feed-back loop diagram of the water basin example

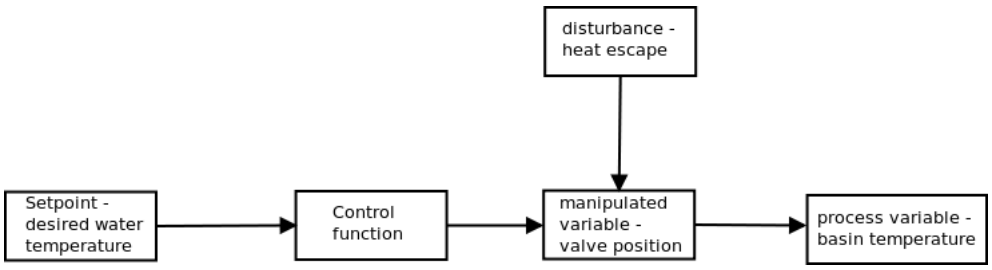

Figure 5.9: Open loop diagram of the water basin example

Further, we will also need a few important concepts that describe the performance of the control system. Suppose, there a new set-point we want the process to reach. The major characteristics of the control are:

- Rise time: the time needed for the process to reach 90% of the set-point for the first time.

- Overshoot: how much the maximum value of the process variable is higher than the steady state value of it during the adjustment phase.
- Settling time: the time needed for the process to converge into the steady state.
- Steady-state error: the difference between the steady state value and the set-point.

5.2.2 PID-controllers

Proportional-Integral-Derivative-controllers (PID) are widely applied in the industry to control the processes when its underlying dynamics cannot be known. In this way the process is regarded as a black box controlled only via the manipulated variable. Although the PID is a very simple and powerful tool, it does not provide the optimal control. The performance of PID can be improved by adding a so called feed-forward control block which is aware of the process dynamics.

PID operates by taking as an input the error between the setpoint and the measured value of the process variable. Then, the PID calculates its output to the process according to the formula:

$$u = K_P e + K_I \int e dt + K_D \frac{de}{dt}$$

where e is the error (the difference between the set-point and the current process variable value), K_P is the coefficient of the proportional gain, K_I is the coefficient of the integral gain and K_D is the coefficient of the derivative gain. By fine-tuning the factors K_P, K_I and K_D in the PID, the control is improved in the terms of characteristics described in the previous subsection. K_P is used to decrease the rise time, K_I is used to reduce overshoot and settling time and K_D is used to remove the steady-state error. There are a number of methods for evaluations of those coefficients, normally they are iterative — based on the response for the initial settings, the coefficients are updated.

5.3 Statistical testing and estimation

In this Section we will introduce mathematical tools required to justify the model we will use. Because our preliminary assumption as well as the field literature use Poisson processes to model customers arrivals, we will primarily concentrate on tests and estimators related to Poisson processes.

5.3.1 Graphical tests

In graphical tests we simply plot the data and examine, whether it has any visibly recognizable patterns. If some patterns are observed, the most important task is to reveal which factors caused the patterns. If the successive arrivals are not independent, why they are not independent? If the interarrival times are not exponential, why they are not? [36]

- **Stationarity: Cumulative arrivals.** Plot the cumulative arrival curve $A(t)$. Draw a straight line between $A(0)$ and $A(T)$, where 0 and T are the start and end points of the interval considered respectively. There should not be any pattern in deviation from the straight line and the difference between the line and the values of $A(t)$ should be small for all values of t. In this work we will see the use of this plot in Figures 6.1, 6.2, 6.3, 6.6, 6.9 and 6.12.

- **Stationarity/Independence: Interarrival times.** Plot the interarrival times as a time series (on the x-axis are ordered arrivals, on the y-axis are their interarrival times). The points should be randomly distributed over y-axis. There should not be any cyclic patterns. In this work we will see the use of this plot in Figures 6.4, 6.7, 6.10 and 6.13.

- **Independence.** Plot the points $(X_n, X_{n-1}), n = 2, 3, ...$, where X_n represents the nth interarrival time. It should be impossible to approximate the data with a straight line. In this work we will see the use of this plot in Figures 6.5, 6.8, 6.11 and 6.14.

- **Exponential interarrival (service time) distribution.** Plot the empirical distribution of the interarrival (service) times and the theoretical exponential distribution with $\lambda = A(T)/T$, where T is the length of the interval observed. There should not be large and systematic deviations between the distributions. In this work we will see the use of this plot in Figures 6.17, 6.18, 6.19, 6.20, 6.21, 6.21, 6.22 and 6.23.

5.3.2 Statistical tests

Graphical tests can give only a guidance for arguing whether an observed data set follows Poisson distribution. For instance some deviations from the correct distribution could be present but how large difference is acceptable. To obtain quantitative measure of the similarity or dissimilarity, statistical tests are used. Statistic is a function of a data and thus itself is random variable. We will examine a few statistical tests and statistics related to them. [36]

Interarrival time distribution. To test, whether interarrival times follow exponential distribution, we can use two statistical tests — chi-squared test and Kolmogorov-Smirnov test. Because we are interested in non-homogeneous Poisson process and thus cannot take the whole data set for testing. This is because we want to test whether the underlying process follow Poisson distribution at *some* time interval, but we already know that it does not follow (stationary) Poisson process at the whole time interval. Thus we will apply the approach described in [20]. In that work the authors split the data set into small equal-size subsets, say 6 minutes, and performed the testing on them.

In chi-squared test the theoretical distribution is generated. In the case of Poisson arrival process we will generate the exponential distribution with the mean $A(T)/T$, where T is the length of the examined period and $A(T)$ is the number of arrived customers. This generated distribution will be called expected distribution in contrast to the observed distribution. Then both the expected and the observed distributions are split into equally wide 'bins', the data points which occur within bounds of any bin are put into that bin. If there are less than 5 points in some bin, it is merged with the neighbor bin. The test statistic is:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \tag{5.11}$$

where O_i is an observed frequency, i.e. the number of observed items in the bin, E_i is an expected frequency, i.e. the number of expected items in the bin and n is the number of frequency classes (bins). The statistic χ^2 approaches asymptotically χ^2-distribution with the $n - 1$ degrees of freedom. [52]

Kolmogorov-Smirnov test operates in the similar way as chi-squared test but instead of placing data points into bins it compares the values of data points not their amount. In Kolmogorov-Smirnov test the theoretical exponential

distribution is generated with the arrival rate λ equal to $A(T)/T$. Then the maximum deviation from the empirical distribution is measured:

$$D = max_x|F_O(x) - F_E(x)| \tag{5.12}$$

where $F_O(x)$ is the observed cumulative probability function and $F_E(x)$ is the expected cumulative probability function. The test statistic D follows its own distribution, which is derived for different sizes n. Kolmogorov-Smirnov test is more accurate and simple than chi-squared test. [49]

Interarrival time independence. The most important property of arriving process is that interarrival times must be independent of each other, or formally: X_n must independent of $X_{n-m},\ m = 1, 2, 3, ...$ Generally, it can be very difficult to prove that each data point is independent of all other. Usually, if there are no reasons to argue otherwise, we assume that whether data points are dependent they depend on the their neighbors. So, we assume that it is sufficient to verify whether successive points are independent or not.

There is a chi-squared test for independence similarly to chi-squared test for distributions. The test is based on placing data points into contingency table and comparing the number of points in each category. The expected and observed frequencies should be roughly the same if the null hypothesis is correct. The disadvantage of this test is that a great amount of observations is needed to be collected the test to be powerful. [36]

The alternative approach is to test weaker hypothesis — whether the correlation coefficient between X_n and X_{n-1} equals zero. Independent random variables have zero correlation but the reverse is not necessarily true.

Chapter 6

Statistical testing for real world data

Before constructing any model we analyzed the data to assist the choice of the right data representation. We analyzed two datasets, one obtained from the HIP's cluster and the second collected from the NorduGrid[1]. HIP cluster was used just by a few users whereas in Nordu there was about 201-500 users and about 1000-5000 computation cores.

We found out just by looking the cluster traces that the arriving jobs are strongly coupled. The reason is that a user seldom sends a single job but instead several jobs are generated and sent to a cluster simultaneously. Unfortunately, this breaks the important property of jobs' independent arrival. One way to solve this problem is to treat the group arrivals as single job. But such pre-process raises its own problems. First, it is hard to determine which jobs belong to which group, let us imagine a user generates a group of jobs and sends them to the cluster or the same user sends several individual jobs one after another with a short interval between submissions. So, it is not well specified within how wide time window the jobs submitted by the same user should be regarded belonging to the same job group. Second, the calculus needed to solve the resulting model is more complicated than in the single jobs case. In the light of those difficulties, we decided to conduct our analysis of the data in two different ways — first, HIP dataset is preprocessed by merging jobs sent in the same time interval (which equals 1 second) into a single job, second both the preprocessed HIP cluster and NorduGrid data are analyzed by plotting them and performing statistical testing on them to see how well they follow Poisson distribution (see Section 5.3).

[1] http://gwa.ewi.tudelft.nl/pmwiki/

6.1 Analysis of HIP-cluster data

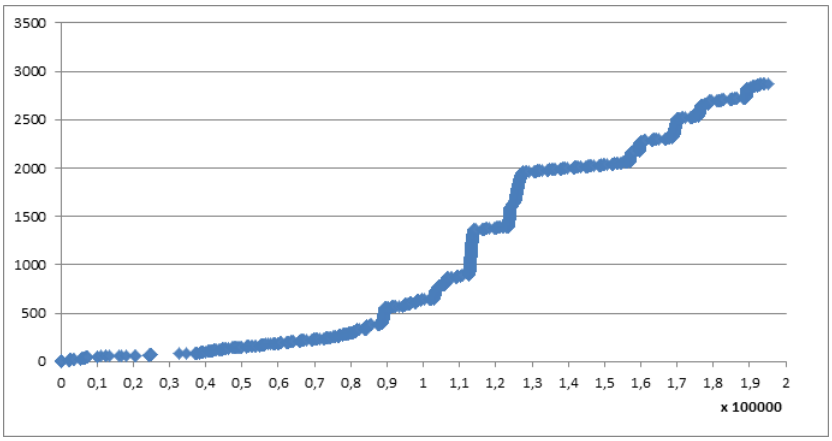

Figure 6.1: Cumulative arrivals plot of HIP's datacluster, number of jobs arrived up to the current moment expressed as a function of time in seconds

We preprocessed HIP cluster data and plotted it into a cumulative arrival diagram (see Figure 6.1). By looking at the diagram we tried to identify the periods of the constant arrival rate. Here we want to stress that those piecewise linear periods can be seen only afterwards, when all data is collected and plotted. If we look at the random time intervals, even as short as 600 seconds, the graph does not resemble Poisson distribution at all (see Figure 6.2). We identified the following piecewise linear periods in Figure 6.1:

1. 0-37500
2. 37500-85000
3. 85000-90000
4. 90000-102500
5. 102500-107500
6. 107500-112500
7. 112500-117500
8. 117500-125000
9. 125000-127500

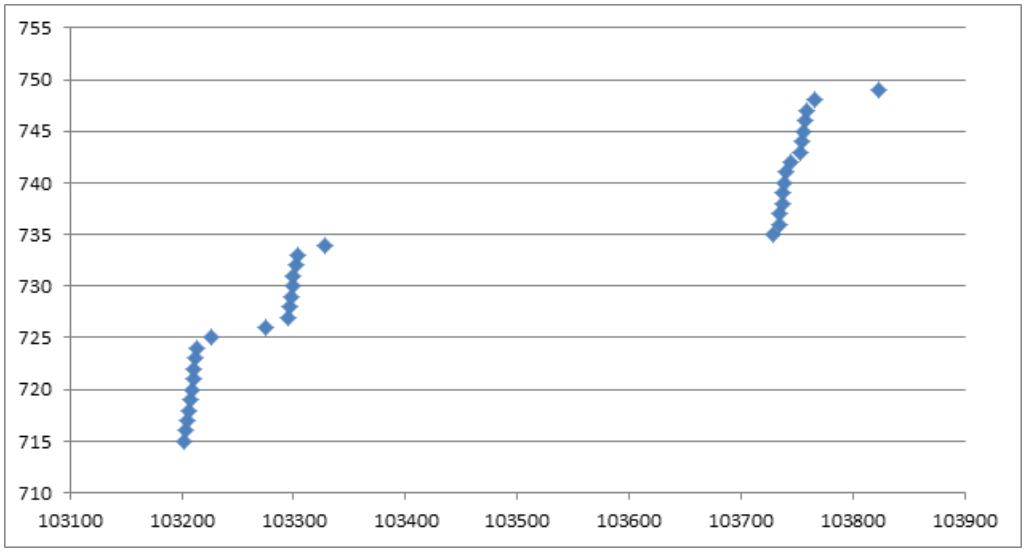

Figure 6.2: Cumulative arrivals plot of the random interval of the Figure 6.1, number of jobs arrived up to the current moment expressed as a function of time in seconds

10. 127500-160000

11. 160000-195000

Below we plot the diagrams of intervals 1, 2, 4 and 10 for checking for the properties of Poisson arrivals (stationary, cumulative) and independence (stationary/independence and independence) described in the Section 5.3.1. Unfortunately, all intervals except 10th exhibit regular deviations. For example in interval 1 many jobs arrive almost simultaneously. The interarrival and service time distribution show interesting behavior — almost the whole probability mass is accumulated in the very small interval.

