Riku Lääkkölä

Data Center Degrowth – an Experimental Study

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 28.4.2015

Thesis supervisor:

Prof. Jukka Manner

Thesis advisor:

Lic.Sc. (Tech.) Markus Peuhkuri

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Riku Lääkkölä

Title: Data Center Degrowth – an Experimental Study

Date: 28.4.2015 Language: English Number of pages: 8+56

Department of Communications and Networking

Professorship: Networking technology Code: S-38

Supervisor: Prof. Jukka Manner

Advisor: Lic.Sc. (Tech.) Markus Peuhkuri

Due to the massive increase in demand for cloud services, and popularity of mobile devices, the number of data centers and the amount of energy consumed by data centers is constantly growing. IT hardware does become more energy efficient according to Koomey's law, but the power proportinality of, e.g., servers and network switches is still quite poor.

In this thesis, the trends in data center energy consumption and efficiency is closely examined, and some alternative methods for reversing the trend of data center power consumption are considered. In the experimental phase, a pilot data center is built and a basic web service architecture is designed on top of it in order to study how optimization and allocation of resources affect the quality of experience of the service. The results from the measurements indicate that, for this particular system, a surprisingly small amount of application processing server instances was required for near optimal quality of experience.

Keywords: Data Center, Degrowth, Quality of Experience, Energy Efficiency, Koomey's Law

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Riku Lääkkölä

Työn nimi: Palvelinkeskusten kasvun purku – kokeellinen tutkielma

Päivämäärä: 28.4.2015 Kieli: Englanti Sivumäärä: 8+56

Tietoliikenne- ja Tietoverkkotekniikan laitos

Professuuri: Tietoverkkotekniikka Koodi: S-38

Valvoja: Prof. Jukka Manner

Ohjaaja: TkL Markus Peuhkuri

Pilvipalveluiden kysynnän ja mobiililaitteiden suosion valtavan kasvun vuoksi palvelinkeskusten lukumäärä ja energiankulutus on maailmanlaajuisesti jatkuvassa kasvussa. Vaikka IT-laitteiston energiatehokkuus paraneekin jatkuvasti Koomeyn lain mukaisesti, on esimerkiksi palvelinten ja verkkokytkinten tehonkulutus edelleen varsin epädynaamista.
Tässä opinnäytetyössä tutkitaan palvelinkeskusten energiankulutuksen ja energiatehokkuuden kehityssuuntia ja selvitetään vaihtoisia toimintatapoja yllä mainitun kehityssuunnan kääntämiseksi. Kokeellisesta osiota varten rakennettiin pilottipalvelinkeskus ja sinne suunniteltiin tavanomainen verkkopalveluarkkitehtuuri, jotta olisi mahdollista tutkia kuinka optimointi ja resurssien allokointi vaikuttavat palvelun käyttökokemukseen. Mittausten tulokset osoittivat, että tässä kyseisessä järjestelmässä yllättävän pieni määrä sovelluspalvelimia riittää lähes optimaalisen käyttökokemuksen tarjoamiseen.

Avainsanat: Palvelinkeskus, Degrowth, Quality of Experience, Energiatehokkuus, Koomeyn Laki

Preface

You are now reading the product of a fairly lenghty journey that started with a part-time employment at the Department of Communications and Networking, where my task was to attempt to hack a bunch of cheap wireless power sensors to give out a better time resolution. Now, after a very fun and educational couple of years of installing servers, disk systems and switches without manuals at ComNet, and an even more fun and somewhat educational N years of being a part of the teekkari community at TKK and later Aalto University, my work here is done.

I would like to express my deepest gratitude to my instructor, Markus Peuhkuri, and my supervisor, prof. Jukka Manner, for great guidance throughout this process, and for their incredible flexibility and promptness despite my own inability to get things done in a timely manner. I would also like to thank all the great people at ComNet for providing an inspiring work environment, especially Pasi, Samuli, Carles, Sebastien, Ada, Antti, Arttu and Jukkis.

For all the merriment I got to experience during my life as a student, I wish to thank SIK, HTMK08, HTMK09, SIKH09, IE09, SIKH11, FTMK11, BioPojat, Nasku Salttu and his brethren, and all the wonderful people I have met during my years in Otaniemi. For more, and sometimes less, healthy pastime activities I want to thank the awesome folk at Yliopiston Taido. And last, I want to thank my family for supporting me in whatever I aspire, and Kaisu for making my life brighter every day.

Espoo, 17.4.2015

Riku Lääkkölä

Contents

Symbols and abbreviations

Symbols

E	Energy
k	Boltzmann's constant
L_l	Landauer's limit, the minimal amount of energy that must be consumed by erasing one bit of information
P	Power, i.e., the rate at which energy is being consumed.
T_l	Page load time, i.e., the time it takes for a web page along with its contents to arrive at the client after the request

Abbreviations

CA	Certificate Authority
CDN	Content Delivery Network
CMS	Content Management System
CRAC	Computer room air-conditioning
DB	Database
DCEE	Data Center Energy Efficiency
FC	Fibre Channel
HTTP	HyperText Transport Protocol
IaaS	Infrastructure-as-a-Service
OS	Operating System
P2P	Peer-to-Peer
PaaS	Platform-as-a-Service
PUE	Power Usage Efficiency
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory

SAAS	Software-as-a-Service
SAN	Storage Area Network
SLA	Service Level Agreement
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URI	Uniform Resource Identifier
VM	Virtual Machine
WSGI	Web Server Gateway Interface

1 Introduction

During the first decades of computing and data processing, working on a computer meant, that you sat in front of a terminal and there was also a data center in the vicinity, where the mainframe did all the heavy processing. The advent of the personal computer shifted some of the load from the data center to the machine you were working on. Today, however, we seem to have come full circle, and in what some would call an ideal situation you would only have a device with reasonable capabilities in displaying graphical content and networking, and the data processing and storage would happen on a remote (possibly, but not ideally, at a distance of hundreds of kilometers) data center.

Whether or not this is a positive evolution from an energy efficiency viewpoint, is not self-evident. However, mobile devices are, according to some predictions, most notably by Cisco [1], fast becoming the dominant group of client entities in the Internet. In order to maintain battery lifetime and affordable prices, mobile devices need to manage without extensive processing resources and storage. This directs the pressure towards remote servers and data centers.

Centralized computing and data storage is also almost mandatory in some aspects of modern society, as during recent years we have witnessed enormous growth in popularity for social media services such as Facebook and twitter, where a distributed architecture might not even be entirely feasible. Although, there are projects such as Diaspora, that attempt to bring people social networks in a decentralized manner [2]. Also, as people wish to have their data and entertainment available wherever they go, without having to set up their own file servers, services like Dropbox, Netflix, Spotify and the likes are probably here to stay.

This means that an increasing amount of aspects in our lives is becoming dependent on network connections and data stored in data centers. Even before the golden age of cloud services, the amount of energy consumed by data centers has increased rapidly. During the period from 2000 to 2005 power used by data centers doubled worldwide, from a 0.5% to a 1% fraction of the total worldwide power consumption [3]. From 2005 to 2010 this growth was approximately 56% instead of doubling [4].

The aforementioned facts combined present us with a problem: How do we keep data center energy consumption in control now, when sources of energy are even scarcer and demand for fast and reliable cloud services keeps increasing? Luckily

power consumed in data centers may also result in a better total energy efficiency when distributed consumption becomes more centralized and thus enables better control over the consumption and efficiency of the power hungry operations. This effect in the case of video recording and streaming was recently studied by Peuhkuri et al [5]. However, global energy consumption is also constantly increasing [6], which means that energy efficiency in data centers will play an even larger role in society.

There are of course multiple levels of energy effiviency related to data center operations, the outermost, and most self-evident, of which is the ratio of the total consumption of electricity to the electricity consumed by the actual "payload" hardware, i.e., the IT equipment such as servers, routers, switches, etc. This metric called PUE (Power Usage Efficiency). Here a large factor in the efficiency is the method of cooling selected for the computer rooms. As an example, traditional uncontained server racks on an elevated floor such as in Figure 1 is far inferior to closed rack cabinets, like in Figure 2, with in row cooling in terms of energy efficiency.

The next level is the efficiency at which the actual IT hardware architecture operates. There are differences in the amount of computations a server can carry out with one kWh of energy between servers, especially a new one and an old one. Also pieces of networking equipment have major differences in energy efficiency. For instance, an ethernet switch may activate an entire block of, e.g., 12 interfaces when only one link is up in the block, whereas other models may regulate power for each interface separately. The most sophisticated devices can even reduce consumption when a link is up, but the amount of traffic is low [8], but in the worst case the switch may consume a nearly constant amount of power whenever it is turned on, regardless of traffic or the number active interfaces. The same problem exists in server hardware as well. [9]

When the infrastructure and hardware resources are optimized to a realistic extent, what is left is making sure that the software architecture of the system is exploiting the resources as efficiently as possible. This means that the operating systems and different software components need to be carefully chosen to best suit the service in question. For instance, the choice of database software may make a great difference in terms of performance.

In addition to reducing the total amount of energy consumed, it is important to remember, that the electric grid as a whole benefits most when power consumption of the peak hours is reduced. This means that, since peak hours of usage in public data center powered services often coincide with the global peak consumption within

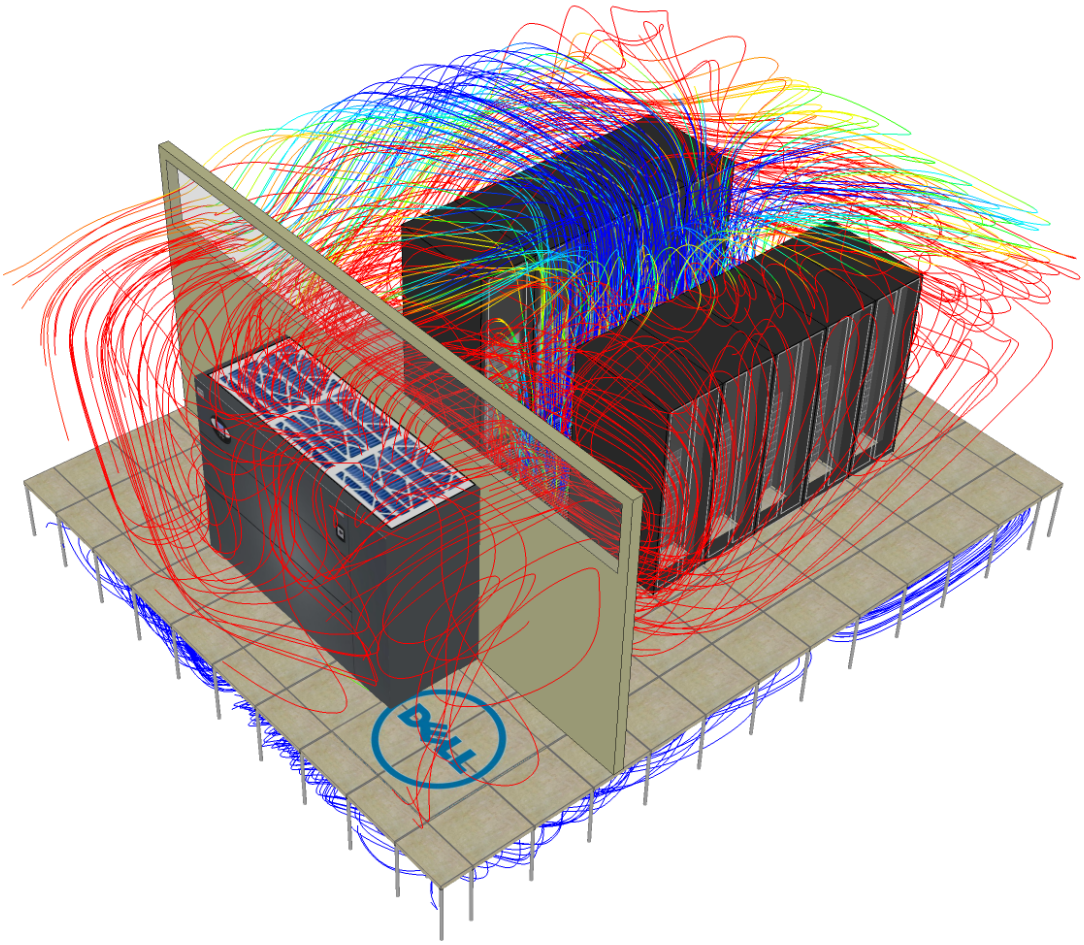

Figure 1: Uncontained computer room cooling solution with elevated floor tiles. Image source: [7]

the time zone, it is not sufficient to optimize power consumption during off-peak hours, but efficient usage of resources during peak traffic must also be pursued.

In this thesis, the aim is to face the problem of optimal resource usage in a slightly backwards manner: How much is the user-perceived QoE (Quality of Experience) affected, when service capacity is reduced in order to save power and resources? The effects of software optimization are also studied, and the effects of Moore's Law [10] and Koomey's Law [11], which predict, respectively, the price and the energy efficiency of computing hardware, are taken into account.