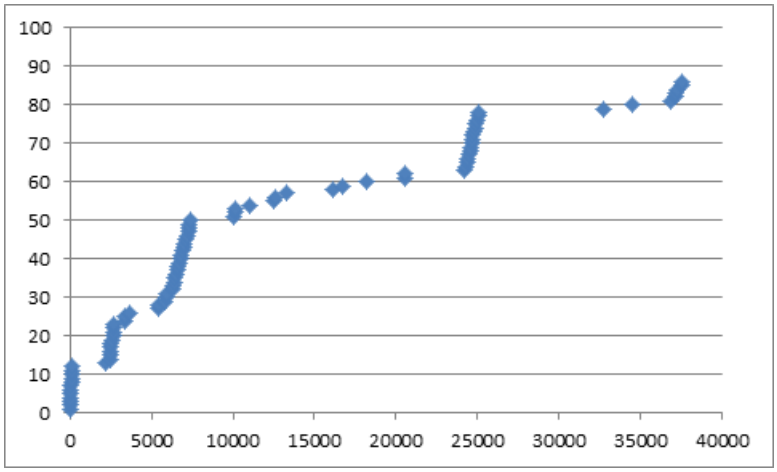

Figure 6.3: Cumulative arrivals plot of HIP's datacluster, interval 1, number of jobs arrived up to the current moment expressed as a function of time in seconds

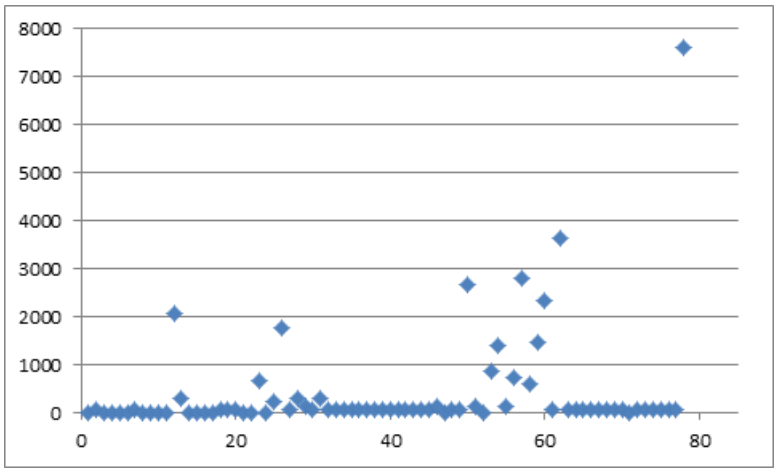

Figure 6.4: Interarrival time plot of HIP's datacluster, interval 1, the time between the subsequent jobs in seconds as a function of job ordering number

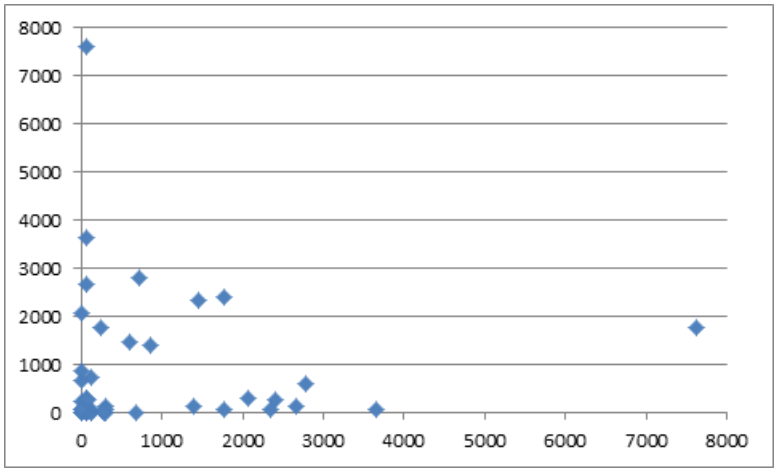

Figure 6.5: Interarrival time dependance on the previous interarrival time plot of HIP's datacluster, interval 1, time of the interarrival time in seconds as a function of the previous interarrival time

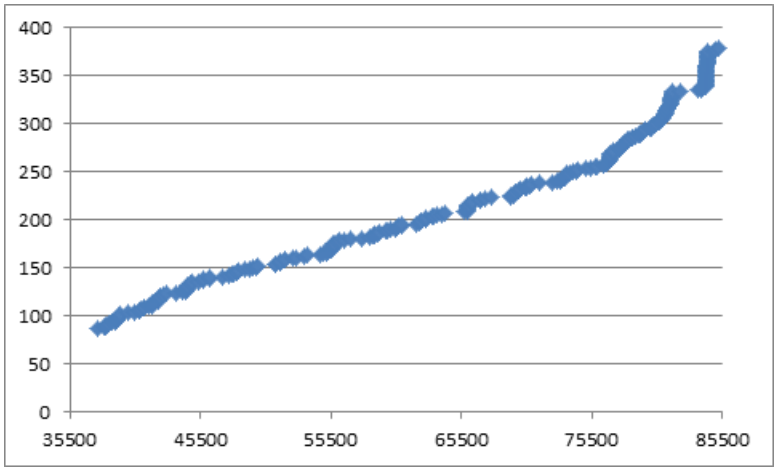

Figure 6.6: Cumulative arrivals plot of HIP's datacluster, interval 2, number of jobs arrived up to the current moment expressed as a function of time in seconds

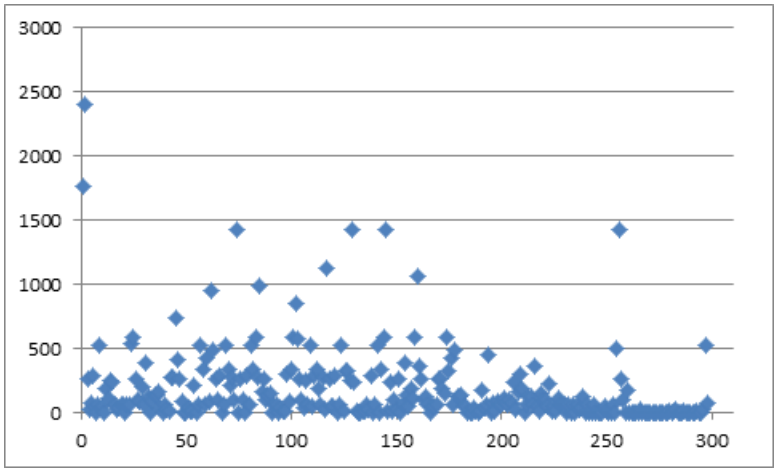

Figure 6.7: Interarrival time plot of HIP's datacluster, interval 2, the time between the subsequent jobs in seconds as a function of job ordering number

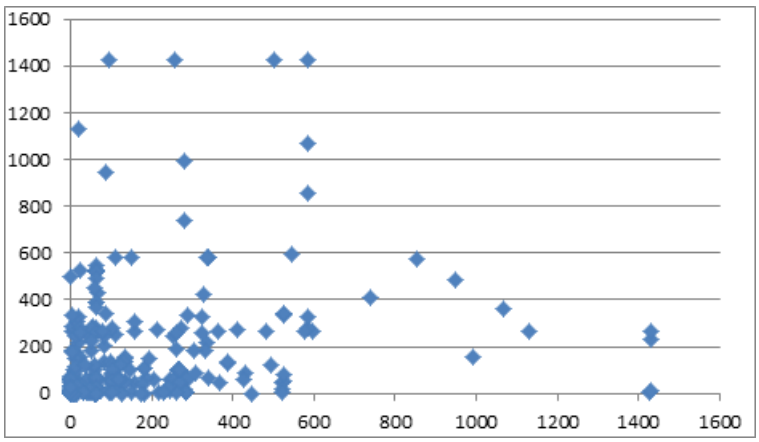

Figure 6.8: Interarrival time dependance on the previous interarrival time plot of HIP's datacluster, interval 2, time of the interarrival time in seconds as a function of the previous interarrival time

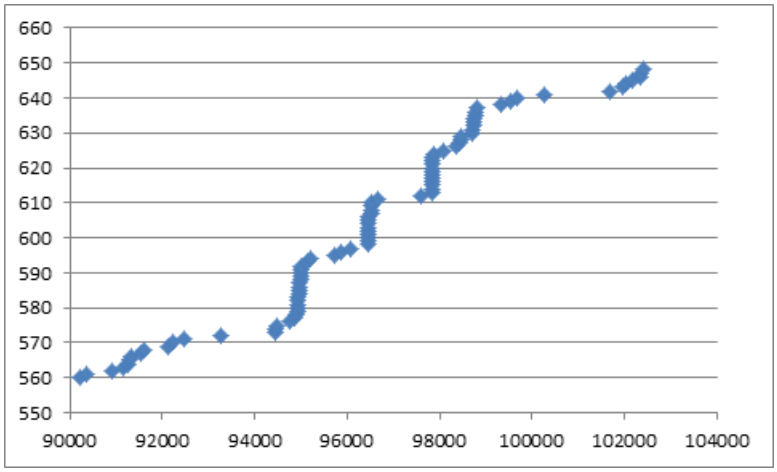

Figure 6.9: Cumulative arrivals plot of HIP's datacluster, interval 4, number of jobs arrived up to the current moment expressed as a function of time in seconds

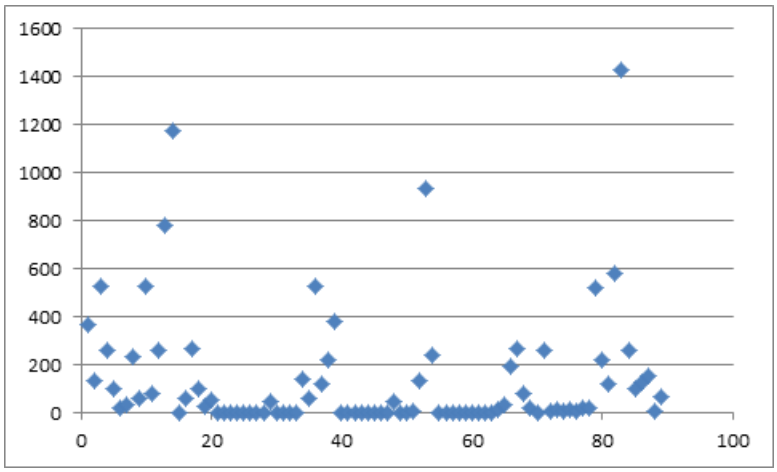

Figure 6.10: Interarrival time plot of HIP's datacluster, interval 4, the time between the subsequent jobs in seconds as a function of job ordering number

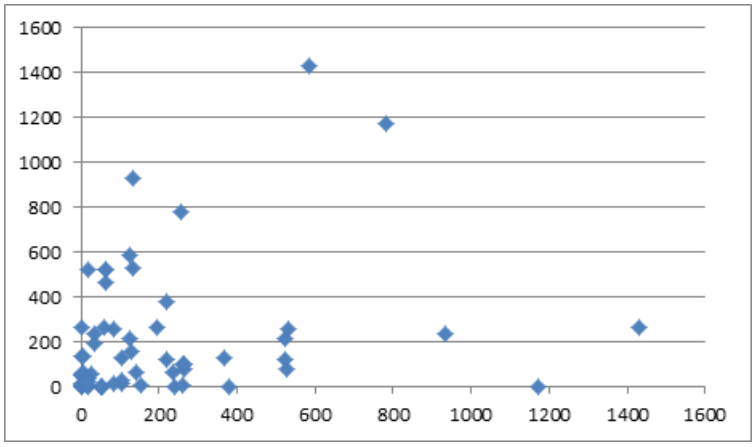

Figure 6.11: Interarrival time dependance on the previous interarrival time plot of HIP's datacluster, interval 4, time of the interarrival time in seconds as a function of the previous interarrival time

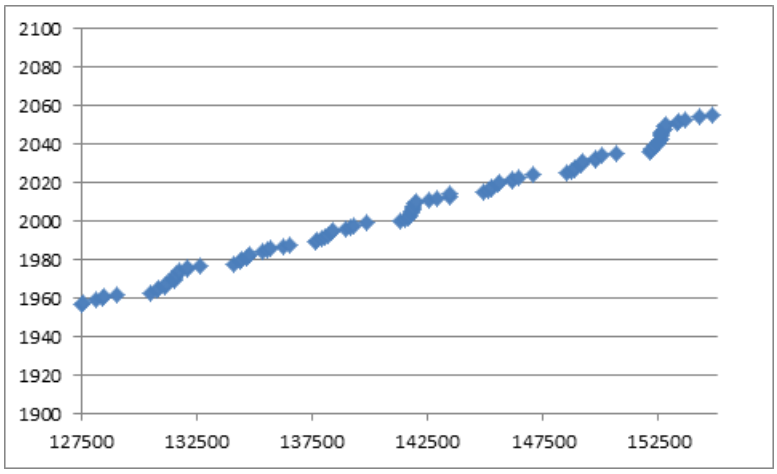

Figure 6.12: Cumulative arrivals plot of HIP's datacluster, interval 10, number of jobs arrived up to the current moment expressed as a function of time in seconds

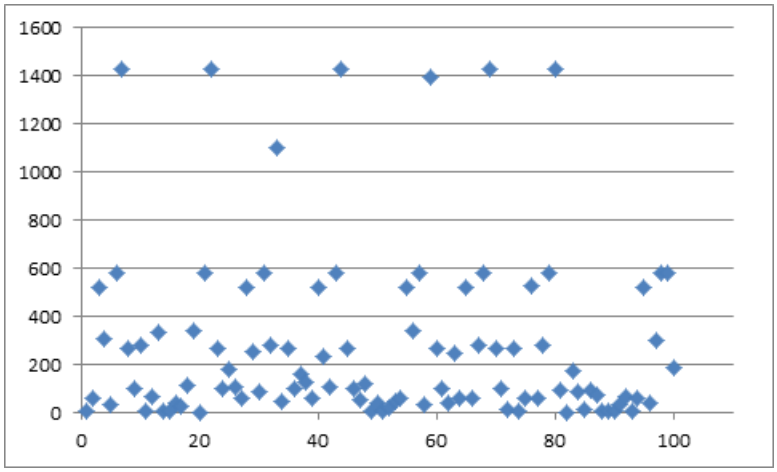

Figure 6.13: Interarrival time plot of HIP's datacluster, interval 10, the time between the subsequent jobs in seconds as a function of job ordering number