The main point of interest in the experimental part of this thesis is the optimization of resource allocation between different software components of a distributed web service system. It hardly makes sense to provision a database server with more processor cores if the bottleneck is somewhere else in the system.

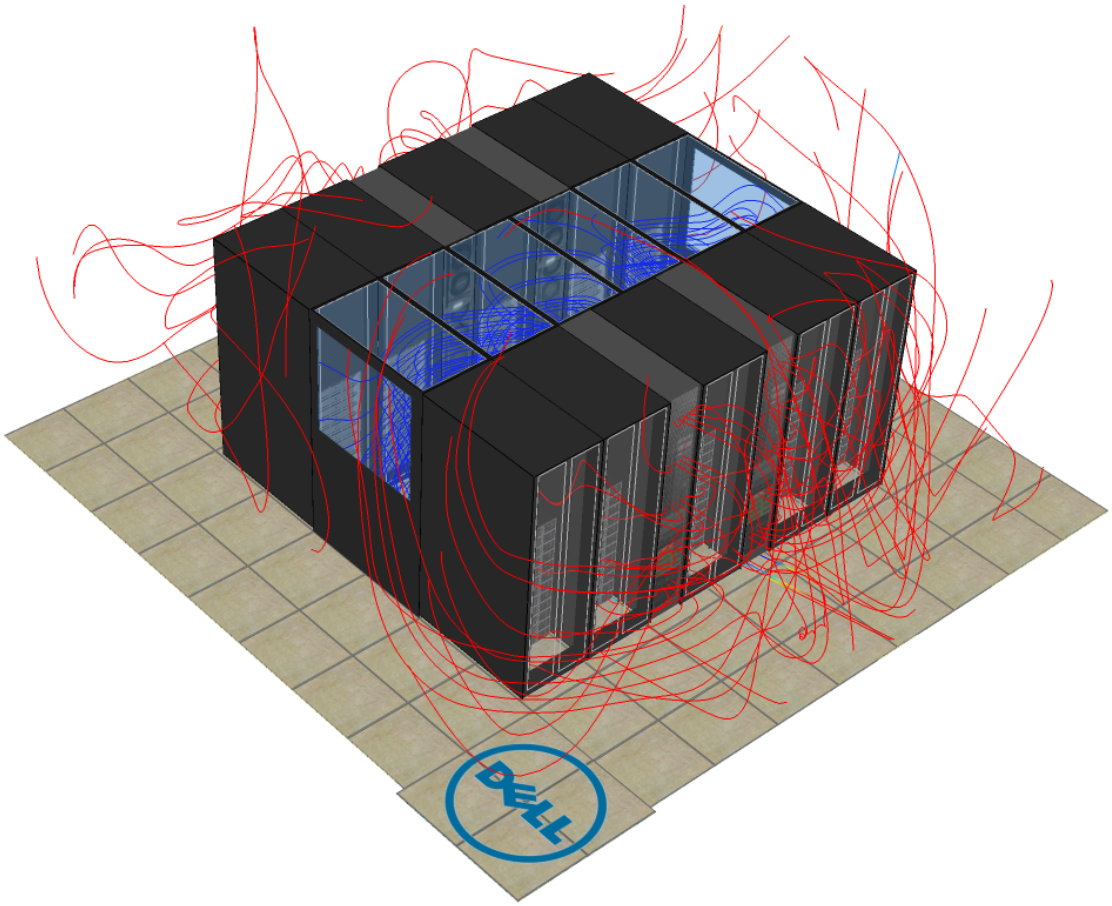

Figure 2: Computer room cooling solution with closed rack cabinets and in row cooling. Image source: [7]

During the experimental phase, the author was responsible for, along with others, installing hardware into racks, setting up and debugging the base systems, designing the software architecture and implementing the service that was tested, setting up and configuring the measurement equipment and carrying out the measurements.

After these efforts, data about the system's performance when a capacity parameter was varied was gathered. The results indicate, that the number of application server instances in a distributed web application setup affects the QoE of the service as expected, and that the maximum useful amount of resources allocated to application servers is relatively low, especially in the case of HTTPS traffic. These results provide additional insight into benchmarking web system performance.

In Chapter 2, data centers and trends in their use and energy efficiency are covered alongside with an analysis of natural limits in computing energy efficiency and met-

rics for QoE and energy efficiency. The web technologies that are most relevant to this thesis are also discussed in some detail. Chapter 3 describes the design of the measurement setup used in the experimental part of the work both from hardware and software point of view in detail, and Chapter 4 showcases the results that were accumulated after the experimental phase. Chapter 5 gives a brief summary of the subjects covered in this thesis.

2 Background and Theory

In this chapter the basics of data centers in general are covered along with some current trends in DCEE (Data Center Energy Efficiency) and illustrative examples of popular cloud based services. Moreover, the effect of Koomey's Law and its limitations are analyzed, and some factors regarding data center location are reviewed. Choosing the correct metrics for both QoE and energy consumption and efficiency is also vital, so these matters are covered in this.

2.1 An Introduction to Data Centers

A data center is a dedicated facility where computing systems and networking, data storage and other associated equipment are physically located, maintained and operated. In the early days of computing, most computer hardware had to reside in a data center like environment, as the power and cooling requirements were quite different from the ones of today's personal computers. In this section, the services provided by data centers are introduced and basic data center infrastructure is examined.

The purpose of a data center is, naturally, to provide services to the entity operating the data center itself or to external clients. Services provided by data centers currently might be roughly divided in the following categories:

- Data storage
- Data processing
- Data transmission

These services are contemporarily usually sold as something which might be called "*Something*-as-a-Service", but essentially the payload operations that are carried out by the data center hardware are one of the above. For example, Amazon Web Services provides, among other things, *IaaS* (Infrastructure-as-a-Service), i.e., virtual server instances, storage volumes and connections to the Internet, which the customer can control via a web interface, but the actual hardware and physical system architecture is asbracted away from the user.

Businesses can then purchase these virtual resources either with constant or *elastic* (scaled to demand) capacity, to cater for, e.g., their private cloud needs or to set

up their SaaS (Software-as-a-Service) or PaaS (Platform-as-a-Service) that they in turn sell to their customers.

In order to provide these kind of services, a data center requires computing servers, storage media and networking equipment. There is a large amount of hardware options in these categories for data centers, and it is up to the maintainer of the data center to choose the most suitable options and set them up properly, according to the service they are providing. The Telecommunications Industry Association provide an extensive set of recommendations in their Standard *TIA-942* [12] for this particular purpose. The standard also includes requirements for four different availability tiers for data center equipment, i.e., criteria that a hardware installation and its surroundings must meet in order to be categorized in a certain tier.

At the heart of the data center are the computer rooms that retain the device racks holding the server and storage equipment. It is usually a closed room with elevated floor tiles and abundant cable channels. The elevated floor tiles enable hiding cabling easily and more importantly, keeping cold air separated from hot air in places where there are no device fan inlets that require the cold air.

The actual IT harware resides in open racks or closed rack cabinets inside the computer room. A rack usually contains an Ethernet switch at the top and servers or storage devices below the switch. Cold air is brought to the fans of the devices by using grated tiles in front of the racks. The air heated by the devices is then usually directed to a CRAC, which cools the air and pushes it back under the floor. Closed rack cabinets may also be separately liquid cooled. There are different options for laying out the racks inside the computer room, as can be seen from Figures 1 and 2.

2.2 Energy and Efficiency

In this section, an overview of the trends in data center energy consumption and efficiency is given, and some theoretical limits are analyzed. Energy efficiency of data center cooling and effects of geographical location are also briefly covered.

2.2.1 Requirement for Energy Efficiency

Cisco Systems estimates, that in 2016 global IP traffic will surpass the one zettabyte per year threshold [13]. Further, the Cisco Visual Networking Index [1] forecasts,

that between 2013 and 2018 video traffic over the Internet will be tripled and web and other data traffic will be doubled in the private consumer segment, from 17 to 62 and from 5.5 to 13 petabytes per month, respectively. This evolution is also presented in Figure 3.

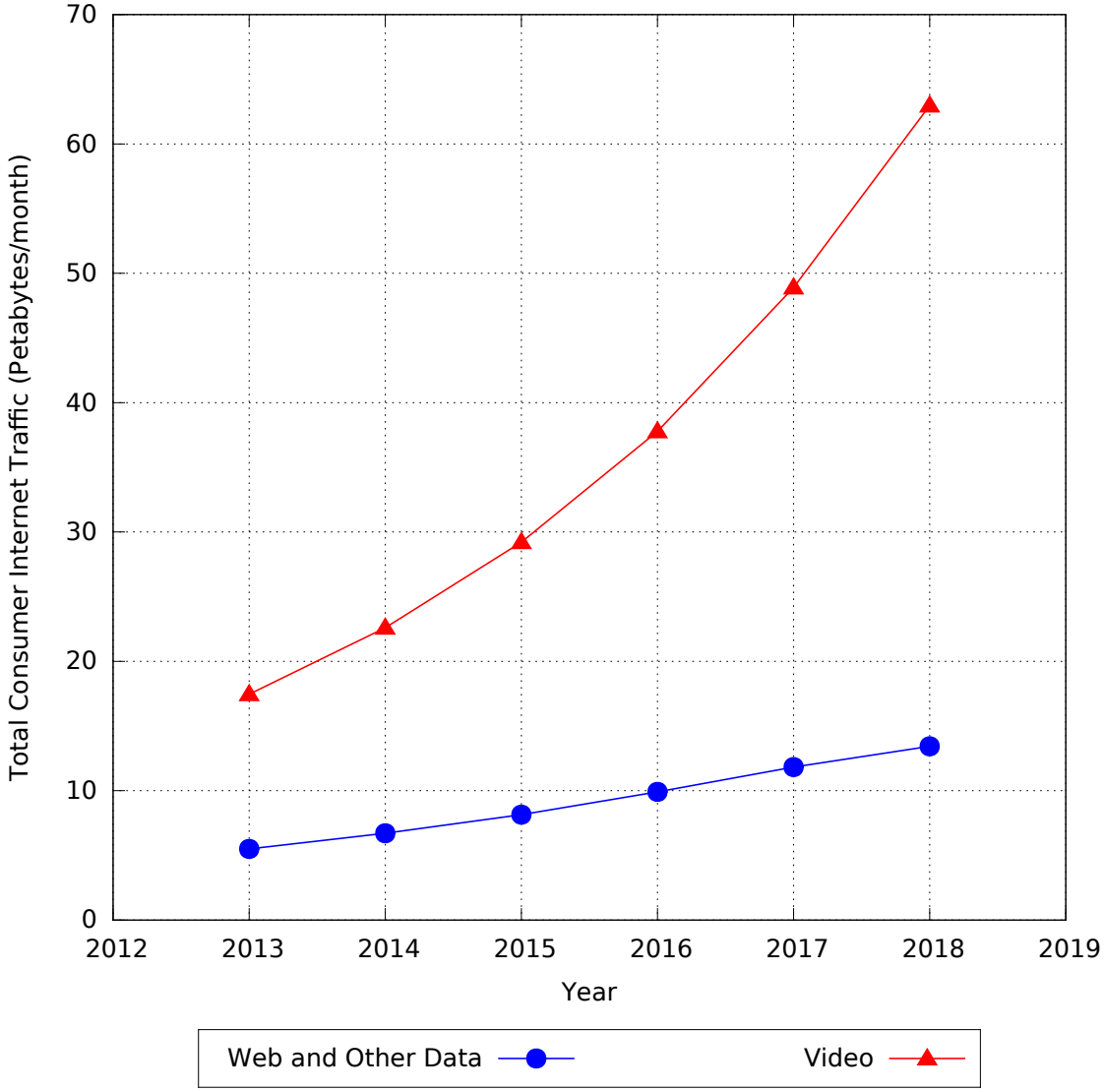

Figure 3: Cisco VNI forecast for consumer IP traffic, including video and web traffic [1].

Cisco also predicts that by 2018 the number of devices connected to an IP network will be almost double the number of people in the world, and that broadband connection speeds will become three times faster, from 16 to 42 Mbps. Most traffic will also originate from mobile or wireless devices. [1]

The overall effects of the increase in especially video traffic are slightly ameliorated by the projected increase in the share of traffic that is provided by CDNs, which

shortens the distance that content needs to travel through the network. This reduces the stress on, e.g., intercontinental links and central routers. CDNs themselves of course employ servers and other standard data center hardware, but as they are limited to serving static content, they can utilize a different set of optimizations compard to generic data center services. As CDNs cannot serve dynamic web content, the requirement for more efficient data center services due to the increase in web traffic remains for a large part a concern.

Also, the sheer number of servers operated by Amazon Web Services alone is difficult to grasp: They are operating in 11 regions, which comprise of a total of 28 availability zones that can each contain several data centers. Each of these data centers house more than 50 000 servers, and larger ones even over 80 000 servers [14]. This means that even by the most conservative estimate possible, this single corporation is running at least 1 400 000 server machines, and the number is constantly growing and with every new machine started, comes an increase in power consumption. This consumption divided by the number of operations the machine can carry out in a time unit is luckily smaller for newer hardware. This is dicatated by Koomey's law, which will be covered in Section 2.2.3.

2.2.2 Current Trends and State of the Art in Data Center Energy Efficiency

Transmitting video is a highly resource intensive task that requires a large capacity of all the aforementioned data center services. Video streaming services are becoming the largest generators of network traffic at a fast pace, and in 2011 Netflix was actually the largest source of Internet traffic in the United States with a 29.7% share of total traffic [15]. Additionally, many network operators in Finland have started offering so called "time shifting" services (e.g. [16] and [17]) to broadband customers. These services take care of recording programmes in the DVB-network remotely, and the customer does not need a set top box with storage capability. As discussed in Chapter 1, these kind of services should reduce the total energy consumption by centralization [5] but they are, however, often facing problems with copyright authorities, at least in Finland.

The requirement for more data center capacity of these types of streaming services may be relieved by utilizing P2P (Peer-to-Peer) techonologies. However, at this point, the only widely used service to do so seems to be the music streaming service

Spotify, which applies a P2P overlay network onto its desktop clients. [18] This approach is more thoroughly presented in Section 2.4.2. A future trend that is making P2P harder to utilize is the increasing share of mobile devices as recipients of multimedia traffic on the Internet. [1] These trends are covered in Section 2.2.1.

As managing computer systems is tedious and expensive, many businesses opt to outsource their data center services to specialized providers. These providers often make contracts that include SLAs (Service Level Agreements) with their customers, which then encourage the providers to ensure that hardware capacity is sufficient at all times. This in turn results to inefficient use of hardware resources and energy. Additinonally earlier, when running purely physical server instances was the only viable option, trying to reduce operational expenses by switching off entire servers and using dynamic voltage control was fairly cumbersome. [19]

A prominent remedy for the aforementioned problem is virtualization. Virtualization in this context means, that the OS that is interfacing directly with the server hardware (the “host” or “hypervisor”) does not run the actual services, but provides means to create containers ("VMs" or “guests”) that emulate hardware on which an additional OS and any service may be run. The guest OS communicates with the actual hardware through the hypervisor. This effectively enables a server to be split into several virtual servers with shared hardware resources. This kind of concept of virtualization dates back even to the 1960s. Hypervisor capabilities are provided by many vendors for multiple host OS platforms, and prominent products for generel purpose servers include free software such as Xen and KVM, and proprietary software like VMWare and Microsoft Virtual Server. [20]

The major advantage of virtualization is the possibility to divide server resources into more fine-grained virtual instances that are logically completely separated from each other. This combined with the ability to move these virtual instances from one physical host to another, even without needing to reboot the instance, enables the data center service provider to offer redundant, distributed and flexible services with a minimal amount of hardware resources. Energy consumption can also be minimized, as during off-peak one physical host can handle more VMs, and migration strategies that leave excess hardware free to be powered off may be applied [21]. This also mitigates the often bad proportional efficiency of server hardware [9].

The drawback of virtualization is of course overhead caused by running multiple operating systems to accomplish one task. This is however far outweighed by the benefits, at least judging by the growth of the virtual service providing industry.

For example estimates put Amazon Cloud Services' revenues for the year 2014 at $5 billion, a 58% growth from the previous year's estimated $3.1 billion [22], and the number of customers at Digital Ocean grew from approximately 1000 to 140000 during the year 2013 [23].

Recently, efforts to reduce the performance penalty of VMs have been taken in the form of projects such as LXC, a type of Linux Container [24], that enables separating processes in a Linux host without launching additional kernels. Also Docker [25] utilizes the same Linux kernel features as LXC, but is aimed more specifically at packaging a distributed system with its dependencies in a form that can be used in development, test and production environments alike. [26] Some even consider general-purpose operating systems ill-suited for data center usage, and are developing a new operating system concept that is designed for maximizing efficiency in virtualized services. [27]

2.2.3 Koomey's Law and Landauer's Principle

Koomey's law [11] predicts, that the energy efficiency of computing hardware, i.e., the number of computations carried out per joule of energy dissipated, doubles every 1.57 years. This law is derived from historical data of performance and power usage details of computers from the year 1946 forward. This data is also illustrated in Figure 4.

One theoretical lower limit for the energy efficiency of computing systems dictated by Landauer's principle [28] is yet to be reached. The Landauer lower limit for energy consumed in erasing one bit of information can be expressed in the following manner:

$$L_l = kT \cdot \ln 2, \tag{1}$$

where k is the Boltzmann constant ($\approx 1.38 \cdot 10^{-23} \frac{J}{K}$) and T is the temperature of the system in kelvins. The value of Landauer's limit in room temperature (25°C) is $2.85375 \cdot 10^{-21} J$, which is a far smaller amount of energy than computers today require – Figure 4 suggests, that in 2010 energy consumed by one computation was something between $3.6 \cdot 10^{-9}$ and $3.6 \cdot 10^{-10}$ joules. When the previous definition of Koomey's law is inverted, it can said that the energy consumed by one computation, on average, is halved every 1.57 years. The number of required "iterations" of

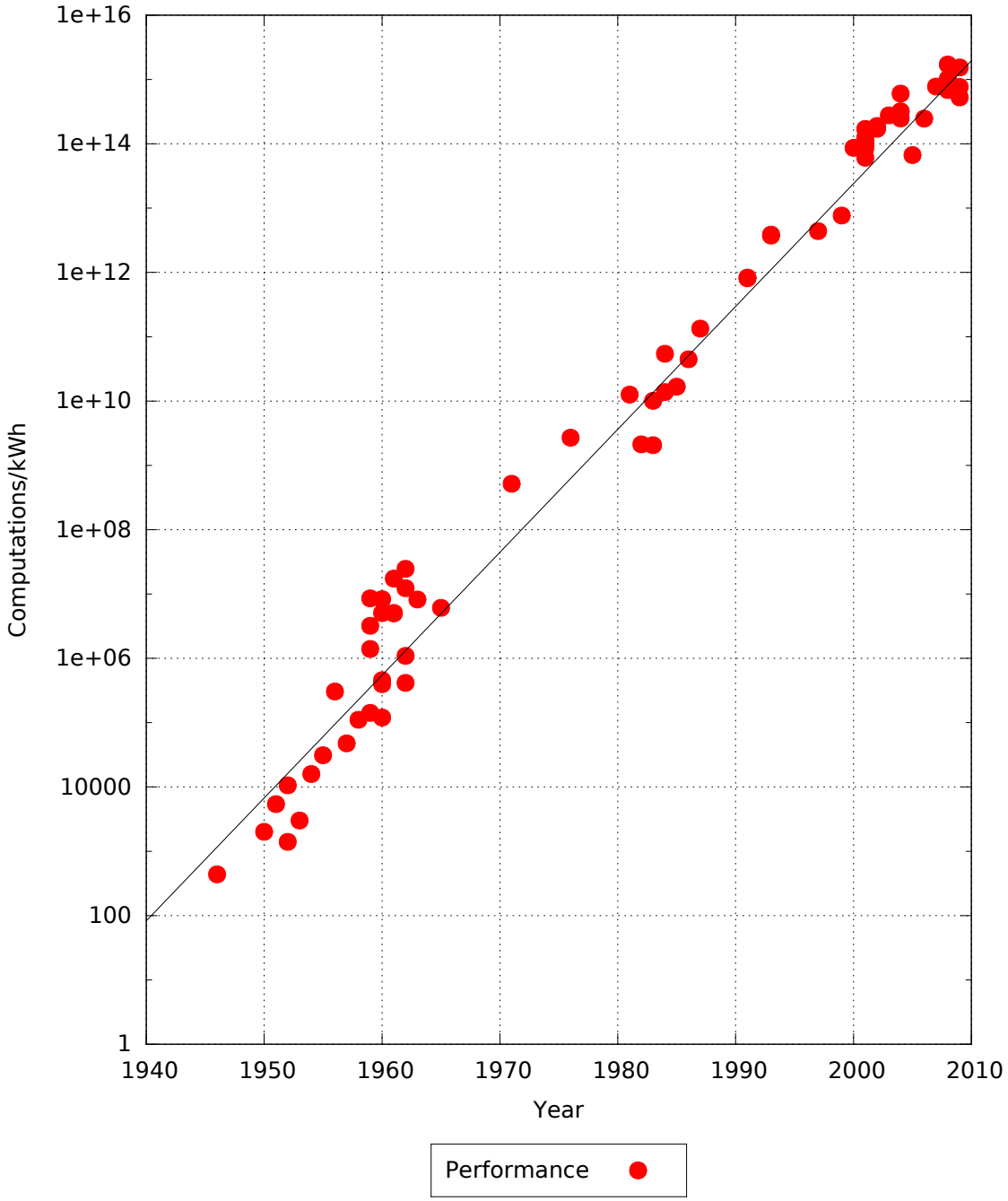

Figure 4: Historical data of energy efficiency of computing. The data points include both personal computers and industrial supercomputers. Data is gathered by Koomey et al. [11]

Koomey's law needed to reach the Landauer limit can then be calculated from the folloving formulation:

$$\left(\frac{1}{2}\right)^n \cdot E_{current} = E_{Landauer}$$
$$\iff \qquad n = -\log_2\left(\frac{E_{Landauer}}{E_{current}}\right) \tag{2}$$

Then, the total time to go from $E_{current}$ to $E_{Landauer}$ will be:

$$\begin{aligned} t &= n \cdot t_{Koomey} \\ &= -\log_2\left(\frac{E_{Landauer}}{E_{current}}\right) \cdot t_{Koomey} \end{aligned} \tag{3}$$

If the values gathered above are inserted to Equation 3, we can calculate the time from 2010 to reaching the Landauer limit to be:

$$\begin{aligned} t_{2010} &= -\log_2\left(\frac{2.85375 \cdot 10^{-21} J}{3.6 \cdot 10^{-10} J}\right) \cdot 1.57a \\ &\approx 57.9a. \end{aligned} \tag{4}$$

So, Landauer's limit should not disturb Koomey's law before the 2060s. There may of course be other natural constraints to the rate of improvements in the efficiency of computing, but predicting the advances of computation decades ahead is out of the scope of this thesis.

Thus, if a highly simplified assumption is made that QoE of a service is only dependent of the aggregate computing performance of the entire system used to deliver it, a company operating a data center with modern and relatively efficient facilites, where actual IT equipment counts for 80% of the total power usage, can cut its electricity bill by 40% after one and a half years of operation just by replacing all IT equipment with new hardware euivalent in capacity. It necessarily does not make much sense to replace hardware this frequently, as the increased CAPEX from the price of the new equipment and cost of installation would – at least with the current ratio between the prices of electricity and data center IT equipment – far exceed the decrease in OPEX.

What a data center operator actually can do to save significant amounts of money and energy is try to estimate required capacity as accurately as possible instead of

purchasing extra capacity just to be on the safe side. In addition to new equipment being more efficient, this reduces the overhead power consumed by "idle" hardware, which can sometimes lead to cooling problems, if this underloaded equipment does not generate enough heat for free cooling to work properly. Constantly acquiring new hardware to keep up with growth in turn leads to a heterogeneous set of hardware in a single data center in terms of processing power and efficiency. Methods for optimizing resource usage in such a setting has also been studied [29].

However, as discussed, the amount of computing and communications hardware, and their energy consumption, is increasing at a fast pace. This would suggest, that computing is a resource that is required in society at a highly increasing rate. This is of course largely accountable to the digitalization of everyday activities, which in itself does usually save more energy than it consumes, but as the amount of energy consumed by ICT increases, the significance of designing digital services to be as energy efficient as possible becomes more and more crucial.

2.2.4 Cooling Efficiency

As most of the energy consumed by the IT equipment in a data center ultimately transforms into heat, a significant amount of power needs to be used in cooling the equipment to prevent malfunction and breakage of hardware. Industry standard basic guidelines for equipment placement and airflow patterns, and environmental specifications such as temperature and humidity for optimal data center operation are provided by ASHRAE [30]. Analysis of metrics for data center air management and additional guidelines can also be found from a study by Tozer and Salim [31].

ASHRAE recommends a temperature of 20℃ to 25℃ for data center equipment considered class 2 IT equipment, such as small servers and storage products. It is reasonable to think, that there may be differences in energy efficiency inside these limits too. According to a study by Patterson [32], keeping the temperature at the lower limit is fairly certainly not efficient, but changes between, e.g., 21℃ and 25℃ may be nondistinguishable energy efficiency wise, as raised room temperature causes the IT equipment to spend more energy on fans. However, if the data center cooling system includes and economizer, raising the temperature has a much more dramatic effect in energy, as the annual period of time when outside air can be utilized for cooling becomes longer when the desired temperature is higher.

Maintaining the suggested threshold values for the environment with the minimal

amount of power consumed, on the other hand, is a separate layout and control optimization problem, which has been studied by, e.g., Zhou et al. [33, 34]. In these studies, it was discovered, that up to 36% of CRAC fan power can be reduced by introducing a dynamic control model for the temperature management of a model data center.