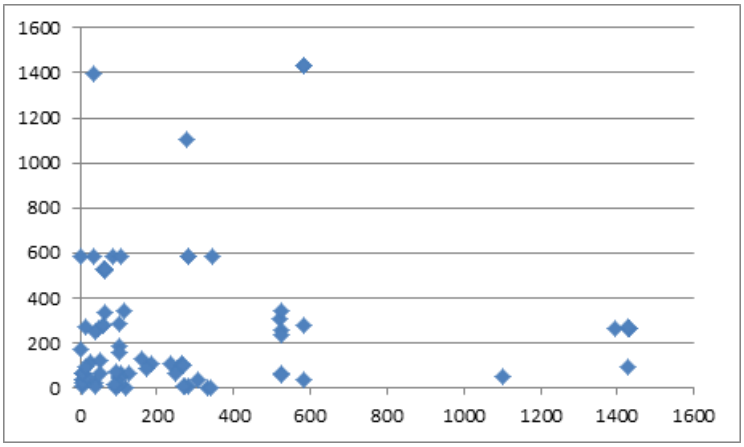

Figure 6.14: Interarrival time dependence on the previous interarrival time plot of HIP's datacluster, interval 10, time of the interarrival time in seconds as a function of the previous interarrival time

We performed Kolmogorov-Smirnov tests (see formula (5.12)) for intervals 1, 4 and 10, because their shape is mostly close to the straight line. For all those intervals the null hypothesis that the samples would come from exponential distribution was rejected with p-value equaling 0 with 4 decimals (the appropriate test statistics for those were 0.41, 0.23, 0.09 for intervals 1, 4, and 10 respectively). For comparison we also present the chi-squared test

results (see formula (5.11)) for interval 10, which actually failed to reject the null hypothesis with the p-value equaling 0.14995. We also performed t-test to verify the interdependency of points in the dataset. The t-test for zero-correlation gave us statistic equaling 0.1311, which resulted p-value of 0.2290, i.e. the test failed to reject the null hypothesis that the interarrival times would be independent on each other. In Figures 6.15 and 6.16 you can see the observed data plot for interval 10 and exponential plot with the mean derived from the observed data.

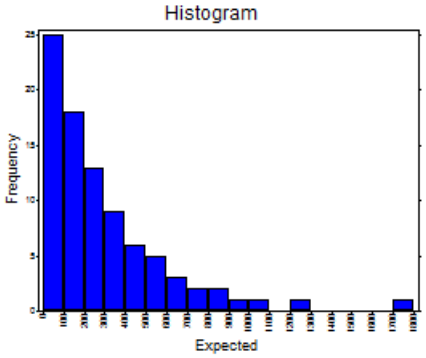

Figure 6.15: Exponential probability distribution function generated with λ derived from interarrival times of interval 10 (see Figure 6.13)

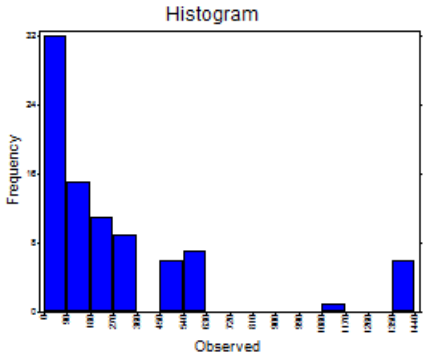

Figure 6.16: Histogram of interarrival times of interval 10 (see Figure 6.13)

We also took a look at service time distribution by plotting different-size samples from different time intervals, see Figure 6.17.

6.2 Analysis of NorduGrid data

We also plotted the data obtained from NorduGrid cluster. We plotted different-size intervals from different time windows (see Figures 6.18, 6.19 and 6.20). The first property we discovered from interarrival plots that they had a great amount of short jobs and a few very long jobs. The results are summarized in Table 6.2. The same analysis we performed on service time. The different size samples are depicted on Figures 6.21, 6.22 and 6.23 and their statistics are summarized in Table 6.2. We see the same behavior as with interarrival times — most of the jobs are relatively short. This leads to the a suggestion that such jobs could be approximated as deterministic jobs (see Section 5.1.2 for deterministic process definition).

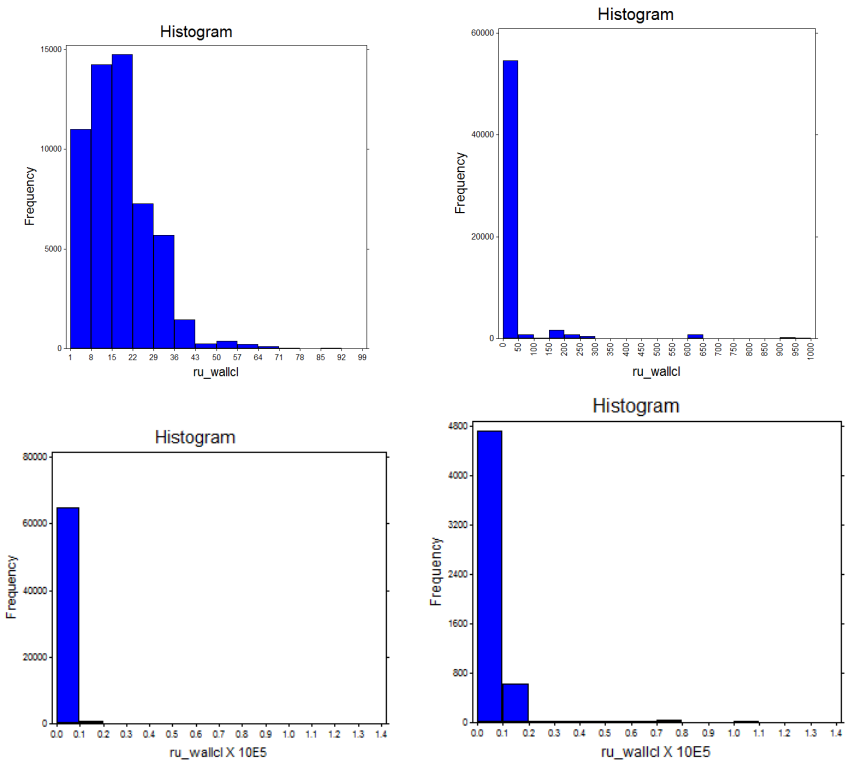

Figure 6.17: Statistics of different sample and different sample-size service times in HIP's cluster, frequency of jobs as a function of time

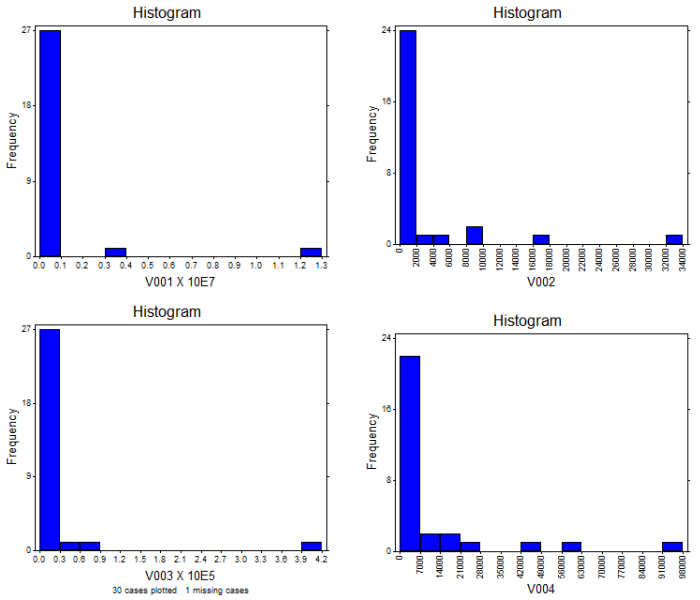

Figure 6.18: Interarrival time histograms for different samples in NorduGrid, frequency of jobs as a function of time in seconds

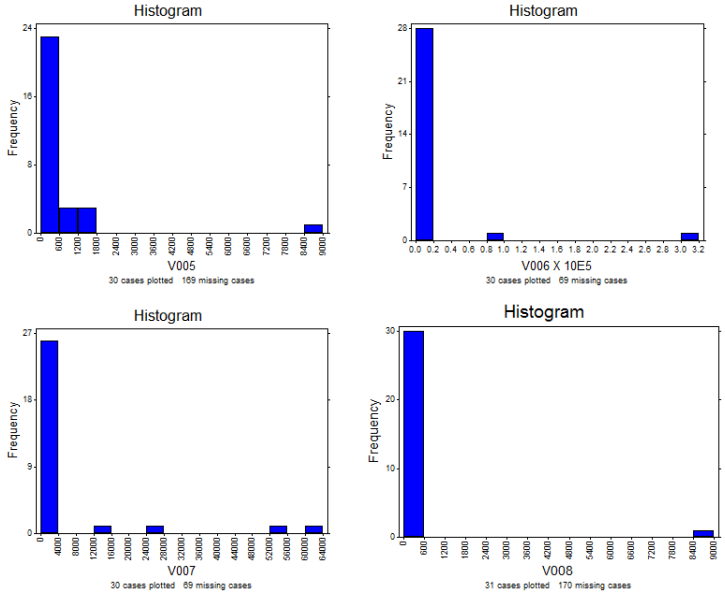

Figure 6.19: Interarrival time histograms for different samples in NorduGrid, frequency of jobs as a function of time in seconds

Table 6.1: Table (statistics of interarrival times of different-size samples in NorduGrid)

sample	sample size	mean	standard deviation	min	max
V001	29	581779	2.469E+06	0	1.296E+07
V002	30	2580.8	6959.4	2	33786
V003	30	19238	76954	0	419106
V004	30	9756.1	21595	0	97077
V005	30	581.40	1639.6	2	8844.0
V006	30	13678	58576	1	312071
V007	30	5697.2	15010	1	60577
V008	31	283.65	1566.3	1	8723.0
V009	61	425.89	1222.2	1	8844.0
V010	101	2364.5	22918	2	230302
V011	201	442.60	3312.0	1	45734

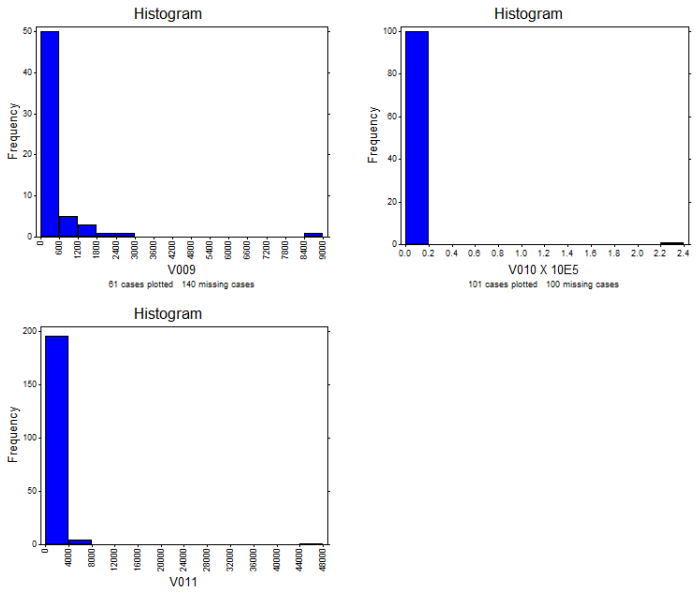

Figure 6.20: Interarrival time histograms for different samples in NorduGrid, frequency of jobs as a function of time in seconds

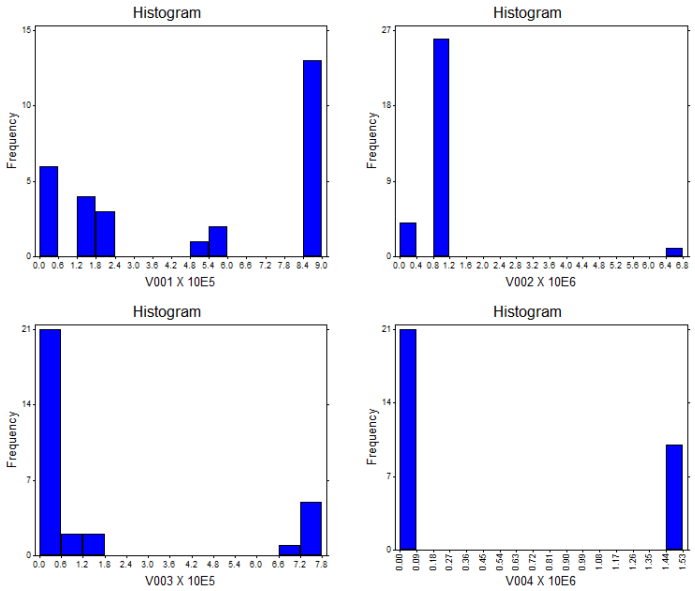

Figure 6.21: Service time histograms for different samples in NorduGrid, frequency of jobs as a function of time in seconds

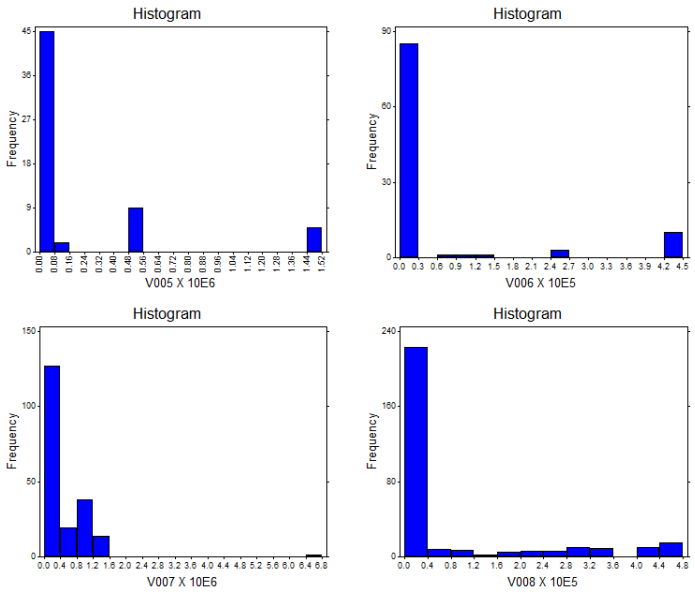

Figure 6.22: Service time histograms for different samples in NorduGrid, frequency of jobs as a function of time in seconds