2.2.5 Data Center Location

When considering locations for new data centers, numerous points affect the cost-effectiveness and utility. Naturally the infrastructure, both in the traditional sense and network-wise, of the site and availability of personnel are prerequisites, but one must also take into account the political landscape, the availability of, e.g., water for cooling, and the robustness of the power-distribution network of the country.

Finland's suitability for housing data centers from the perspective of the power grid and environment has been analyzed by Malkamäki et al. [35] in 2011. They concluded, that when a data center has a relevant portion of operations or processing that are not urgent, scheduling the jobs according to the price of electricity has, in addition to being economical for the operator, a natural balancing effect on the electrical grid. On the other hand, it may be smart for a data center operator to choose to run jobs in locations or times that have a lower outside temperature, as savings may be incurred from cooling costs, but this has the adverse effect of increasing peak consumption in the grid.

Reusability of the heat produced by a data center during colder periods is also a factor dependent on location. A study conducted in 2011 in Finland concluded, that excess heat recovered from a particular data center could have met the yearly heating demands of a 26954 m^2 office building [36].

2.3 Metrics

In this section, prominent metrics related to data center operations and energy efficiency are discussed, and their suitability for the purposes of this thesis is analyzed.

2.3.1 Metrics for Energy Efficiency

The most popular and widely accepted metric for energy efficiency in data centers is PUE, which was developed by Green Grid [37].

PUE is defined as follows:

$$PUE = \frac{\text{total annual facility kWh}}{\text{annual kWh IT load}} \tag{5}$$

This is simply the fraction of the total power consumed by the data center that is actually used by the ICT hardware, or 1 + overhead kWh consumption. This metric thus considers only the excess power consumed by the data center infrastructure. However, the actual ICT hardware may be used in a manner that is not effective even if the PUE value of the data center is low. In this thesis the aim is in minimizing the required energy used by the ICT payload, so a different model has to be considered here.

A prominent metric for server power consumption in a virtualized environment has been presented in [38]:

$$P_{\text{serv}} = P_{\text{standBy}} + P_{\text{hypervisorIdle}} + A \cdot N_{\text{VM}} + a \cdot U_{\text{active}}, \tag{6}$$

where $P_{\text{standBy}} + P_{\text{hypervisorIdle}}$ represents the server power usage with no virtual machines on, $A \cdot N_{\text{VM}}$ represents the amount of power N_{VM} virtual machines consume at idle state and $a \cdot U_{\text{active}}$ represents the amount of power consumed by running applications.

The model defined in (6) is, however, highly dependent of the hardware used, and is probably most useful in cases where a data center company is contemplating what kind of server hardware to acquire. In this thesis, large scale data centers are considered and the target is to offer the same service with less power consumed, or alternatively, to offer more service with the same amount of power.

In a virtualized environment running at a high utilization rate this would mean being able to run more virtual machines on the same servers, which would in turn enable turning off some of the servers or running more virtual machines in total. Running services on dedicated servers without virtualization might be slightly more efficient, omitting the virtualization overhead, but in a large service, where relia-

bility is required, redundant nodes are necessary in any case. Also, by minimizing the amount of separate instances, the overhead created by different OS (Operating System) processes can also be minimized.

Thus, for the scope of this thesis, a highly simplified metric for virtualized system energy efficiency is proposed:

$$VE = \frac{N_{\mathrm{VM}}}{N_{\mathrm{host}}}, \tag{7}$$

where N_{VM} is the amount of "standard" virtual machines with a fixed amount of processing power and memory and N_{host} is the amount of physical machines hosting the virtual machines.

2.3.2 Metrics for QoE

ITU-T recommends the following definition for QoE: "The overall acceptability of an application or service, as perceived subjectively by the end-user." [39] Easily measurable QoS (Quality of Service) properties do not always portray the actual quality of the end-user experience. The relationship between QoS and Qoe has been analyzed in [40].

In web browsing, the general notions about perceived response times described by Nielsen [41] can be applied:

- If the system responses within **0.1 second**, it is perceived by the user as *instantaneous*.
- A response time under **1.0 second** will normally keep the flow of thought of the user *uninterrupted*.
- Before **10 seconds** has passed, the users *focus* will remain on the subject, but after that, the user will want to perform different tasks in the meanwhile.

When website (news portal, blog, etc.) browsing is considered, the most intuitive approach would be to use the page load time as a direct metric for QoE. However, contemporary browsers and websites can begin rendering a page before all resources have been received, which makes it more difficult to choose a specific point where the website has rendered sufficiently many visible components for the user. This is of course also dependent on the kind of service offered by the website in question.

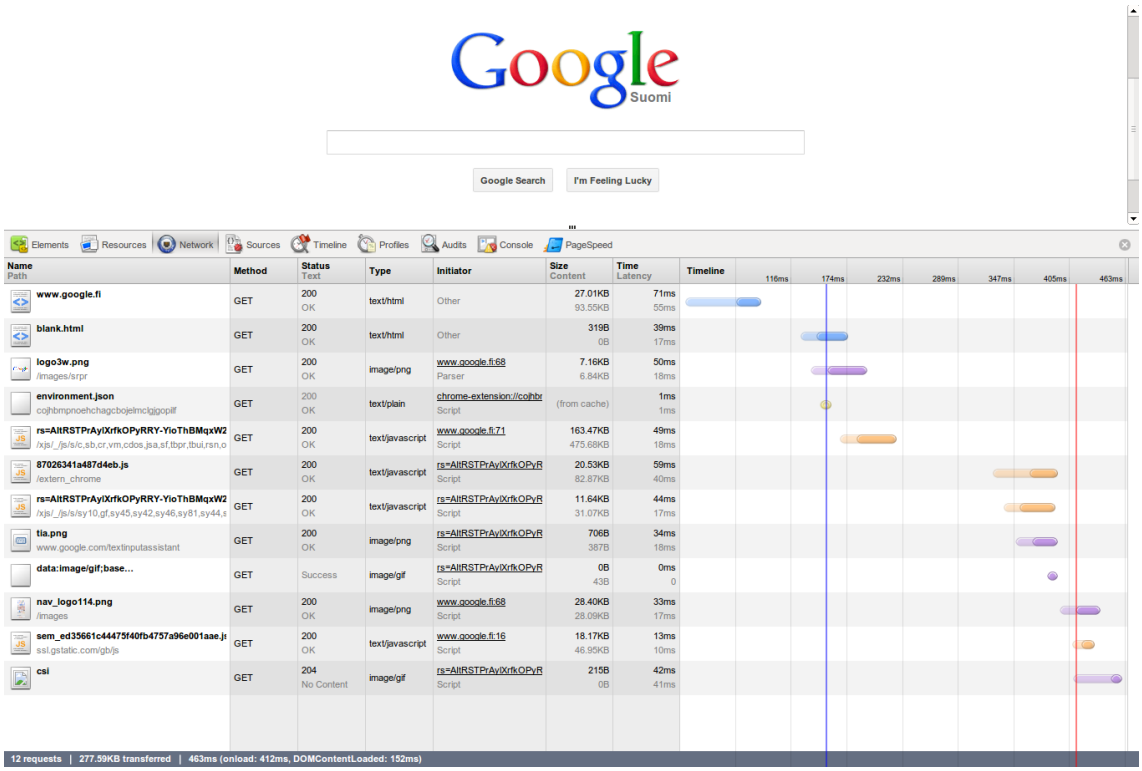

Figure 5: Screenshot of the website performance analysis tools provided by Google Chrome

Modern browsers such as Google Chrome and Mozilla Firefox provide excellent means for visualizing the process of getting the complete website visible on the user's screen from the first HTTP request onwards. This is illustrated in Figure 5. Google Chrome also provides a tool called PageSpeed, which gives advice for the website administrators on how to speed up loading their sites by means of optimization.

In this thesis, as the actual degradation in QoE as capacity is reduced is the point of interest, the notions above may be used to decide on some limits for a reasonable QoE which can be used as a basis in the testing scenario. The metric that is actually needed here is one for the change in QoE, which can justifiably be represented in the case of website browsing as the change in page load time, i.e., seconds, formally:

$$\Delta QoE = \Delta T_l. \tag{8}$$

Considering the page load time T_l, the shape of the curve can be estimated by deducing first, that as the server capacity approaches zero, the expected page load time grows infinitely high. So if the page load time is considered a function of server

capacity x, it can be formulated:

$$\lim_{x \to 0} T_l(x) = \infty. \tag{9}$$

Also, it is easily deducible, that as server capacity grows limitlessly, the expected page load time must asymptotically approach some positive value C:

$$\lim_{x \to \infty} T_l(x) = C. \tag{10}$$

Therefore it is likely, that the behaviour of the function $\overline{T_l(x)}$, i.e., the mean page load time, approximates a function of the following form:

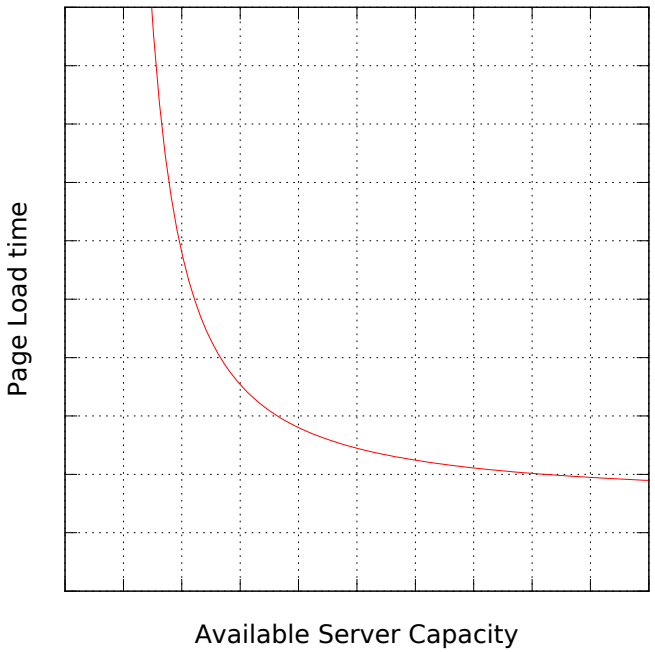

Figure 6: Illustration of the predicted behavior of the QoE of a system presented as a function of a single capacity parameter. The function used here is $\mathbf{a} \cdot e^{1/x} + \mathbf{c}$.

$$\overline{T_l(x)} = \mathbf{a} \cdot e^{1/x} + \mathbf{c}, \tag{11}$$

which is depicted visually in Figure 6. From this representation, the lower limit C for the page load time for a particular system, where one capacity parameter is

modifiable, can be calculated:

$$C = \lim_{x \to \infty} T_l(x) = \mathbf{a} \cdot e^0 + \mathbf{c} = \mathbf{a} + \mathbf{c}. \tag{12}$$

Now, if a web service setup is built where one or more parameters that should affect the performance of the system can be varied, approximate values for a and c in this particular setup can be found. When the objectives of this thesis are considered, the minimum value $a + c$ is not necessarily as interesting as the shape, particularly the steepness, of the curve. This is because the point where the gain in QoE becomes negligibly small when capacity is increased significantly is the most interesting. After this point it is most likely most beneficial to find where the bottleneck of the system has shifted to and start adding resources there. A method for allocating resources optimally when the bottleneck shifts has been studied already by Pradhan et al. in 2002 [42].

2.4 Web Technologies

Data centers store and manipulate the content required by the user, but in order to reach the user, the content needs to travel through the Internet. In this section, the technologies and protocols relevant to the user cases of this thesis are presented.

2.4.1 HTTP and HTTPS

Most of today's internet traffic, be it plain web pages, sound or video, is transported using HTTP (HyperText Transport Protocol). HTTP utilizes a TCP connection to transmit data between a server and a client.

In services where sensitive data is passed between the user and the server, passing the data unencrypted over the open Internet is ill-advised. In HTTP-based services, the standard way of creating a secure connection is to use the TLS (Transport Layer Security) protocol or previously its predecessor, SSL (Secure Sockets Layer). The Íatest version of the TLS protocol is defined in RFC 5246 [43].

In essence when accessing a resource via HTTPS, the security is established by performing an additional handshake where encryption key information is exchanged between the client and the server. To make this handshake traffic secure, the first message from the server, the server HELLO, contains a certificate including a public

key that the client can use to encrypt its messages so that only the server can decrypt them. Usually the client also verifies that the server is actually who it claims to be by checking that the server's certificate is signed by a trusted CA (Certificate Authority). Using the server's public key, the client may now negotiate a shared secret that can be used to encrypt data traffic. This handshake procedure is also depicted in more detail in Figure 7. [44]

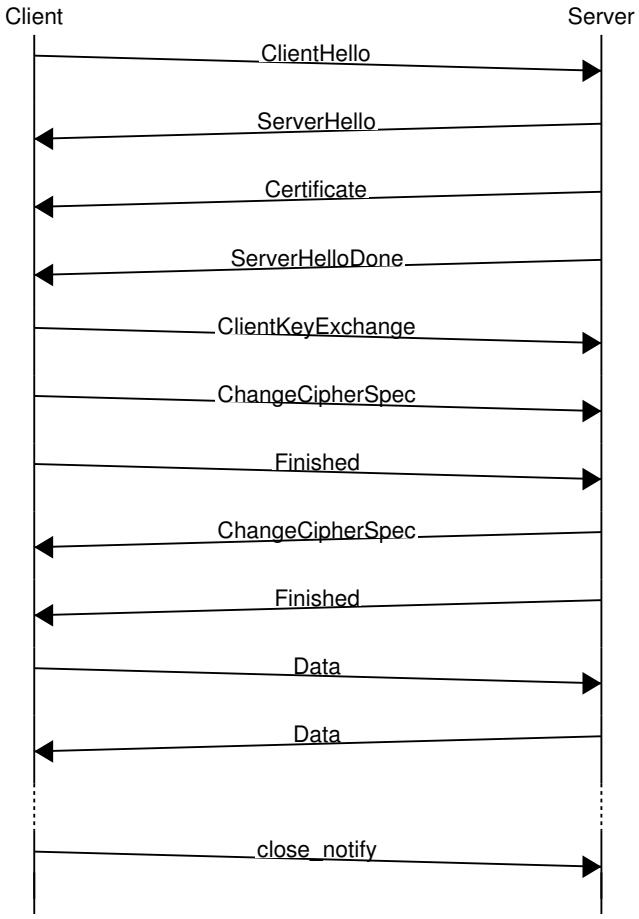

Figure 7: An overview of the basic SSL/TLS handshake

Like most modern cryptography, encryption algorithms utilized by TLS rely on random numbers, which means that a heavily loaded server needs to be able to generate sufficiently random pseudorandom numbers in large quantities. Additionally, encrypting and decrypting data naturally consumes more CPU-cycles. This combined with the additional round of handshakes depicted above makes communication over HTTPS slower and more resource intensive.

How the total processing time of a HTTPS transaction is divided between different phases has been studied in detail: 70% of CPU capacity is used in SSL processing,

where for small (< 32 kilobytes) amounts of payload the largest portion of CPU resources are consumend in the initial handshake phase. For larger amounts of data the encryption begins to make a difference too. [45]

Thus, there is a trade-off between security and QoE, and a web service designer needs to carefully analyze what the requirement for security in the particular task at hand is, as the choice of cipher suites supported by the service can have a dramatic effect on the hardware resource requirements for the service. The amount of available cipher suites for TLS is vast [46] and the relevant differences between them are in speed, client support and vulnerability to attacks.

In this thesis the aim is to distinguish the differences between HTTP and HTTPS traffic in the user-perceived QoE when certain server capacity parameters are altered. This information could then help designers of web services to plan how to allocate hardware resources to different parts of the system, such as application servers, HTTP servers and random number generators.

It is fairly self-evident that when capacity is added to an individual part of the system, at some point some other part will become the dominant bottleneck of the entire system when it is being loaded with similar load profiles. This effectively induces a lower limit for the QoE in such a scenario. The primary goal in this thesis is to locate this lower limit in a particular system for both HTTP and HTTPS, and to discover how the system behaves before reaching this limit. This information provides additional insight for optimizing services for energy and cost efficiency.

2.4.2 P2P and CDNs

Peer-to-peer networking in general is defined by Schollmeier to mean the following:

> A distributed network architecture may be called a Peer-to-Peer (P-to-P, P2P, ...) network, if the participants share a part of their own hardware resources (processing power, storage capacity, network link capacity, printers, ...). These shared resources are necessary to provide the Service and content offered by the network (e.g. file sharing or shared workspaces for collaboration): They are accessible by other peers directly, without passing intermediary entities. The participants of such a network are thus resource (Service and content) providers as well as resource (Service and content) requestors (Servent-concept). [47]

Ẃhen P2P-technologies are mentioned, many people might think of illegally downloading copyrighted content. However, P2P has proven very useful in, e.g., multimedia applications such as Spotify and Skype, and distributing large files of open content via BitTorrent [48]. Of these, the Spotify use case is the most interesting one, as it is a very popular service, and the P2P-scheme enables significant savings in data center capacity by utilizing the bandwith and storage space of its users.

For P2P to be a beneficial option for a service architecture, an intuitive requirement is, that there has to be a large enough cumulative upstream bandwidth worth of peers sharing the requested resource at any given time. This can be achieved either in a situation where a large amount of users want the same resource at the same time or when each user has a large amount of different resources constantly cached for sharing purposes even if they are not using the resource themselves. In the case of Spotify, both of these scenarios apply, as popular songs are listened by many people simultaneously, and due to the relatively low bitrate of even high quality audio files, a large amount of music can be cached on the users' hard drives.

Schollmeier's definition of "hybrid" P2P is as follows:

> A distributed network architecture has to be classified as a "Hybrid" Peer-to-Peer network, if it is firstly a Peer-to-Peer network according to Definition 1 and secondly a central entity is necessary to provide parts of the offered network services. [47]

Spotify utilizes a P2P architecture that could be considered to be of the hybrid flavor, as opposed to the "pure" variety, as central servers are required for initial dissemination of the content and also for retrieving information about peers that have the requested content available. To minimize waiting, the first 15 seconds of content that could not be predicted during playback of the previous track are also fetched directly from the central servers. [18]

As was mentioned in Section 2.2.1, the amount of video traffic will most likely continue to grow at an accelerating rate. Streaming media providers are thus required to make sure that their services scale to the requirements of the near future in time. Netflix seems to be looking in the direction of P2P, at least judging by their job advertisements [49]. This is a particularly welcome, as well as natural, development trend, since it may allow for substantial savings in energy and global network capacity in the near future, as the premises for popular video content such as TV series and movies are quite similar to sound. apart from the bitrate, which for HD video

is quite high, at least in comparison.

However, not all types of services benefit from P2P significantly. Namely, services that provide content that is dynamic and distinct for each user still require central servers that store the data and process responses to different kinds of user requests according to the business logic of the service in question. This usually contains querying databases and combining different kinds of content, such as multimedia. As many present-day services, especially in social media, have a large amount of customers or users on every continent, serving data-intensive content like images, sound and video quickly becomes very challenging if each and every piece of data needs to be transferred from a single central location. In addition to the servers, this puts great strain on the network infrastructure.

To ameliorate this situation, most of the aforementioned data-intensive content is served via CDN (Content Delivery Network) services such as Akamai Technologies, Amazon CloudFront and Rackspace Cloud Files. These services essentially provide a set of globally distributed servers which act as cache servers that are relatively close to the client. Serving all static, i.e. non-dynamic or rarely changing, assets are served through these servers. In addition to the large media files, it is also beneficial to serve, e.g., client side javascript and CSS files via a CDN.

This has left the service provider's servers only to take care of the business logic of the service, i.e. the software that accepts incoming requests and responds accordingly. This usually also requires querying a database and processing the data, which requires a varying amount of memory and CPU resources.

2.5 Chapter Summary

This chapter was about data centers and data center trends in general and the factors and limits affecting their energy efficiency. A salient point is, that data centers are an integral part of modern society, and increasingly so, but unless measures are taken to drastically improve their energy efficiency, trouble is ahead.

Luckily there are multiple layers at which the energy efficiency can be improved upon, namely, PUE, or the efficiency of the data center infrastructure, the actual IT equipment, which currently is not very power proportional, and finally, the optimality of the service itself, architecture and configuration wise.

Furthermore, Koomey's law dicdates, that energy efficiency of computers is con-

stantly increasing at a regular pace for a few decades to come still, so data center operators need to take into account the savings that can be incurred by switching to newer hardware.

3 Experimental Methods

Experiments on the effects of different system parameters in the service cases in this thesis was conducted utilizing real enterprise hardware in a real data center environment. In this chapter, the test system is described in detail.

3.1 Requirements for the Experimental Setup

As was mentioned in Chapter 2, the main point of interest in the experimental part of this thesis is to find out the effect that reducing server capacity has on the QoE of a web service that is hosted in a data center like environment. In order to achieve this, a setup where a service is running is naturally required, along with measurement software or equipment that is capable of generating load and measuring response times.

For the system infrastructure, in practice there were two options: either purchase virtual server instances from an IaaS provider or build an own virtualized server setup somewhere. Both of these options have their benefits. Outsourcing the actual hosting is much cheaper and faster, and it requires less expertise from the system administrator and practically no equipment or space. Building your own system however enables complete control over the system and its parameters, and measurement accuracy is not limited by the network bandwidth. It also is a great learning experience.

Measurements can be carried out either by purely software based tools or a dedicated device. Software tools have the advantage of being cheap and simple, but expensive measurement devices offer means for creating very sophisticated testing patterns and higher loads. Additionally, online services such as `http://www.webpagetest.org/` are available for measuring differences between clients in diffent geographical locations.

3.2 Infrastructure and Hardware

The test system resides in a data center on the base floor of Otakaari 5, Espoo. The data center features a CRAC (Computer room air-conditioning) and a rack cabinet with direct liquid cooling. The blade server chassis, disk system and network switch used in this experiment are located in the water cooled rack cabinet. The liquid

heated in the heat exchanger of the rack cabinet cooling system is circulated through a separate cooling facility outside the building, in the parking lot between Otakaari 5 and Otakaari 7, which can be seen in Figure 8. The CRAC is utilizing the same liquid for cooling the air above the floor tiles, which it then pushes under the tiles. The cold air is brought up from under the floor in the front of open server racks through grill tiles.

Figure 8: A photo of the cooling unit that utilizes rooftop fans to cool the liquid that is used in the data center by the CRAC and water cooled cabinets.

The servers used here are IBM HS22 blade servers that reside in an IBM BladeCenter H chassis. The blade servers are equipped with an Intel Xeon processor with 6 cores at 2.27 GHz and 18 Gb DDR3 RAM. The blade servers are connected to the network switch (HP ProCurve) via two 1 Gbps copper pass-through ethernet interfaces each (one for customer traffic, one for out-of-band management), and connection to the disk system (IBM DS5020) is established via a 4 Gbps FC (Fibre Channel) interface. The physical hardware setup and cooling system layout are illustrated in Figure 9 and Figure 10 includes a photo of the actual hardware.

Initially the hardware setup also included Cisco Nexus 5000 series switches, which were set up between the disk system and the server chassis in order to provide redundant connections and to better simulate a connection to an actual SAN (Storage area network), but these switches were returned before any measurements were

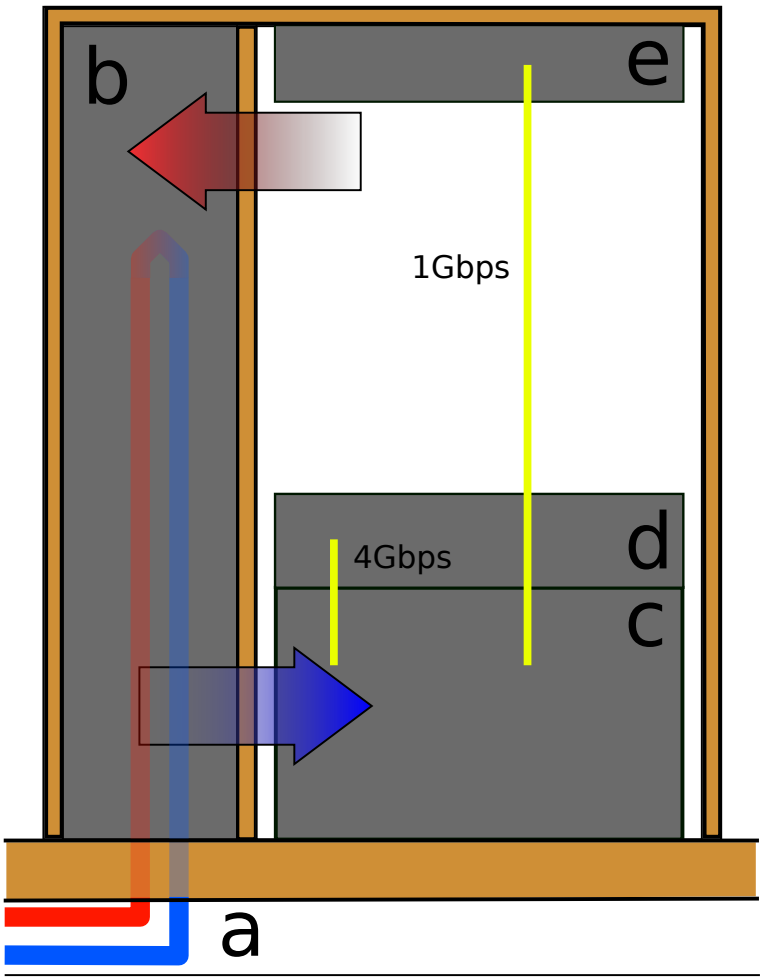

Figure 9: An illustration of the hardware setup used in the measurements, including the cooling system integrated into the rack cabinet. Legend: a) the space between the floor and the elevated tiles, which houses cold air and the pipes for the cooling liquid, b) the heat exchanger and the fans that take in the hot air from the top of the server cabinet, cool it and pass it back to the bottom of the cabinet, c) BladeCenter chassis, d) HS 5020 Disk system and e) ProCurve Ethernet switch.

Figure 10: A photo of the hardware used in the measurements.

carried out.