Table 6.2: Table (statistics of service times of different-size samples in NorduGrid)

sample	sample size	mean	standard deviation	min	max
V001	29	482536	370012	159.00	852948
V002	31	928336	1.117E+06	56.000	6.741E+06
V003	31	161376	299805	24.000	770220
V004	31	490962	721886	55.000	1.521E+06
V005	61	210449	438211	40.000	1.520E+06
V006	101	53828	134156	54.000	433885
V007	199	378539	653044	24.000	6.741E+06
V008	301	75966	144307	18.000	463672
V009	501	3012.7	7212.1	77.000	153999
V010	1000	99712	333375	18.000	6.741E+06

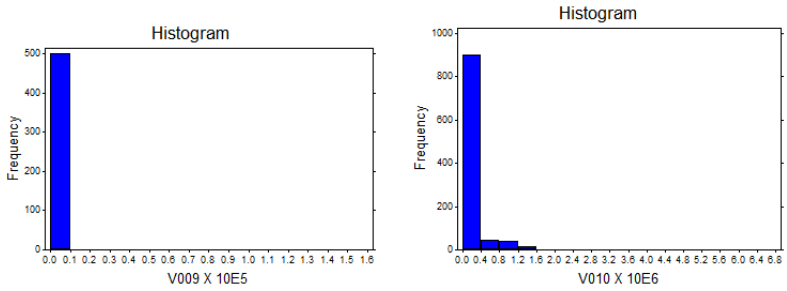

Figure 6.23: Service time histograms for different samples in NorduGrid, frequency of jobs as a function of time in seconds

Chapter 7

Optimization problem and its solution

In this Chapter we will describe the real life problem we aim to solve. We investigate how and why the problem should be modeled using queueing theory. Also, we discuss briefly the alternative modeling methods to solve the problem. We present our solution in terms of queueing theory and also describe its algorithmic implementation. Finally we give a checklist we used as a justification for our approach.

7.1 Problem description and possible solution approaches

As we described in Sections 3.2 and 3.3 our main objective is to optimize the servers utilization. The workload experienced by the servers varies through the day. If the datacenter servers are on all the time, the utilization would be dramatically low during the less active workload periods, because datacenter capacity is usually provisioned for peak rather than average load. Our idea is to adjust the number of servers to the current workload such that the quality of service remains approximately the same. By putting everything together, we formulate the optimization problem as follows: our task is to minimize electricity consumption in a computing cluster; we aim to it by increasing cluster utilization with as minor negative side-effects on quality of service as possible. The current problem can be modeled with different mathematical tools. In this Section we will describe some of approaches as well as their pros and cons.

First, we can try to attack the optimization problem from the scheduling theory perspective. By using the terminology of [12] we can distinguish between high and low server's workload phases. During the high phase there are jobs to be executed and a server is performing useful computation by running the jobs and during the low phase the server is idling because there are no jobs. Also there are two energy states in the server. In the active state the server consumes r energy units per time unit no matter whether it is performing computation or not. In the passive energy state the server does not consume any energy and the transition from the active to the passive state takes no additional energy but the transition from the passive to the active consumes β energy. The purpose is to decide when to perform the transition to minimize the ultimate energy consumption. This kind of power management problem is called *on-line problem* — at any time point we do not know about future events and thus do not know whether it is worth to transfer into passive energy state. The algorithms community resorts to *competitive analysis*, in which the on-line algorithm is compared to the optimal off-line algorithm. The off-line algorithm sees the whole future and thus is able to compute the state transition schedule and in this way achieves the minimum energy consumption. [12]

In [12] a very simple on-line algorithm is presented, which transfers a single server into passive state whenever there is no computation for at least β/r time period for that server, where β is energy units required to transition the server from the sleep state to the active state and r is the power consumption rate in server active state expressed in energy units per time unit. The algorithm is proved to be 2-competitive. This is because the optimal algorithm will not transfer into passive state if the duration of idle period T multiplied by the energy spent is less than state transition energy β, that is $rT < \beta$. The on-line algorithm will spend the same amount of energy as optimal algorithm because the idle period is less than $\beta/r \Rightarrow T < \beta/r \Leftrightarrow rT < \beta$. When the idle period is longer than β/r, the on-line algorithm spends $r * \beta/r$ energy and then transfers into passive energy state and finally goes back to active energy state and consumes β for transition. The optimal algorithm knows in advance the length of the idle period and transfers there immediately in the beginning of it and thus spends only β energy. The result of 2-competitiveness from that. The authors also prove that no deterministic on-line algorithm can beat the competitive ratio of 2.

Second, we can represent mathematically the behavior of the arriving jobs in the terms of probability distributions. If we know also the job's durations or at least their distribution, we can express the problem with queueing theory. As we described earlier, in queueing theory we bind together in terms of

formula the concepts of jobs' arrivals, their duration and quality of service. We can also think that we let some jobs to wait in the queue at the expense of other jobs that will be performed immediately. Interestingly that in [23], which classifies different types of scheduling algorithms, queueing theory is also present in their taxonomy and is regarded as global-static-optimal algorithm (see Figure 7.1). The classification goes as follows: Global means that the decision is made to which computing node the job will be sent, contrary to the local scheduling, wherein the assignment of jobs to the time-slices of a single processor is involved. Static versus dynamic division determines the time when the scheduling decisions are made. In static case the decisions are made already in a compilation phase, whereas in dynamic the decisions are made when the jobs arrive. Optimal versus sub-optimal division tells, as its name says, whether the solution provided by the scheduling algorithm provides optimal or only 'close-enough-to-optimal' solutions. From this taxonomy we can see that scheduling and queueing theory are very close to each other and aim to the same goal, but still operate with a slightly different terms. We will explain the queueing theory model we used in our research in the next Section.

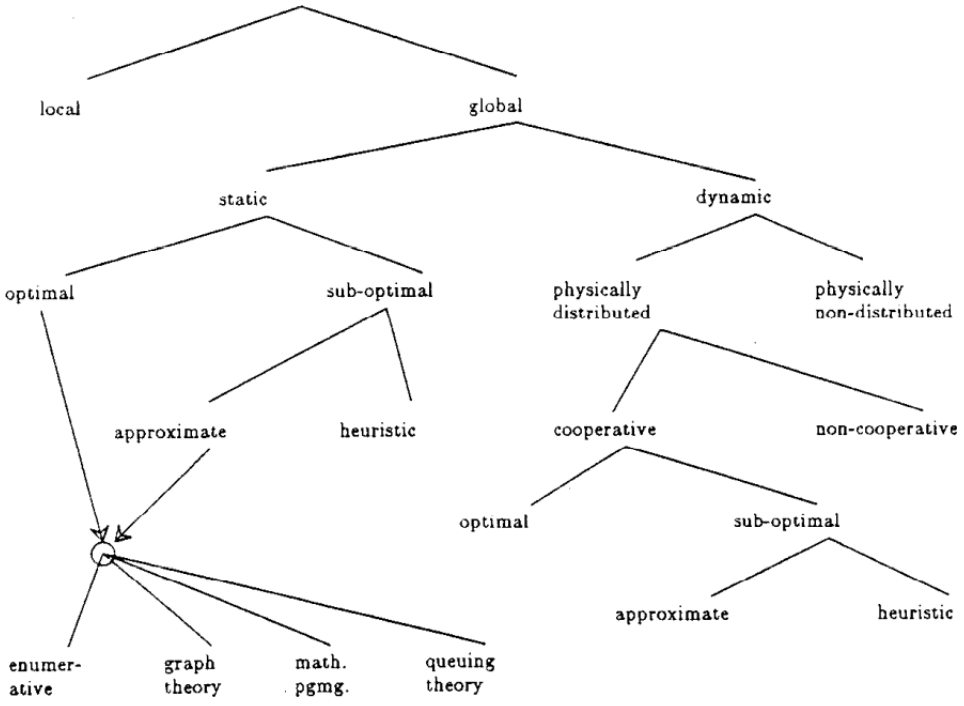

Figure 7.1: Taxonomy of different types of scheduling algorithms [23]

Third, we can express the problem as a pure optimization as was done in

[45]. In this approach described in more details in Chapter 4 the number of computing nodes in a cluster during the day could be represented as a time series, where on x-axis there is time and on y-axis there is number of servers. An appropriate limiting function is required to express how increase of servers affects electricity bill and similarly how decrease of servers and thus degraded quality of service affects profits gained from service. By substituting to the whole time series correct values, we can achieve the maximum for our profits.

Last, we can model the problem with control theory. In the such approach we are set a variable we want to control, which usually means keeping to equal to the specific value. The value is chosen such that if the system stays in the state represented by the value, savings related to the another variable are achieved (see Section 5.2).

7.2 Our queueing theory model

We want to obtain a queueing theory model that is as simple as possible but is expressive enough to provide us with information helping in our decision making such that energy savings could be achieved. Thus, we construct a queueing system in which jobs arrive to the system queue and based on the cluster status are either sent immediately to a server or forced to wait until there will be free space available on the server. So, in the model there is a waiting queue, responsible for buffering all incoming jobs, and a pool of parallelly connected processing queues, which accomplish the jobs arriving from the waiting queue (see Figure 7.2). Translating to the real setting, a waiting queue is a cluster scheduler, processing queues are cluster servers (computing nodes) and adding and removing processing queues means switching on and off cluster servers.

As we described in Section 7.1, the aim is to adjust the number of running servers to the current workload. This means that, ideally, a waiting queue exists only during workload intensity changes and otherwise it stays empty. This is not very radical assumption if we think what happens during workload intensity changes. If intensity decreases, there are more servers than are needed for the current workload, and arriving jobs are sent immediately to the servers. If intensity remains the same, the amount of servers is optimal to the current workload and all arriving jobs are again sent immediately to the servers. If then intensity grows, more servers are needed related to the workload and the arriving jobs are forced to wait in the waiting queue, but the system reacts to the situation by adding more servers. If we assume that our system reacts to the intensity increases fast enough and that the workload

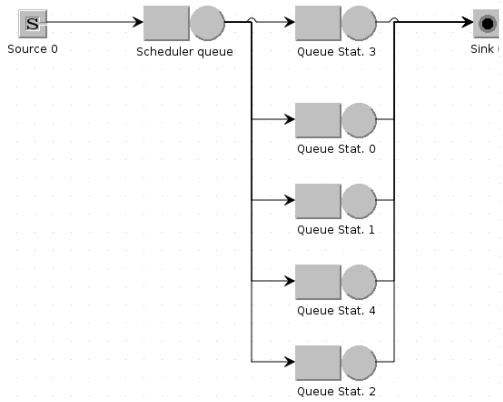

Figure 7.2: A queueing theory model representing a computing cluster (the diagram is generated by the JMT-tool [18])

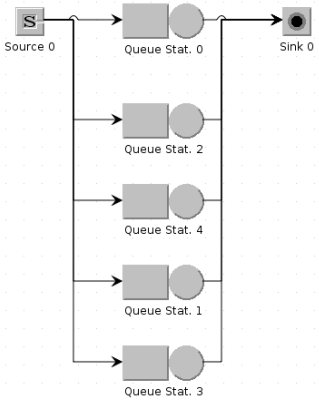

Figure 7.3: A simplified queueing theory model representing a computing cluster (the diagram is generated by the JMT-tool [18])

never grows larger then there possible servers to process it, the waiting queue stays most of the time empty. Thus we can represent our system with a simpler model, in which the waiting queue is deleted of convenience reasons (see Figure 7.3).

In queueing theory terminology our model is n $X/X/1$ queue with varying n. What probability distribution functions interarriving and serving time follow? This has to be decided based on analyzing the data. As we did in Chapter 6, the statistical tests should be performed. As we saw, the arriving jobs do not always conform to Poisson distribution. The main reason is that in computation clusters jobs are sometimes sent in bursts. Still, in the sufficiently large clusters the arriving jobs come approximately from Poisson distribution (interarrival times are exponentially distributed) with varying arriving rate and service time follows some general distribution. Thus we achieve n $M/G/1$ queue. Servers process all their jobs simultaneously, so we have process sharing (PS) queueing paradigm for processing queues (servers). As we saw in Section 5.1.8, $M/G/1 - PS$ queue behaves in the same way as $M/M/1 - FIFO$ queue, so we can freely use results obtained there.

A few words should be said about why our model is not an $M/G/K$-queue. An $M/G/K$-queue could be a tempting choice, here is a pair of reasons. First, we might want to abstract away what happens inside a single node

thus eliminating own queues from the nodes. Second, one might argue that the scheduler queue exists all the time, actually the solution could be implemented in a way that the queue always contains a few jobs. It is true that the abstraction of node's internal events would result in a simpler model. But the problem is that in real world setting there a non-zero probability of several jobs being on a node while no jobs on the other. On the other hand, such situation is totally impossible for the $M/G/K$-queue model [61]. Additionally, the implementations of our algorithm use the information of how many jobs there are currently on a node. This is because it is more natural to mark the node to be switched off which has the least amount of running jobs. For the second reason we can argue that the queue is non-empty for very short time periods and if there some jobs in it, they could be sent to computing nodes for execution immediately. If we want to accumulate more jobs to be executed at once, we come to trade-off solutions for quality of service and electricity cost balance, which we shortly describe in Section 10.1.

Now, we summarize what model we will use as well as its advantages and disadvantages. Our model is n $M/G/1-PS$ queue, where incoming traffic follows Poisson distribution with a varying rate and service time follows some general distribution (here we want to stress again that variability in incoming traffic is not controlled but the service time variability is our reaction to the changes in incoming traffic). The model makes a number of simplifying assumptions each of them is responsible for errors in model's predictions. Those assumptions are:

1. The system reacts to the workload intensity changes very rapidly, such that the waiting queue stays empty most of the time (see Section 5.1.10).

2. The system is never overloaded for long periods ($\rho < 1$), such that queueing theory equations are valid (see Section 5.1.10).

3. The jobs arrive approximately independently of each other and according to Poisson distribution. We discussed this in Section 6 how well these assumptions hold in our solution.

4. The cluster scheduler sending jobs to the servers preserves Poisson distribution. This means that the scheduler splits the incoming workload randomly between the servers.

7.3 Algorithmical solution

In this Section we describe the algorithm implementing the idea presented in the previous Section. We implemented three different algorithms called `queueing theory with averaging filter` (`QTwAF`), `queueing theory with exponential filter` (`QTwEF`) and `control theory with queue adjusting` (`CTwQA`). The pseudo-codes for these algorithms is presented in the Appendix.

`QTwAF` and `QTwEF` operate in a similar way. They measure the interarrival times from every two successive jobs coming to the cluster and calculate the estimate for the current arrival rate. The algorithms differ in how they calculate it. `QTwAF` takes an average from three successive interarrival times (the value of 3 was chosen for our preliminary test setting but a deeper research on the value optimality should be done). Theoretically such signal processing can be thought as filtering the input with FIR [53]. `QTwEF` scales the current and the past values in the way that the latest value is thought as more significant. The algorithm multiplies the latest interarrival time by factor 0.8 and the whole rest history by factor 0.2 and then sums them up (again the values of 0.8 and 0.2 were chosen for our preliminary test setting but a deeper research on the values optimality should be done). Such processing can be thought as filtering the input with IIR [53]. After that the estimate for the current arrival rate is put into the formula that determines the number on computing nodes required to maintain the predefined level of service whenever the arrival rate changes.

$$m = \frac{E[T]}{E[T] * \mu - 1} * \lambda \tag{7.1}$$

where m is number of required nodes, $E[T]$ is predefined fixed response time, which we want to keep constant, μ is the service rate of the node (or core if there are several cores per a node) and λ is the current arrival rate of jobs. We gave fixed values of 600 seconds and $1/90$ seconds for parameters $E[T]$ and μ respectively. This formula is derived from formula (5.9). When we made our calculations we multiplied m by the number of cores on a node, because in our test setting there were 4 processors (cores) on a computing node. Generally, m can be thought as an abstract processing unit, an entity which executes customers' jobs in $1/\mu$ time on average.

Theoretically the formula (7.1) should give the correct number of nodes but the problem is that we can only obtain an estimate of λ, not its correct value. If the estimate is too high (jobs are arriving often) compared to the real value,

we overbook computing resources resulting in an inefficient use of servers. On the other hand, if the estimate is too low (jobs are arriving rarely) compared to its real value, we may break the terms of agreement as our currently allocated computing resources are not enough to provide predefined quality of service. The algorithms `QTwAF` and `QTwEF` actually differ in how they try to estimate the arrival rate. But both algorithms do not see the effect of their estimation, they do not get information about the change in response time when the number of nodes is changed.

For this reason it would be reasonable to use a feedback system of the type described in Section 5.2. The most simple such system could be a PID-controller (not necessary using all its parts — proportional, integrator and derivator), which regulates the number of running nodes such that the length of the queue stays zero. If there are jobs waiting in the queue, the node could be switched on and if there are no jobs on some node, it could be switched off. Such controller might keep the queue in balance. We implemented this approach in the algorithm `CTwQA`.

`CTwQA` could provide the moderate solution to the problem but its shortage is that it does not know the dynamics of the system, which leads to slow reaction of the arrival rate changes. Better solution could be achieved if we combine two approaches — queueing and control theories, the number of nodes is set based on the arrival rate and calculated using the formula (7.1) but after that the fine-tuning is done by PID-controller, which observes the queue length. We did not implemented such hybrid algorithm because our cluster is small (three computing nodes), but we state that this approach might be optimal for big clusters.

7.4 Justification of the approach

Here we adopt the guidance described in [39] to sketch the systematic approach we used to tackle our problem. The guidance is the list of questions that all should be answered affirmatively if we want to confirm the reader about rigidity of our study. We present the most important of those questions below with our answers and possible references to the chapters we discuss the topic in more detail.

1. Is the system correctly defined and the goals clearly stated?

   Our system is a computing cluster, which receives jobs from user. The cluster is implemented as three slave nodes and a master node all connected via a network switch, the master receives jobs and passes them

to the slaves. The goals are 1) to develop an algorithm that achieves energy savings in the cluster computation 2) to investigate whether queueing theory is worth to be used as a model to solve the first goal.

2. Are the goals stated in an unbiased manner?

   We do not promote any specific theory or approach — the main focus is on the queueing theory but alternative methods are also presented.

3. Have all the steps of the analysis followed systematically?

   At least we have aimed to it. We selected all needed system parameters, factors, metrics and workload types.

4. Is the problem clearly understood before analyzing it?

   In Section 3.2 we described the important concept of energy proportionality — the idea to make a computing cluster use as much energy as it performs useful work. We also investigated what is the current state of computing clusters energy proportionality.

5. Are the performance metrics relevant for the problem?

   We chose the same metrics, which are used in the appropriate theory — response time, queue length, interarrival time between two successive jobs. Additionally we measure electricity consumption with an electricity meter. The only important missing metric, which is not included, is utilization due to the difficulty to measure it. Still, its absence is not that crucial because it can be approximated from the queue length and it basically tells us the same information we get from electricity consumption, in other words how energy efficient is our algorithm.

6. Is the workload correct for this problem?

   Ideally the workload should be the same as the one used by real users. Our users are physics researchers, which usually run simulations using Grid software. We cannot install such software into our test setting, so we used a commonly used benchmark, which actually loads the server in the similar way as the real workload. We also analyzed the average duration of the workload and chose the benchmark of the similar length.

7. Is the evaluation technique appropriate?

   We used a real measurement as our evaluation technique. Analytical evaluation can be easily performed as well by using equations derived in Section 5.1.

8. Is the list of parameters that affect performance complete?

   We included all parameters, which have been used in the similar studies: speed of the server CPU, bandwidth of the network, distribution of the jobs size and distribution of the workload intensity.

9. Have all parameters that affect performance been chosen as factors to be varied?

   As a factor we chose interarrival times. It varies in two dimensions, first it is randomly drawn from exponential distribution and second the parameter of the exponential distribution is also a random variable. We did not vary job duration, because the data we analyzed demonstrates strong deterministic nature — most of the jobs (90%) are short (< 90 seconds) and their variation inside this interval is small.

10. Is the experimental design efficient in terms of time and results?

    With a small number of runs (4 each of one day long) we can decide which algorithm is worth to investigate further as a tool for energy savings.

11. Would errors in the input cause an insignificant change in the results?

    Some parameter (such as whether a computing node is running or shut down) values are collected in such a way that they may contain a little delay. For example a node might be marked as shut down at a some time point although it is actually completing its last jobs. Still does not bring misleading results, because we compare the efficiency of competing algorithms in terms of electricity consumption, which is measured separately.

12. Have the outliers in the input or output been treated properly?

    We ignored very long jobs which were present in the real cluster logs and did not model them. In the output there were no outliers.

13. Have the future changes in the system and workload been modeled?

    No, but that was not the purpose of our study. We searched for a solution to the current energy consumption problem.

14. Has the variance of input been taken into account?

    No, but as wee saw from Section 6 after abandoning very long jobs, other fall in very small interval.

15. Is the analysis easy to explain?

    Queueing theory is state-of-the-art in telecommunication systems such that models based on it should be well understood.

Chapter 8

Implementation

8.1 Hardware settings

Our test environment was the following (depicted in Figure 8.1):

- A front-end computer with Sun Grid Engine 6.2u4 (SGE) [9] installed on it. SGE is a widely used batch system. Its scheduling interval was set to 5 seconds. The default scheduling algorithm of SGE was used. It allocates a newly arrived job to the computing node with the least workload (with the smallest number of running jobs).
- As the network configuration we had a 1 Gb D-link connecting the front end and three nodes. Linux operating system, kernel version 2.6.32-33, was installed on the nodes.
- 3 computing nodes with 4 core computers, which had Intel 2.4 GHz processors, 8 gigabytes of memory and a 217 GB hard disk. Each computing node has a number of job execution places. Place number tells how many jobs can be allocated to that node (in the terms of queueing theory place number corresponds to the queue buffer length). In our test setting place number of each node was set to 4 (the same as cores number). All jobs allocated to the same node are executed simultaneously in round robin fashion.
- The electricity was measured with the Watts Up Pro electricity meter.

We implemented our algorithms in Bash. It is a script that periodically checks the state of the cluster, i.e. observes whether the arrival intensity

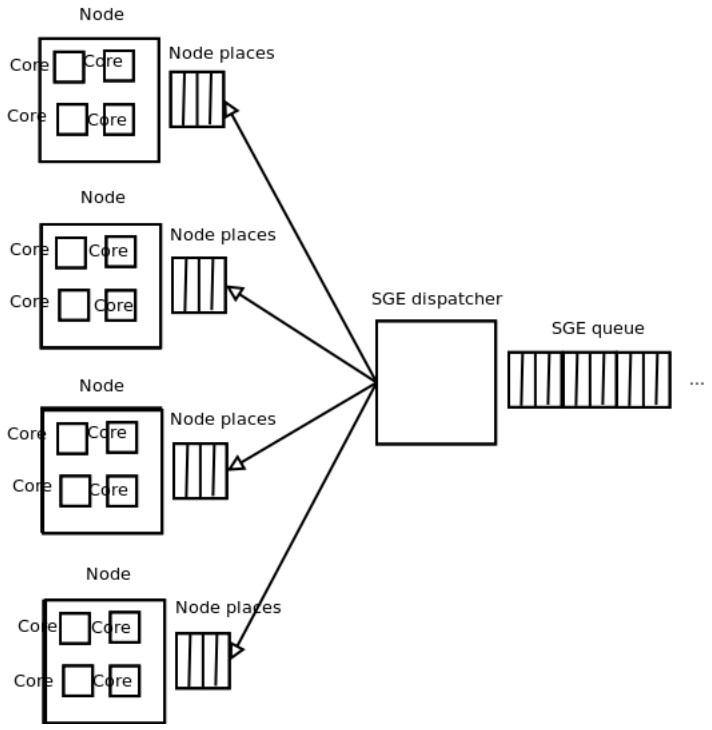

Figure 8.1: The hardware test setting

changed significantly such that the current number of nodes is not any more representative for it. Then based on the situation the algorithm performs appropriate actions i.e. adds or deletes a node from the cluster or does nothing. All three algorithms differ in the way how they estimate the current traffic.

The node deletion works in three phases: first, we delete the node from SGE queue such that no more jobs can be sent to it; second, we wait until the last job is completed; third, we suspend the node to the disk. There are two modes into which a node can be suspended [10]. The first is suspend to memory mode, during its suspend period the node consumes very little energy, but its wake up lasts only a few seconds. The second is suspend to disk mode, in which node consumes no energy at all but its wake up lasts more than one minute. The suspend to memory would be more preferable for us but unfortunately this mode is rarely supported on server type computers. Adding a node is performed with Linux `wakeonlan` [8] command, which sends a so called magic packet to the suspended node.

8.2 Workload settings, case 1

We generated different interarrival times from exponential distributions with varying λ. So, the generation of the interarrival time series happened as a generation of two random variables. First, we drew the random value for λ but in the way that it falls in the predefined interval. This is because we are limited in a number of computing nodes and cannot allow arbitrary high intensity, on the other hand we cannot allow too low intensity because then the nodes will idle most of their time and the test setting would not be fair. Second, we drew the random value from the exponential distribution with the parameter deduced in the first step. Thus we obtained the series of interarrival times. We want to stress here that this procedure results in Poisson distribution with a constant intensity. Our traffic simulator sends jobs to the cluster after the next interarrival time in the series have been elapsed.

We used lloops_64 benchmark from Roy Longbottom benchmark collection[1] as a test job. This benchmark was reported as a state-of-the-art test for scientific computing [39].

8.3 Workload settings, case 2

We also wanted to test our solution in an environment with arrivals distribution more close to the real world situation. That is why after running the test setting in case 1 (see Section 8.2) we chose the most promising algorithm (which turns out to be `CTwQA`) and tested it in the another test setting.

We wanted to use the workload from some real cluster and chose HIP's cluster for this purpose. Unfortunately it is significantly larger than our test cluster. That is why we extracted the data from HIP's cluster data log for only four nodes (because our cluster has also four nodes). Still the intervals between the consecutive arriving jobs were so long that without any preprocessing the test run would take too long. We scaled the data by multiplying each interarrival time by the constant.

Instead of previously used benchmark we used a so called dummy job. It just occupied a computing node for specified amount of time and did nothing. The executing times were also picked from the HIP's cluster data. They were scaled in the way such that their average should equal to 600 seconds. We used such setting because we wanted to simulate the effect of varying-

[1] http://homepage.virgin.net/roy.longbottom/index.htm

length-jobs and with standard benchmarks this would be difficult to achieve. The problem with the test setting in which jobs perform nothing but occupy a node for a specified amount of time, is that the energy consumption cannot be measured directly. But we can deduce it if we know what is an average consumption of a running node.