3.3 Software and Service Architecture

For our HTTP test case, we have set up a distributed system (although in this case all the different components reside on the same physical server) using Xen virtualization and separate VMs (Virtual Machines) for different tasks. These tasks are

- Application processing,
- Static file serving,
- Load balancing,
- Database and
- Caching.

For each of these tasks there is at least one VM assigned. To boost performance, we created additional application processing VMs, and altering the number of these VMs gives us the basic parameter of our measurements. Each of the VMs have a Ubuntu Linux operating system underneath the actual service software. It is possible to allocate CPU resources for VMs in such a way, that all VMs share the same CPUs with certain priorities, but for this study, dedicated CPUs were set up for each server in order for the effects of actually modifying the available capacity to become more observable.

The choice and configuration of software handling the different tasks above makes a great difference, and it is an important task for the people responsible for designing the particular parts of the system to try to choose the optimal software components and configure them according to the task. For many of the normal task assosciated to web services, there is an abundance of choice, and one can choose between a simplified and easily maintainable piece of software over a more complicated one with more features. Nearly every task can also be handled using free and open source software. In the experimental setup for this thesis every piece of software apart from XenServer is free software.

For application processing a CMS (Content Management System) called Mezzanine was chosen, which is based on the django [50] framework used by, e.g., Instagram [51] `http://www.washingtonpost.com/`, `http://www.nytimes.com/` and `http://www.`

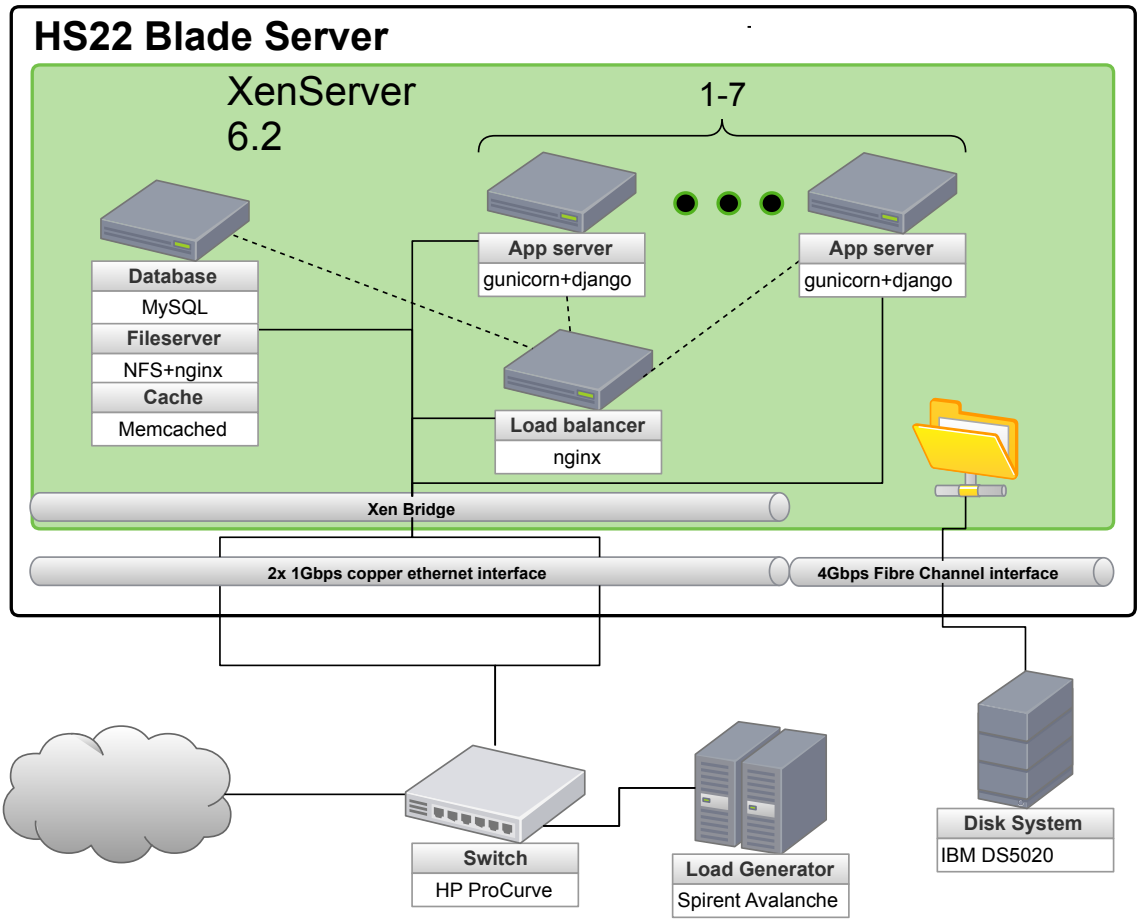

Figure 11: An illustration of the software architecture used in the HTTP measurements. All the software components are physically run in one blade server inside the BladeCenter chassis, while hard drive volumes are provided by the DS5020 Disk system. All traffic towards clients are passed through one 1 Gbps copper ethernet interface (the second one is used for out-of-band management).

`theonion.com/`. The actual HTTP serving on the application servers is carried out by the Gunicorn WSGI (Web Server Gateway Interface).

For serving static files and load balancing the popular nginx [52] server software was used. The DB (database) server was equipped with MySQL [53] and caching was provided with the memcached [54] software. This distributed infrastructure is managed by the Chef [55] configuration management tool. The architecture presented here is illustrated in Figure 11

It is probably worth noting, that the collection of server software, or *technology stack*, used here is comprised entirely of freely available open source software, and is fairly similar to the one used currently by – among others – Instagram[51], with the exception of database software.

The major difference between the test setup and a real large scale services aimed at a global audience is the fact that the latter frequently rely on a CDN to deliver static assets.

When building the service, it became clear already in the beginning, that there is an enormous amount of options and optimization schemes for each software component of the system. For example, during initial testing, the choice of cipher suites offered for HTTPS traffic had a tremendous effect in page load times. For cipher suites there luckily is scientific data around to use as a basic guideline, but for most of the other components, mostly anecdotal evidence of a piece of software being superior to another exists. This is of course due to the fact, that there is such a wide variety of web services that do different things in different environments, so no one best solution for every scenario can usually be identified.

Managing a single server instance can be tricky enough, but as the amount of virtual machines grew past two, the management became so cumbersome, that the Chef tool became invaluable. In essence, Chef enables a system administrator to define the system infrastructure in an environment resembling a code repository, where different roles are assigned different software configurations (called recipes in Chef), and as server instances are assigned roles, all changes can be easily and simultaneously deployed to relevant instances. An example of a recipe used to configure the application server nodes in the system can be found in Figure 12.

The last line of the recipe in Figure 12 shows, that the Gunicorn WSGI servers running in the application server nodes are configured to utilize asynchronous workers using the Eventlet library [56], which enable concurrent processing using threads. Another piece of configuration which is meaningful from a performance point of view is the nginx configuration that is used in the client facing servers, i.e., the load balancing proxy and the static asset server. Relevant bits, such as the number of worker processes, allowed SSL ciphers and usage of compression, of this configuration are listed in Figure 13. Additionally, for the load balancing proxy, a load balancing mode that chooses the server with the smallest amount of active connections to serve an incoming request was configured.

3.4 Anatomy of the Client-Server Intercommunication

In this section, a detailed description on how a client request is processed and a response is returned in our distributed test setup is given. As mentioned, the

```
include_recipe "supervisor"
include_recipe "python::pip"

app_name = "dc2f_blog"
local_port = "8088"
app_url = "blog.prod.dc2f blog.mgmt.dc2f"

project_home = "/var/www"
app_home = "#{project_home}/#{app_name}"

python_pip "django"
python_pip "mezzanine"
python_pip "eventlet"
package "gunicorn"

git "#{app_home}" do
  repository "git://git.mgmt.dc2f/git/dc2f_blog"
  revision "master"
  action :sync
  notifies :run, "execute[collect_static]", :immediately
  notifies :restart, "supervisor_service[dc2f_django]", :delayed
end

execute "collect_static" do
  command "/usr/bin/python #{app_home}/manage.py collectstatic --
    noinput"
  action :nothing
end

supervisor_service "dc2f_django" do
  supports :enable => true, :disable => true, :start => true, :stop =>
    true, :restart => true
    autostart true
  autorestart true
  redirect_stderr true
    user "www-data"
    command "/usr/bin/python #{app_home}/manage.py run_gunicorn -b #{
    node['mgmt_ip']}:#{local_port} -w 3 --worker-connections 2000 -k
    eventlet"
end
```

Figure 12: An example of a Chef recipe used in the experimental setup. This recipe installs and configures the relevant software components used in the application server nodes of the system.

```
...
worker_processes 4;
...
events {
    worker_connections 5000;
}

http {
    ...
    gzip on;
    gzip_disable "msie6";
    gzip_vary on;
    gzip_types text/plain text/css application/json application/x-
    javascript text/xml application/xml application/xml+rss text/
    javascript;
    gzip_min_length 150;
    proxy_buffers 8 256k;
    ssl_ciphers ALL:!kEDH:!ADH:RC4+RSA:+HIGH:+EXP;
    ...
}
```

Figure 13: Relevant snippets from the nginx configuration file utilized by the client facing nodes of the system.

software component stack is fairly similar to many contemporary web services, and thus also the behaviour of the system is commonly seen in applications accross the web.

First, the client wishes to access a resource, say, the listing of posts in a blog. This is done by sending an HTTP GET request to the server with the resource URI. To do this, the client requires the IP address of the server, which it requests from a DNS server. If DNS-level load balancing [57] was employed, the DNS server would respond with an address from a pool of server addresses that are associated to the DNS name in the request. These addresses are distributed to clients in a round-robin fashion, or in sophisticated setups utilizing multiple data centers or a CDN, the physical location of the client may be used to decide which server would be most suitable. However, in this test setup, DNS-level load balancing was not used.

The server that initially receives the request, i.e., the server facing the Internet directly, in this setup is essentially a load balancing proxy server. The software handling the request here is *nginx*, which makes a decision based on the request URI to pass the request to one of the application software servers running the django software and the Gunicorn WSGI. The application server software then processes the request by first queries the cache server running *memcached* to find out if a

response to the request in question is already cached. If not, the django software makes the necessary DB queries to the DB server and assembles the HTTP response – An HTML document in this case – and saves it into the cache before sending.

The application server then passes the response back to the proxy server, which relays it to the client, encrypting it first if HTTPS is requested. The client then processes the HTML document and finds that it needs to request also static assets such as Javascript, CSS and images from the server. The proxy server associates the URI of the request to a static resource and passes the request on to the server dedicated to serving static assets. This server is also running nginx, which encapsulates the response into an HTTP message and sends the message back to the proxy. The proxy then handles this in a similar way to the HTTP messages from the app servers. This entire exchange is also depicted in Figure 14.

3.5 Measurement Equipment and Methods

For generating server load, i.e., HTTP requests, and capturing responses with timing a Spirent Avalanche network testing device was utilized. The Avalanche enables creating a variable load profile where the amount of base load and churn can be accurately controlled on an "amount of simultaneous users" basis, which is very approppriate for emulating a usual stress scenario for a web service. Regrettably the device is slightly limited in terms of analyzing the server response times, as it only provides values for average, maximum and minimum response time.

A load profile for a 10 minute stress test was set up, where the base load is first at 100 simultaneous users, then at 300 and finally at 200, and above that there is a 50 user churn. The load profile is illustrated by simultaneous users as a function of elapsed time in Figure 15. Each of these simulated users in the load profile send a series of HTTP (or HTTPS) requests to the server, mimicking the behaviour of a web browser that fetches content from a blog site to a user.

A test utilizing the load profile described in Figure 15 was executed on the test setup 14 times with two varying parameters: amount of application server virtual machines turned on (1-7) and SSL encryption on or off.

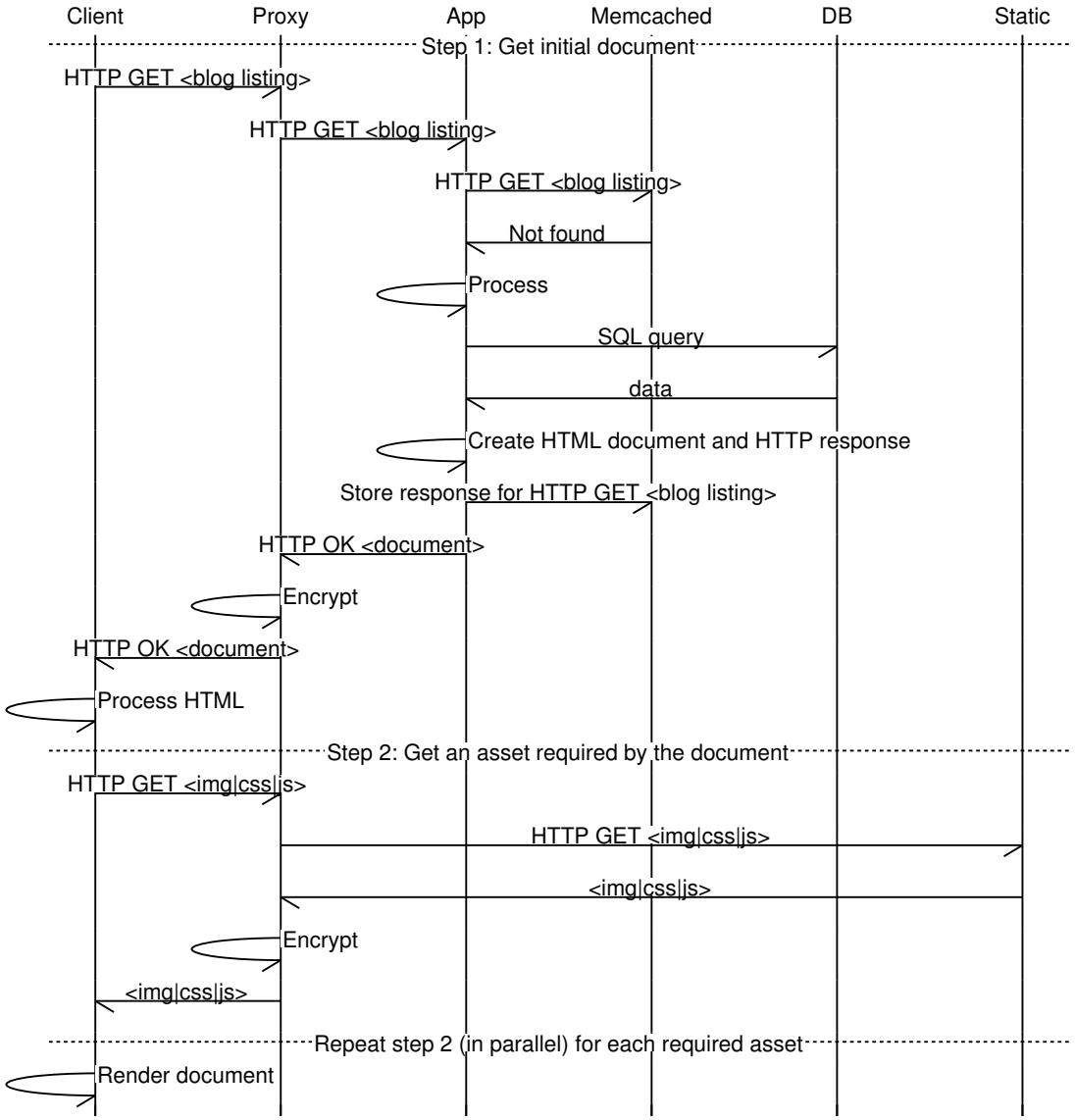

Figure 14: Description of the communication between separate entities in our test setup in a scenario where a client wishes to display contents of a web page.

3.6 Chapter Summary

In order to properly measure web service performance and QoE, one must set up an actual service. For this thesis, the system was designed and built completely from scratch, but using real industry standard components and an open source software stack similar to the ones used by many popular actual web services.

The components of the system have countless parameters that can be tweaked, and to get reliable results, these configurations need to be sane, consistent and actually usable. On the software side, particularly the proxy server SSL configuration was

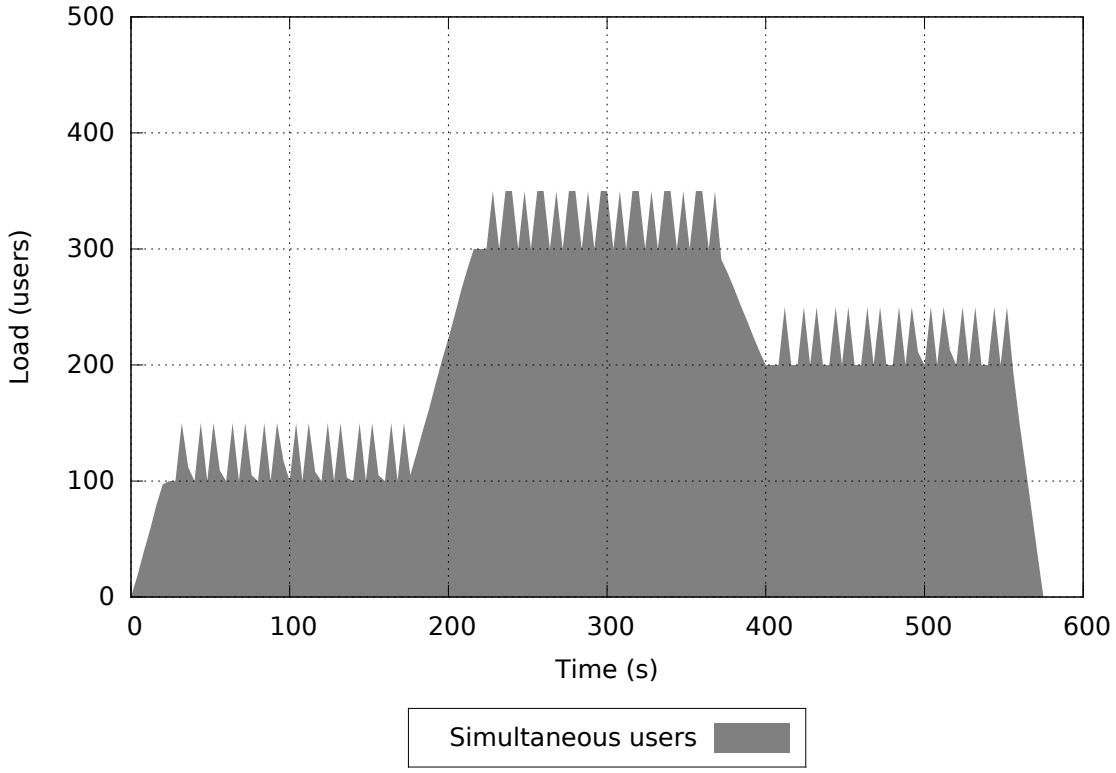

Figure 15: The load profile used in the HTTP performance measurements. The graph represents the amount of simultaneous users still waiting for a complete response from the server as a function of time elapsed into the test.

very important to optimize correctly, along with the application server processes.

Also, to be able to create a measurement load configuration that measures the right things, one must understand the processing and different requests that are carried out during a web page load.

4 Results

Measurements using the system and methods described in the previous chapter were carried out, and result spreadsheets were collected from the measurement equipment. Essential figures were then identified from the data, and they are presented in this chapter using graphs and tables, and then evaluated.

4.1 Graphs and Figures

As was discussed in Section 2.3.2, the goal was to identify how the QoE of the service behaves when a performance parameter is varied, i.e., QoE as a function of server capacity, in this particular setup for both HTTP and HTTPS, and compare the results. The measurement equipment provided an abundance of data, with the exception of standard deviations, e.g., average, maximum and minimum times for URL load, page load, first TCP stream byte received, etc. To best visualize in detail the collected data for this task, the average page load time was plotted as a function of time, with the number of active clients for reference, and the maximum page load time to approximate the deviations. An example of these plots can be seen for the scenario of one application server and HTTP in Figure 16. Figures for the other scenarios are presented in appendix A.

However, from the detailed plots such as Figure 16 it is difficult to see the development of the QoE accurately, as there are multiple different phases of load in a single figure, and to see the big picture, one would have to combine 7 figures visually. To better summarize the result data and visualize the effects on QoE, three 100 second time regions of nearly constant (apart from the client churn) load were chosen from the load profile according to Table 1 and Figure 17. The regions were selected so that the system would most probably be in steady state after the transients caused by the growing or decreacing amount of simultaneous users in the beginning and between different mean load regions. Load times in these regions were then averaged, and the results are depicted in Figure 18 and Figure 19 for HTTP and HTTPS, respectively.

From the curves fitted for the HTTP experiment data in Figure 18 it can be seen, that behaviour of the system corresponds to the model projected in Equation 11 in Section 2.3.2 rather nicely. In this particular case, it would seem that utilizing more than 3 application software servers for the task would not yield significantly

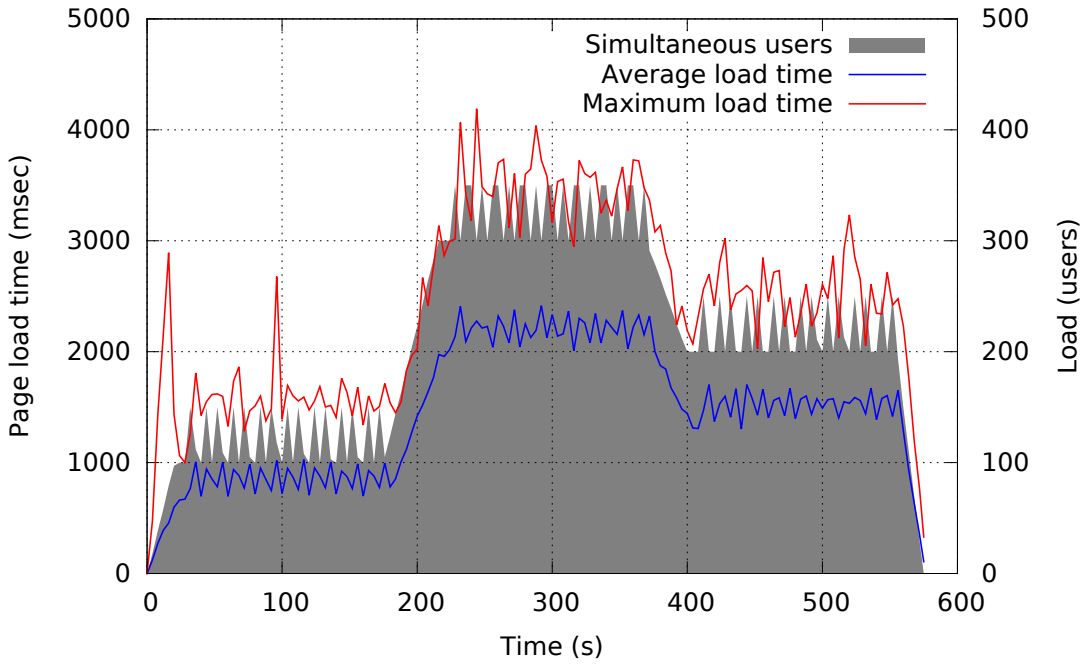

Figure 16: Load times for 1 app server for HTTP requests.

Table 1: The time regions from which data is averaged in the combined result plots.

Region	Time span (s)
1	50 – 150
2	250 – 350
3	450 – 550

improved performance, as the curve is rather steep in the beginning, and almost level after that point.

This notion seems to hold for each of the different load regions equally, which would indicate, that the bottleneck of the system is shifting to some other component at that particular point. The absolute change in page load time going from 1 application server to 3 servers appears to be somewhat greater for the busier load region (region 2) compared to the regions of lesser load: in region 2, increasing the amount of application servers from 1 to 3 causes the page load times to drop from an average of about 2200ms to 1300ms, whereas for regions 1 and 2, the drops are from 900ms to 300ms and from 1500ms to 500ms, respectively.

The data gathered from the HTTPS experiment seen in Figure 19 behaves in a similar way compared to the HTTP experiment, with the exception that the curve

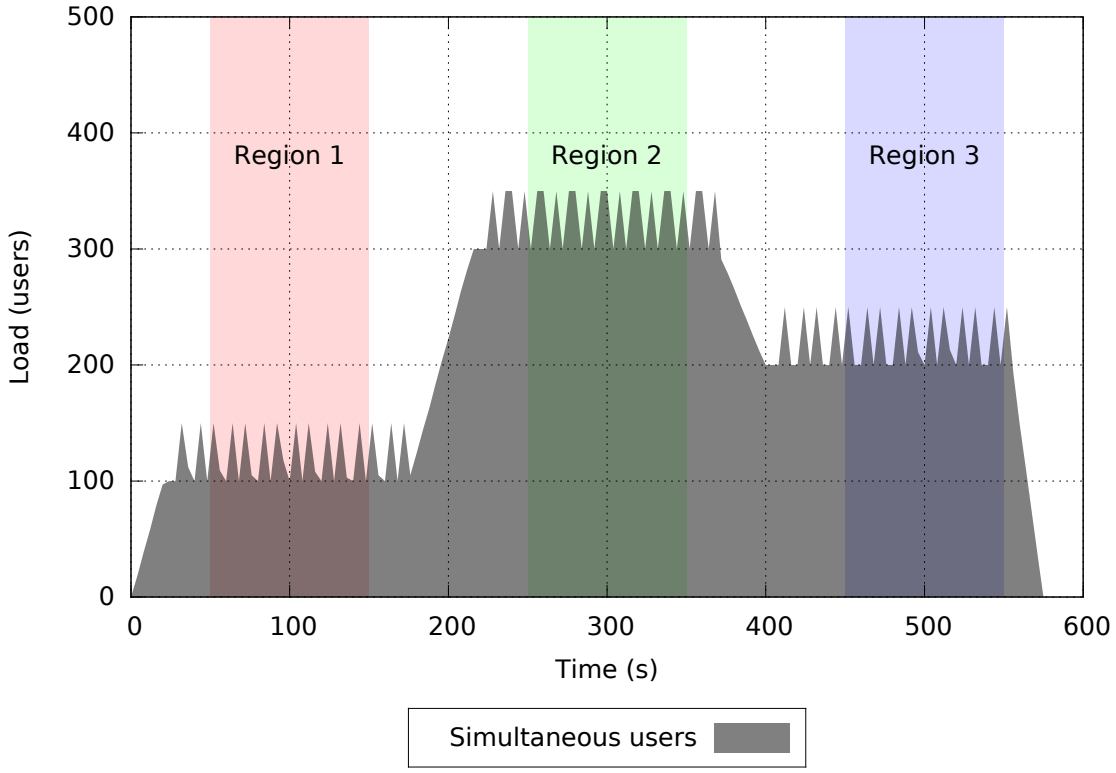

Figure 17: Visualization of the time regions from which data is averaged in the combined result plots.

is even steeper and the lower limit is thus reached with an even smaller amount of application servers, namely two. This indicates quite clearly, that the bottleneck in this case after two or more application servers becomes the proxy server, as from the system point of view, HTTP and HTTPS processing differ only at that point, as the proxy server needs to do additional processing to decrypt the requests and encrypt the responses.