8.4 Workload settings, case 3

Our third test setting was the following, we generated time intervals of a random length but still longer than average job execution duration, which was 90 seconds. Then for each time interval we generated arrival intensity λ, which is constant during this interval. Then we generated interarrival times within each time interval from the exponential distribution with the parameter λ generated in the previous step. Thus we obtain randomly long time interval with randomly generated interarrival times inside each interval. After that we sent jobs to our cluster after each interarrival time.

We used a 'winner' algorithm `CTwQA` and lloops_64 benchmark as job simulators. Then we sent jobs by using the same interarrival times to the cluster without any algorithm and compared consumed electricity. We also measured response time of every job to compare how the optimization algorithm affects quality of service.

Chapter 9

Results

9.1 Results for the test setting 1

We performed the series of tests with hardware setting described in Section 8.1 which implements the algorithms described in Section 7.3 and the workload settings described in Section 8.2. The test run lasted approximately 19 hours (69168 seconds) and it was executed 3270 jobs. In the subsequent plots we can see the behavior of the algorithms in terms of their transient electricity consumption (see Figures 9.2, 9.5, 9.8 and 9.11), the length of the queue (see Figures 9.3, 9.6, 9.9 and 9.12) and the response time (see Figures 9.4, 9.7, 9.10 and 9.13). All results are summarized in Table 9.1. A reader might wonder the term length of the queue as we claimed in Section 7.2 that the queue will stay empty. This is true but it is important to remember first, that our model is only approximation and, second that although the queue should be empty, it occasionally contains some jobs, such that this effect raises the average from zero. Still, as we claimed the average queue length is very close to zero.

As we can see all three algorithms stayed in the service level agreement, which was set to 600 seconds. The best result in terms of whole electricity consumption is provided by `CTwQA`. `CTwQA` spent 5648 Watt hours which is 11.9 % less then in the setting without any algorithm (6413 Watt hours spent) with the average response time only 32.3 % greater then in the no-algorithm version (119.0 seconds against 90.0 seconds).

The algorithms based on 'pure' queueing theory approaches `QTwAF` and `QTwEF` are less energy-efficient than `CTwQA` but still outperform the setting without any algorithm with an acceptable degradation in quality of service. Interest-

Table 9.1: Table (The comparision of the algorithms)

	QTwAF	QTwEF	CTwQA	No algorithm
cumulative electricity consumption (Wh)	5829	6186	5648	6413
minimum transient electricity (W)	225	195	197	252
maximum transient electricity (W)	393	453	473	486
average number of jobs in service	3,983	4,084	3,990	4,216
average number of jobs in the queue	0,198	1,288	1,459	0,105
average response time (s)	92,7	117,8	119,0	90,0

ingly that `QTwAF` turned out to be more efficient than `QTwEF` in both electricity consumption and response time. Their difference was that `QTwAF` calculates an average from interarrival times and `QTwEF` calculates an exponential decay from them. The reason behind inefficiency might be that `QTwEF` is too sensitive to abrupt changes in incoming traffic such that it switches nodes on and off rapidly although the better strategy would be to adopt to the changes smoothly.

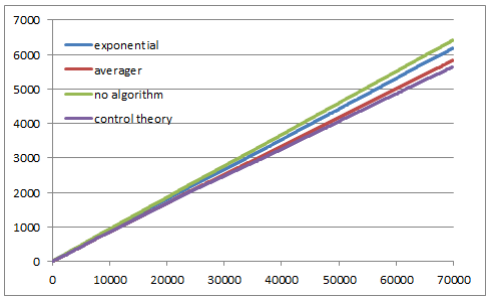

Figure 9.1: Comparision of the algorithms: cumulative electricity consumption in Watt hours is depicted as the function of time in seconds

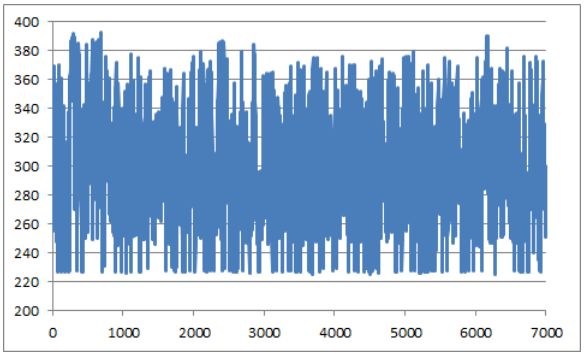

Figure 9.2: Queueing theory with averaging filter: transient electricity use in Watts plotted as a function of time in seconds

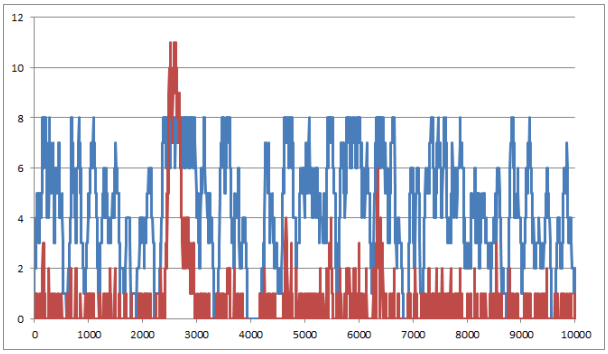

Figure 9.3: Queueing theory with averaging filter: (red) the length of the queue, (blue) the number of jobs in the system as a function of time in seconds

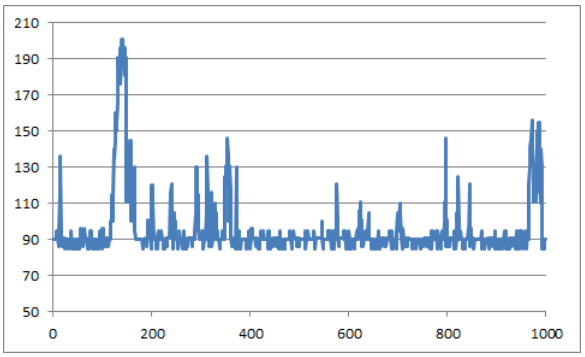

Figure 9.4: Queueing theory with averaging filter: response time in seconds as a function of time in seconds

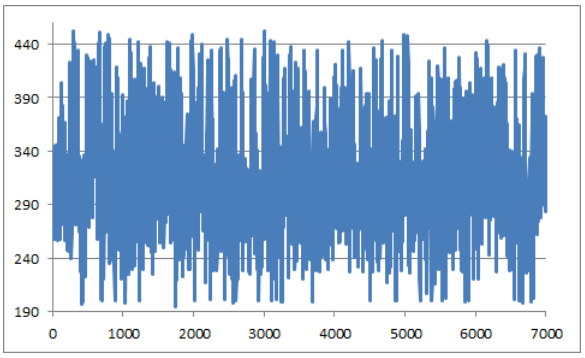

Figure 9.5: Queueing theory with exponential filter: transient electricity use in Watts plotted as a function of time in seconds

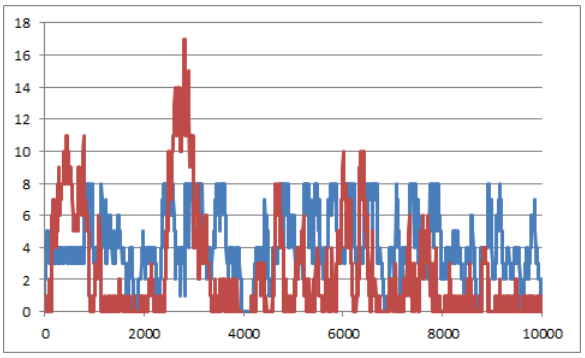

Figure 9.6: Queueing theory with exponential filter: (red) the length of the queue, (blue) the number of jobs in the system as a function of time in seconds

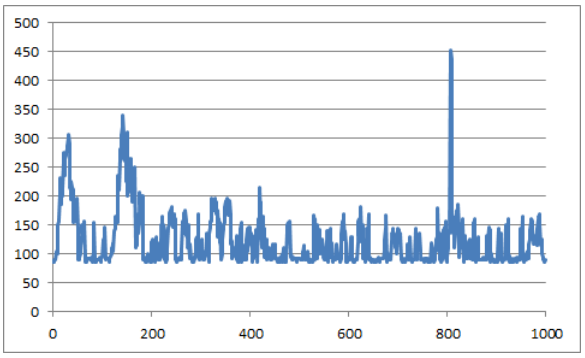

Figure 9.7: Queueing theory with exponential filter: response time in seconds as a function of time in seconds

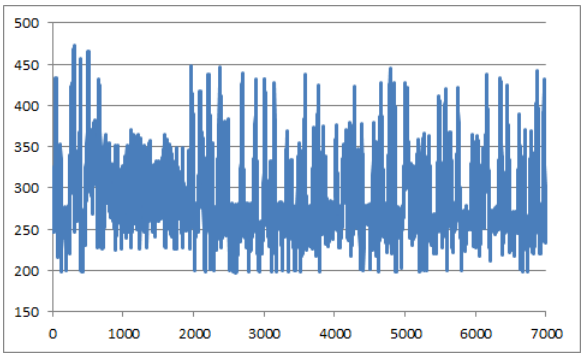

Figure 9.8: Control theory with queue adjusting: transient electricity use in Watts plotted as a function of time in seconds

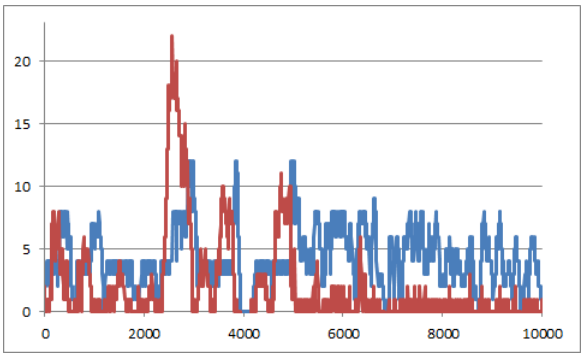

Figure 9.9: Control theory with queue adjusting: (red) the length of the queue, (blue) the number of jobs in the system as a function of time in seconds

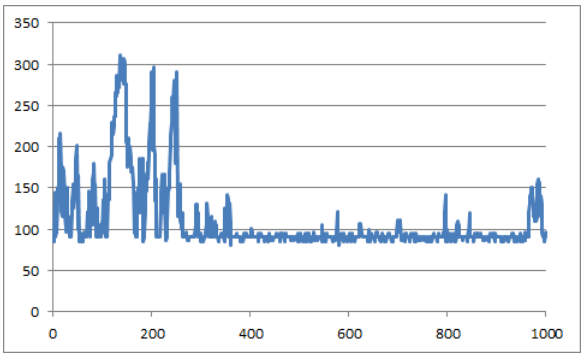

Figure 9.10: Control theory with queue adjusting: response time in seconds as a function of time in seconds

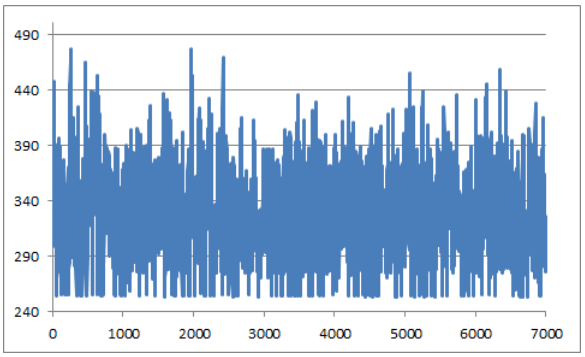

Figure 9.11: No algorithm: transient electricity use in Watts plotted as a function of time in seconds

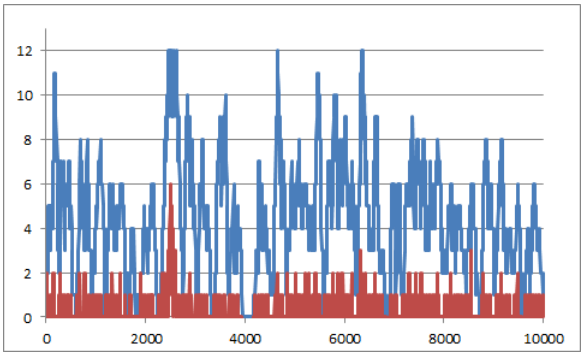

Figure 9.12: No algorithm: (red) the length of the queue, (blue) the number of jobs in the system as a function of time in seconds

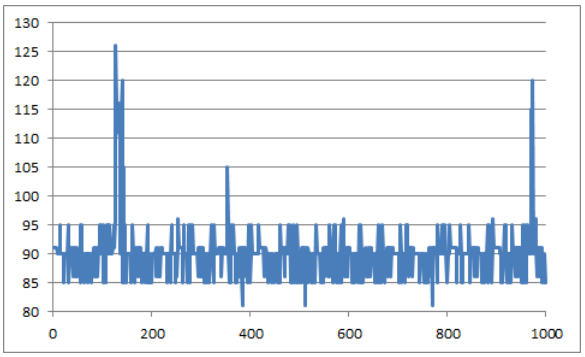

Figure 9.13: No algorithm: response time in seconds as a function of time in seconds

9.2 Results for the test setting 2

Then we run test for the winner algorithm — `CTwQA` with the same hardware setting (see Section 8.1) but with different workload settings, which are more close to real world case (see Section 8.3). The test run lasted approximately 18 hours (64438 seconds) and it was executed 900 jobs. In the subsequent plots we can see the behavior of the algorithm in terms of its queue length (see Figures 9.14), the number of nodes (see Figures 9.15) and the response time (see Figures 9.16).

The average response time for HIP's cluster data after scaling was 335.8 seconds and the average response time of jobs ran in our test was 349.0, which is only 3 % longer. The average number of jobs in the service was 0.559 and in the queue 0.250. We calculated the theoretical amount of energy saved during this time — 18 hours by approximating how much electricity would be consumed if all three nodes would be running (we assumed that a running computing node consumes some predefined amount of electricity). According to our calculations we should have achieved 35 % savings in electricity consumption during such short time period.

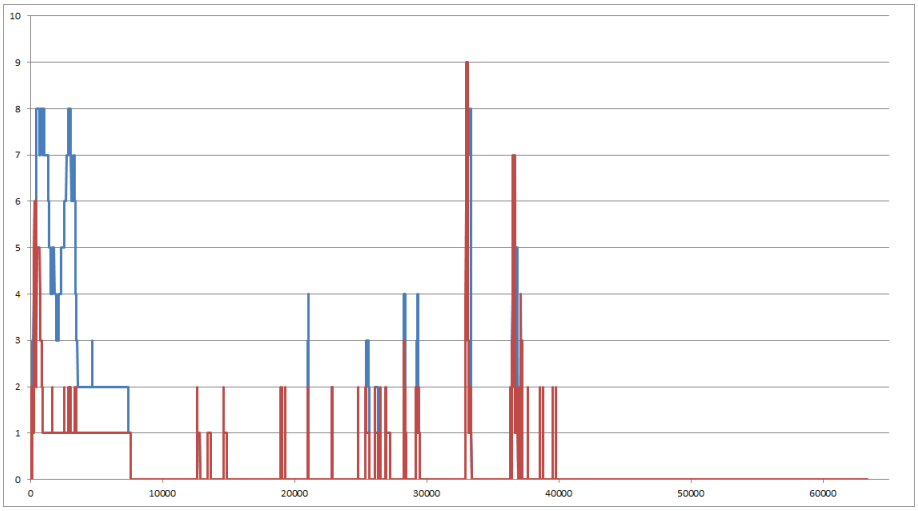

Figure 9.14: Control theory with queue adjusting, another test setting: (red) the length of the queue, (blue) the number of jobs in the system as a function of time in seconds

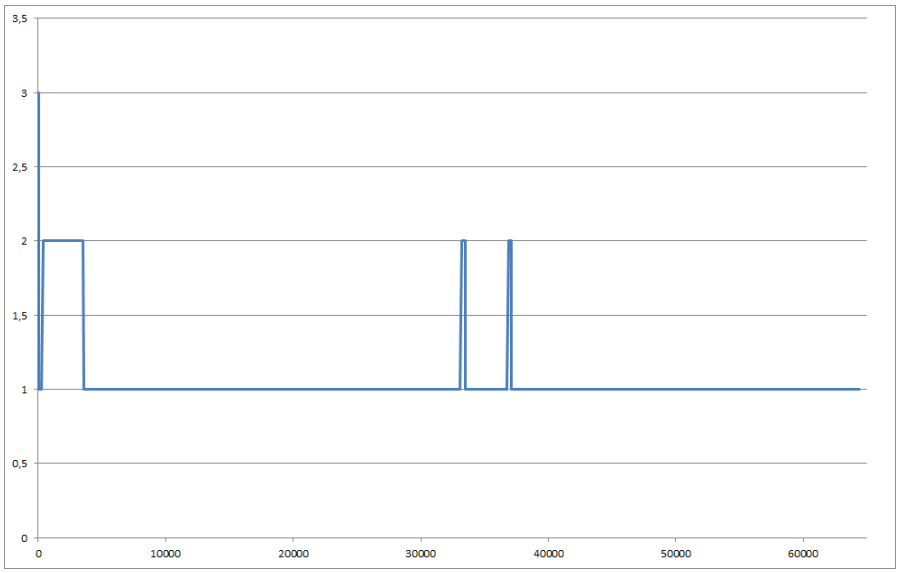

Figure 9.15: Control theory with queue adjusting, another test setting: the number of running computing nodes as a function of time in seconds

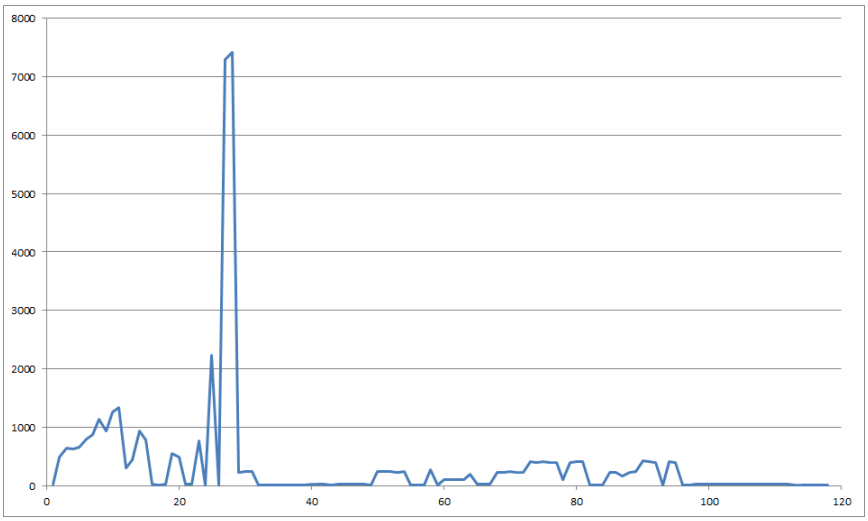

Figure 9.16: Control theory with queue adjusting, another test setting: the jobs response time in seconds

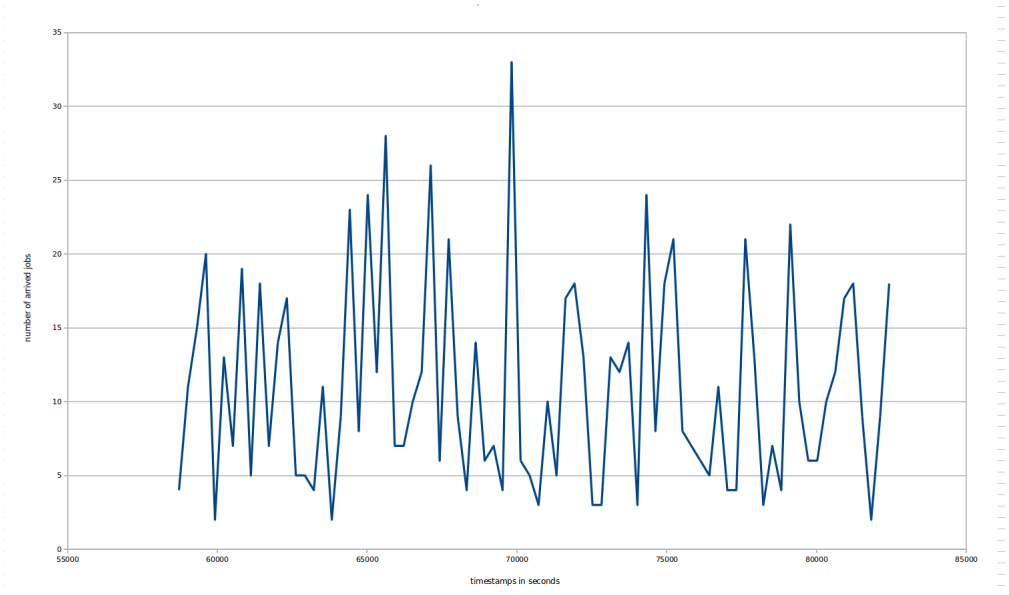

Figure 9.17: Randomly generated arrival intensity expressed as a number of jobs arrived during five minutes interval (time unit is a second)

9.3 Results for the test setting 3

In the third test setting, described in Section 8.4, we compared the measured electricity savings gained by `CTwQA` compared to the setting without any algorithm. The test lasted approximately 8 hours and we gained 27.7 % in electricity savings (only 873 Wh:s spent versus 1207 Wh:s) with only 35.5 % longer response times (139.6 s versus 90.1 of the average response time).

In Figure 9.17 is shown how arrival intensity varied through the test period. In Figure 9.18 is shown how the number of running nodes varied through the test period. In Figure 9.19 is shown current power consumption of `CTwQA`.

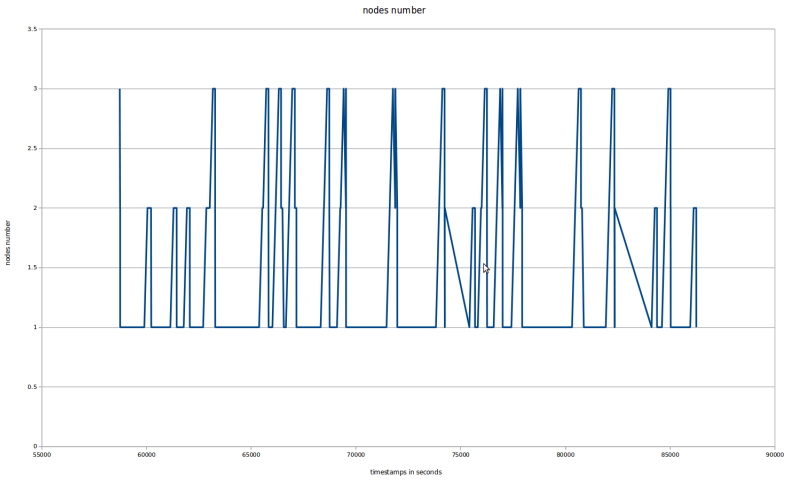

Figure 9.18: The number running nodes as a function of time in seconds

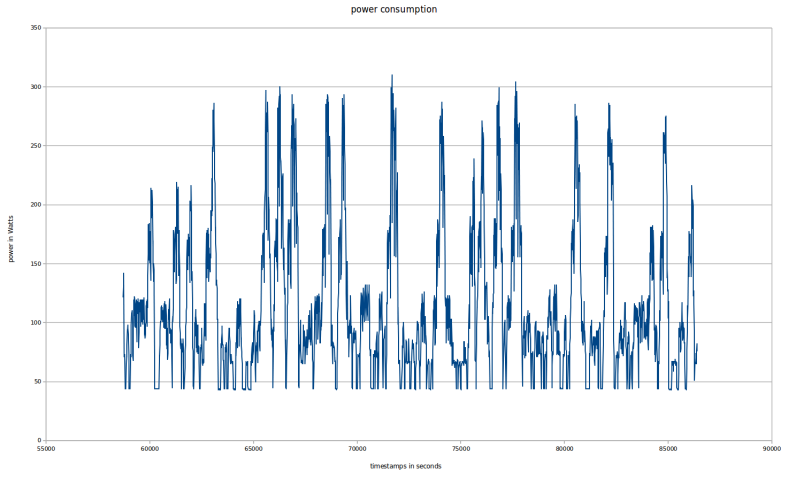

Figure 9.19: Control theory with queue adjusting, third test setting: transient electricity use in Watts plotted as a function of time in seconds

Chapter 10

Conclusions and further work

As we saw from Table 9.1 and the figures in Chapter 9, all algorithms outperform the setting without an algorithm with minor effects on quality of service. All three solutions provide significant energy savings with moderate degradation in quality of service. We strongly suggest to adopt the solutions we have proposed. As a starting point for small size clusters (less then 10 nodes) we suggest to consider `CTwQA`. For a bigger size clusters we suggest to develop a solution based on mixed approach of queueing theory and control theory. In such an algorithm the appropriate number of running nodes in a cluster is first estimated based on queueing theory equations and then fine-tuned with control theory PID-controller.

10.1 Improvements and further work

A Master's Thesis is a project which is limited in time and space such that many analyses and verifications have been omitted and interesting nuances are left for the further research. Still, this thesis comprises the complete entity and all suggestions should be understood as suggestions not as criticism.

First, as we pointed out in Chapter 6 the real world data from clusters do not follow Poisson distribution. Thus we were forced to use M/G/1-model as an approximation (see Figure 5.7). In this case it would be nice to run simulations to prove that the model used gives the correct predictions. In such a simulation the real world data would be fed to the system, the quality of service is then measured and compared to the predictions obtained from the analytical M/G/1-model. If the difference between the measurements and the predictions is neglectable, it is acceptable to use the approximation.

We did not performed this because of the lack of time but the results about energy savings gained with our solution are enough alone. For additional research a very good background information about simulation can be found in [42].

Second, in our solution we provided a reader only with a road map — what should be improved (servers utilization) and how it could be done (our three algorithms). But obviously the solution is not optimal. Although our algorithms can be applied as working solutions, if the solution would be applied in a real industrial datacenter cluster, we strongly recommend to fine-tune them for the concrete case. As we described earlier, we claim that a hybrid solution with both queueing theory and control theory would be optimal. By using the terminology described in Section 5.2, the queueing theory equations would work as feedforward-blocks that enable quick responses to the changes in the arrival rate. Thus the cluster can simultaneously switch on or off many computing nodes. After that the control theory based PID-controller can fine-tune the final amount of the nodes. Thus the queueing theory provides quick response and control theory fine-tuning and correction of the errors made in the estimation of the arrival rate.

Third, very close to our solution's further improvements is the problem regarding how to decide to which server a newly arrived job should be allocated. In the literature this is called dispatching problem. In our solution we used the default dispatching algorithm of SGE, which tries to balance the load between running nodes and thus sends a new job to the node with the least current workload. Still, intuitively a better results could be achieved if a newly arrived job is sent to the most loaded node. More considerations about possible dispatching solutions can be found in [54] and [22].

Fourth, better implementation techniques can be used. As we described in Chapter 8, when the node is 'switched off' it is put to the suspend-to-disk-state. It consumes almost no energy but waking the node up can take a pair of minutes. The suspend-to-memory-state would be more preferable but unfortunately it is not supported on the most of server nodes. The second technology improvement concerns the use of live migration [26]. Virtual machines could be installed on the computing nodes and thus when some node was marked to be switched off, instead of waiting until it process its jobs to the end, the jobs could be transferred to an another node on the fly.

Finally, more investigations can be performed about prediction of the future traffic and the trade-off between savings in electricity costs gained by restricting the service and revenue losses caused by the customers abandoning your service because of degraded quality. Queueing theory contains the tools

for expressing the relation between those two in the terms of so called utility function. Interested reader can explore more in [36].

10.2 Summary

In our study we have given a comprehensive view of green IT. We have described a number of energy-efficient solutions, metrics expressing the goodness of those solutions and projects associated with energy-efficiency in ICT. We have mainly focused on the energy-efficient solutions in datacenter servers. We have also described the bounds of possible new solutions — the requirement to maintain reasonable quality of service for users. We have presented our solution based on queueing theory. A detailed theoretical background was provided for deep understanding of our approach such that ideally no other literature is needed for understanding and implementing our solution. We justified our solution with the extensive data analysis. Also we tended to follow as much as possible the guidelines of the state-of-the-art performance analysis. We have implemented our solution and tested it on a real hardware. The test setting was close to the real world use-case. The preliminary results show that our solution achieves notable savings in electricity. We suggest to investigate our solution in a more detail and give a few suggestions for the further research.

Bibliography

[1] Cern webpages. http://public.web.cern.ch/public/en/About/History54-en.html.

[2] Cern webpages. http://public.web.cern.ch/public/en/About/Mission-en.html.

[3] Cern webpages. http://public.web.cern.ch/public/en/LHC/LHC-en.html.

[4] Cern webpages. http://public.web.cern.ch/public/en/About/History08-en.html.

[5] Cern webpages. http://public.web.cern.ch/public/en/Science/Higgs-en.html.

[6] Google datacenters. http://www.google.com/about/datacenters/.

[7] The green grid datacenter power efficiency metrics: PUE and DCiE. http://www.thegreengrid.org/sitecore/content/Global/Content/white-papers/The-Green-Grid-Data-Center-Power-Efficiency-Metrics-PUE-and-DCiE.aspx.

[8] *Magic Packet Technology*, November 1995.

[9] *BEGINNER'S GUIDE TO SUN GRID ENGINE 6.2 Installation and ConfigurationWhite*, 2009.

[10] *Advanced Configuration and Power Interface Specification*, April 2010.

[11] ICT sustainability: The global benchmark 2011.

[12] Albers, S. Energy-efficient algorithms. *Commun. ACM 53* (May 2010), 86–96.

[13] Andersson, M., Cao, J., Kihl, M., and Nyberg, C. Performance modeling of an Apache web server with bursty arrival traffic. In *IC'03: proceedings of the international conference on internet computing* (2005), CSREA Press.

[14] Aoyama, Yukiya; Nakano, J. *RS/6000 SP: Practical MPI Programming*, 1999.

[15] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. Above the clouds: A Berkeley view of cloud computing. Tech. rep., University of California at Berkeley, February 2009.

[16] Barroso, L. A., and Hölzle, U. The Case for Energy-Proportional Computing. *Computer 40*, 12 (Dec. 2007), 33–37.

[17] Barroso, L. A., and Hölzle, U. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2009.

[18] Bertoli, M., Casale, G., and Serazzi, G. Jmt: performance engineering tools for system modeling. *SIGMETRICS Perform. Eval. Rev. 36*, 4 (2009), 10–15.

[19] Bhat, U. N. *An Introduction to Queueing Theory: Modeling and Analysis in Applications*. Statistics for Industry and Technology. Springer, Dordrecht, 2008.

[20] Brown, L., Gans, N., Mandelbaum, A., Sakov, A., Shen, H., Zeltyn, S., and Zhao, L. Statistical analysis of a telephone call center: a queueing science perspective. Tech. Rep. 03-12, Wharton Financial Institutions Center, Nov. 2002.

[21] Cao, J., Andersson, M., Nyberg, C., and Kihl, M. Web server performance modeling using an M/G/1/K*PS queue. In *10th International Conference on Telecommunications (ICT'03)* (2003), pp. 1501–1506.

[22] Casalicchio, E., and Colajanni, M. A client-aware dispatching algorithm for web clusters providing multiple services. In *Proceedings of the 10th international conference on World Wide Web* (2001), ACM, pp. 535–544.

[23] Casavant, T. L., and Kuhl, J. G. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Softw. Eng. 14* (February 1988), 141–154.

[24] Choi, K., Soma, R., and Pedram, M. Dynamic voltage and frequency scaling based on workload decomposition. In *Proceedings of the 2004 international symposium on Low power electronics and design* (New York, NY, USA, 2004), ISLPED '04, ACM, pp. 174–179.

[25] Christensen, K. Green networks: Reducing the energy consumption of networks. `http://www.csee.usf.edu/~christen/energy/koreaTalk10.pdf`.

[26] Clark, C., Fraser, K., and Steven, H. Live Migration of Virtual Machines. In *Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2005), pp. 273–286.

[27] Corporation, I. *Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor - White Paper*, 2004.

[28] David Kaneda, Brad Jacobson, P. R. Plug load reduction: The next big hurdle for net zero energy building design. *ACEEE* (2010), 120–130.

[29] Dean, J., and Ghemawat, S. Mapreduce: Simplified data processing on large clusters. *Commun. ACM 51* (Jan. 2008), 107–113.

[30] Fan, X., Weber, W. D., and Barroso, L. A. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th annual international symposium on Computer architecture* (New York, NY, USA, 2007), ISCA '07, ACM, pp. 13–23.

[31] Foster, I., Zhao, Y., Raicu, I., and Lu, S. Cloud Computing and Grid Computing 360-Degree Compared. In *2008 Grid Computing Environments Workshop* (Nov. 2008), IEEE, pp. 1–10.

[32] Gandhi, A., Harchol-Balter, M., Das, R., and Lefurgy, C. Optimal power allocation in server farms. In *SIGMETRICS/Performance* (2009), J. R. Douceur, A. G. Greenberg, T. Bonald, and J. Nieh, Eds., ACM, pp. 157–168.

[33] Govindan, S., Sivasubramaniam, A., and Urgaonkar, B. Benefits and limitations of tapping into stored energy for datacenters. *SIGARCH Comput. Archit. News 39*, 3 (June 2011), 341–352.

[34] Green IT Promotion Council, J. Concept of new metrics for data center energy efficiency introduction of datacenter performance per energy. Tech. rep., 2010.

[35] Grid, T. G. The Green Grid Data Center Power Efficiency Metrics: PUE and DCiE. Tech. rep., 2007.

[36] Hall, R. W. *Queueing methods: For services and manufacturing.* Prentice Hall, 1991.

[37] Hamilton, J. Cloud economics of scale. `http://channel9.msdn.com/events/MIX/MIX10/EX01`.

[38] Hennessy, J. L., and Patterson, D. A. *Computer Architecture: A Quantitative Approach (The Morgan Kaufmann Series in Computer Architecture and Design).* Morgan Kaufmann, May 2002.

[39] Jain, R. *The Art of Computer Systems Performance Analysis: Techniques for experimental design, measurement, simulation, and modeling.* Wiley, 1991.

[40] Jayant Baliga, K. H., and Tucker, R. S. Energy consumption of the Internet. *COIN-ACOFT 24-27* (2007).

[41] Kesselman, C., and Foster, I. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers, Nov. 1998.

[42] Law, A. M., and Kelton, D. W. *Simulation Modelling and Analysis.* McGraw-Hill Education - Europe, 2000.

[43] Levine, W. S. *Control system fundamentals.* CRC Press, 1999.

[44] Limited, D. Understanding the cloud computing stack: SaaS, PaaS, IaaS. *Diversity Unlimited* (2011), 1–17.

[45] Lin, M., Wierman, A., Andrew, L. L. H., and Thereska, E. Dynamic right-sizing for power-proportional data centers. In *Proc. IEEE INFOCOM* (Shanghai, China, 10-15 Apr 2011), pp. 1098–1106.

[46] Lincoln, D. *The Quantum Frontier: The Large Hadron Collider.* 2009.

[47] Liu, X., Heo, J., and Sha, L. Modeling 3-tiered web applications. In *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 307–310.

[48] Lydia Parziale, David T. Britt, C. D. J. F. W. L. C. M. N. R. *TCP/IP Tutorial and Technical Overview*, 2006.

[49] Massey, F. J. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American Statistical Association 46*, 253 (1951), 68–78.

[50] Meisner, D., Gold, B. T., and Wenisch, T. F. PowerNap: Eliminating server idle power. *SIGPLAN Not. 44* (Mar. 2009), 205–216.

[51] Meisner, D., Sadler, C. M., Barroso, L. A., Weber, W.-D., and Wenisch, T. F. Power management of online data-intensive services. In *ISCA* (2011), R. Iyer, Q. Yang, and A. González, Eds., ACM, pp. 319–330.

[52] Milton, J. S., and Arnold, J. C. *Introduction to probability and statistics: Principles and applications for engineering and the computing sciences*. McGraw-Hill, 1995.

[53] Mitra, S. K. *Digital signal processing: a computer-based approach*. McGraw-Hill, 2001.

[54] Niemi, T., and Hameri, A.-P. Memory-based scheduling of scientific computing clusters. *The Journal of Supercomputing* (2011), 1–25.

[55] P. Reviriego, K. Christensen, J. R., and Maestro, J. A. An initial evaluation of energy efficient ethernet. *IEEE Communications Letters 15* (May 2011), 578–580.

[56] Poess, M., and Nambiar, R. O. Energy cost, the key challenge of today's data centers: A power consumption analysis of TPC-C results. *Proc. VLDB Endow. 1*, 2 (Aug. 2008), 1229–1240.

[57] Poess, M., Nambiar, R. O., Vaid, K., Stephens, J. M., Huppler, K., and Haines, E. Energy benchmarks: A detailed analysis. In *e-Energy* (2010), H. de Meer, S. Singh, and T. Braun, Eds., ACM, pp. 131–140.

[58] Sator, S. Managing office plug loads. *Energy Manager's Quartery, ESCD Newsletter* (June 2008), 1–11.

[59] Webb, M., and Al., E. Smart 2020: Enabling the low carbon economy in the information age. *The Climate Group London* (2008).

[60] Weinman, J. Cloudonomics. `http://cloudonomics.com/`.

[61] Willig, A. *A Short Introduction to Queueing Theory*, 1999.

Appendix

Below is listed the pseudo-code of the algorithms. The main scheduling algorithm `ControlCluster()` controls a cluster queue. It periodically checks the incoming traffic — `new_intensity := CalculateIntensity(Algorithm);` and if it changed, the algorithm switches off on on a computing node. The place, where three algorithms differ from each other, is how they evaluate an arrival rate. `QTwAF` is listed in `CalculateIntensity(Average)` and calculates the average of the last N successive arrival times and uses the approximation to deduce the required nodes number — `CalculateNeededNodesNumber`. For that it of course needs the mean response time and the mean delay. `QTwEF` listed in `CalculateIntensity(Exponential)` operates in the similar way except that it evaluates the incoming traffic by weighting differently the very last arrival and the remaining $N - 1$ arrivals. `CTwQA` listed in `CalculateIntensity(Queue control)` does not look at arrivals at all but approximates the arrival rate from the length of the cluster queue.

```
ControlCluster() {

  // the initial number of running nodes
  running_nodes_number := INITIAL_NODES_NUM;
  current_intensity := 1;
  // 0 measns that the intensity decreased,
  // 1 that it keeps the same
  // 2 that it increased

  do forever {
      sleep for N seconds;
      new_intensity := CalculateIntensity(Algorithm);

      // the intensity increased
      if (new_intensity > current_intensity &&
          running_nodes_number < MAX_NODES_NUMBER) {
```

```
          WakeupNode();
          running_nodes_number++;
      }

      // the intensity decreased
      else if (new_intensity < current_intensity &&
               running_nodes_number > MIN_NODES_NUMBER) {
      node_to_be_removed := ChooseNodeWithTheLeastWorkload();
      RemoveNodeFromQueue(node_to_be_removed);
      ShutDownNodeWhenEmpty(node_to_be_removed);
      running_nodes_number--;
      }

  }
}

CalculateIntensity(Average) {

  last_arrivals[] := GetNLastArrivals();
  average_lambda_inverse :=
           CalculateAverageLambdaInverse(last_arrivals[]);
  needed_nodes_number :=
           CalculateNeededNodesNumber(average_lambda_inverse,
                                      MEAN_DELAY,
                                      MEAN_SERVICE_TIME);

  // the intensity increased
  if (running_nodes_number < needed_nodes_number)
    return 2;
  // the intensity decreased
  else if (running_nodes_number > needed_nodes_number)
    return 0;
  // the intensity stays the same
  return 1;
}

CalculateIntensity(Exponential) {

  last_exponential_lambda_inverse := GetLastLambdaInverse();
  last_arrival := GetLastArrival();
  exponential_lambda_inverse :=
```

```
            COEFFICIENT1 * last_arrival +
            COEFFICIENT2 * last_exponential_lambda_inverse

  needed_nodes_number :=
            CalculateNeededNodesNumber(average_lambda_inverse,
                                       MEAN_DELAY,
                                       MEAN_SERVICE_TIME);

  // the intensity increased
  if (running_nodes_number < needed_nodes_number)
    return 2;
  // the intensity decreased
  else if (running_nodes_number > needed_nodes_number)
    return 0;
  // the intensity stays the same
  return 1;

}

CalculateIntensity(Queue control) {

  waiting_threshold := SLOT_NUMBER + 1;
  // SLOT_NUMBER  is the number of jobs allowed to be
  // executed on a single computing node

  running_threshold := running_nodes_number * SLOT_NUMBER - 1;

  waiting_jobs_number := GetNumberOfJobsInQueue();
  running_jobs_number := GetNumberOfJobsInService();


  if ((waiting_jobs_number - waiting_threshold) <
      (running_threshold - running_jobs_number)) {
    // the intensity decreased
    if (running_jobs_number < running_threshold) {
      return 0
    }
    // the intensity increased
    if (waiting_jobs_number > waiting_threshold) {
      return 2
    }
```

```
  }
  // the intensity stays the same
  return 0;
}
```