As the theory matches the actual data quite well, values for the coefficients a and c in Equation 11 for this particular system and scenario can be estimated from the results of the least squares fitting. These results from each scenario are compiled in Table 2, along with the values for the lower limit in QoE calculated according to Equation 12.

Another interesting piece of information provided by the measurement equipment is the time from client birth to first received byte of data inside an HTTP message. This information was plotted in the exact same way as the page load time, and an example of the plot can be see in Figure 20. The rest of these detailed plots can be

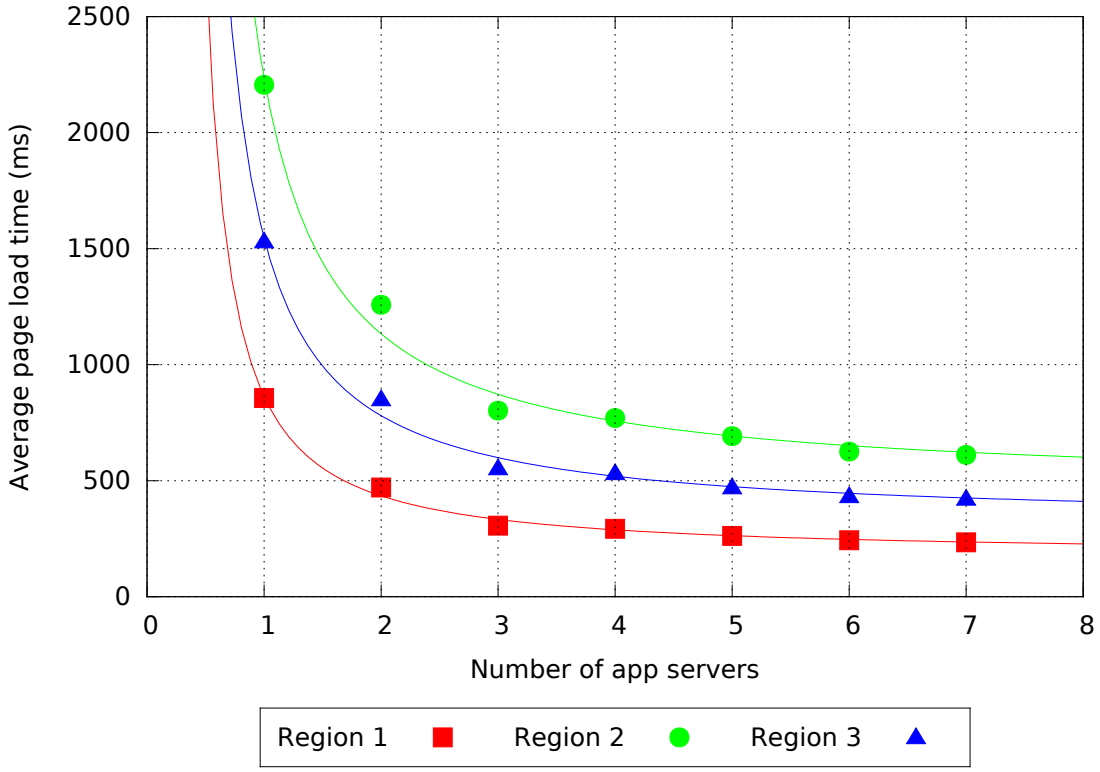

Figure 18: Average page load times in the different load regions for HTTP traffic with varying application server capacity. The values on the x-axis correspond to the amount of application servers active in the system. The curves that are fitted to the data points using the least squares method are in the format $y(x) = \mathbf{a} \cdot e^{1/x} + \mathbf{c}$.

Table 2: The numerical results of the curve fitting for Figures 18 and 19 and the lower limit for QoE in each case calculated from these values according to Equation 12.

Protocol	Region	Parameters		$\lim_{x\to\infty} \overline{T_l(x)} =$
		$\mathbf{a}$ (ms)	$\mathbf{c}$ (ms)	$\mathbf{a}+\mathbf{c}$ (ms)
HTTP	1	401 ± 15	-227 ± 24	174 ± 39
	2	1031 ± 49	-466 ± 78	564 ± 127
	3	716 ± 29	-400 ± 46	313 ± 75
HTTPS	1	187 ± 22	$328. \pm 35$	515 ± 58
	2	471 ± 68	883 ± 109	1355 ± 177
	3	326 ± 45	613 ± 73	939 ± 118

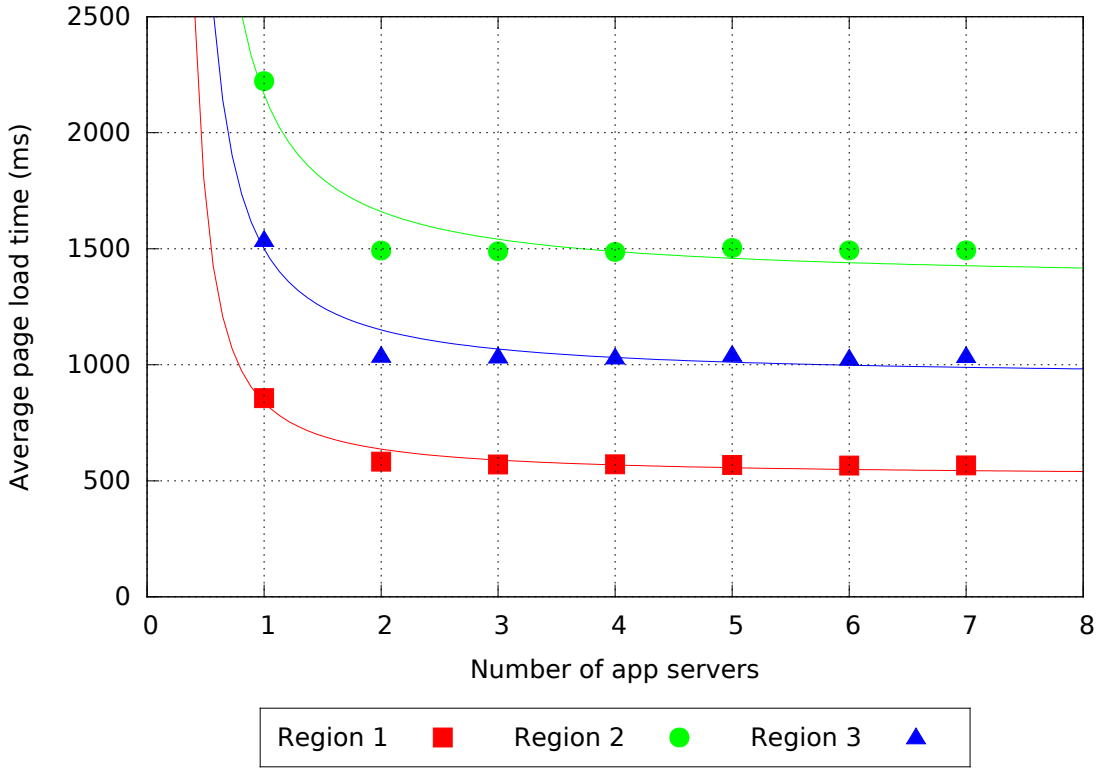

Figure 19: Average page load times in the different load regions for HTTPS traffic with varying application server capacity. The values on the x-axis correspond to the amount of application servers active in the system. The curves that are fitted to the data points using the least squares method are in the format $y(x) = \mathbf{a} \cdot e^{1/x} + \mathbf{c}$

found in appendix B.

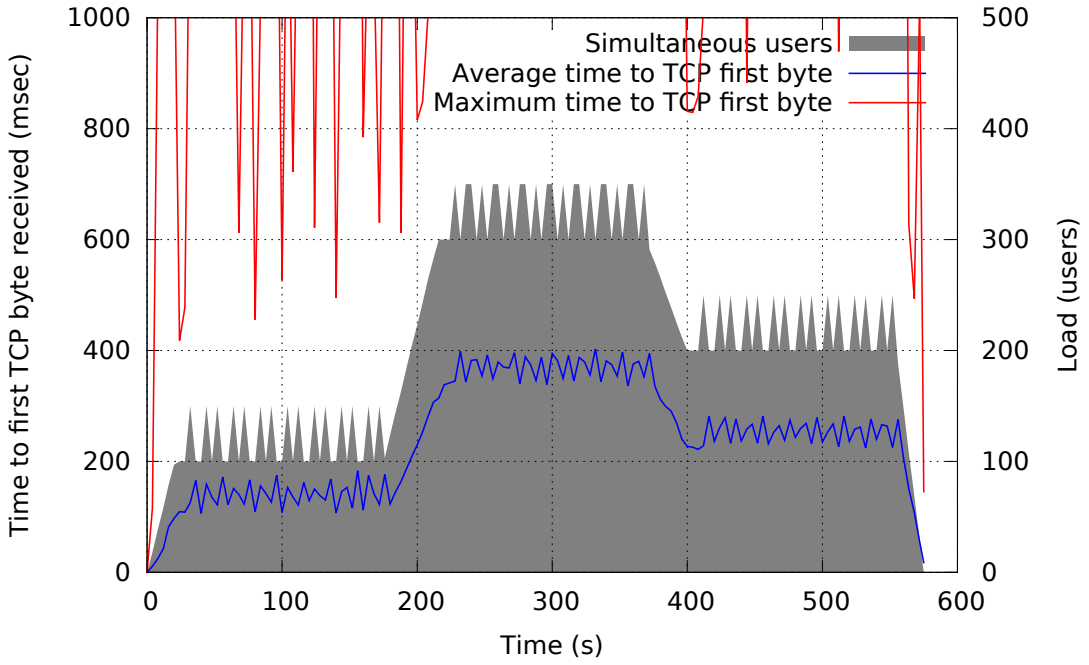

Figure 20: Time to first reveived byte for HTTP requests with 3 application servers..

Interesting about this information is, that while the HTTP scenarios behave similarly to page load times, for HTTPS, the time of receiving the first byte is actually increasing as more application servers are added to the system. Additionally, the times for HTTPS are significantly lower than for HTTP, although intuitively one would think it should be the other way around. This may be caused by, e.g., some component of the system doing caching in a place we were not aware about. This mostly goes to show, that in these kind of systems, where the amount of separate software components is large even for a fairly simple task, it is quite difficult to be aware of all the effects a piece of software has on the end result.

4.2 Evaluation of the Results

The results presented above do not necessarily provide any figures that would be directly applicable as some kind of guidelines or general thresholds. In this section, the significance of the results is evaluated and the additional requirements that should be posed to the measurement system in order to obtain additional and more useful results are discussed.

As was mentioned in Chapter 2, the spectrum of components available for building

a web service is vast, and even within a single piece of server software, there is a massive amount of configuration that can be tweaked. This means that conducting an exhaustive study where all different options are compared, even for a single use case, would be next to impossible.

Here the figures that were presented are all obtained using the same exact architecture and software components and configuration, so in order to make sure that different choices in, e.g., software used does not completely transform the results, additional setups should be built and measured according to the same specifications. Before this kind of verification measures are completed, the results presented here have to be considered fairly anecdotal.

Nevertheless, the kind of holistic results from particular use cases gathered here have mostly been absent in research of this field, as most existing data is centered on either the performance of single components of a service or, on the other hand, energy consumption in data centers in a more general setting. This might of course indicate, that these kind of results are not too useful as such, but again, providing more insight to identifying bottlenecks and the distribution overhead in systems like the one studied here could enable coming up with more sophisticated heuristics and optimization processes for mitigating these problems.

The most important limitation of the test setup was, that altering the hardware capacity was practically achievable for only one component of the distributed system architecture. In order to find further bottlenecks, the amount of virtualized server capacity allocated for other components such as the proxy server, the cache tier and the database tier should also be modifiable. As the amount of degrees of freedom increase in the measurement scenario, the more complicated the actual measuring becomes, and automating the measurements and the changes in the system capacity values between measurement runs becomes essential.

Creating this kind of automation should be feasible using the same tools and systems that were in use here: Xen and Chef are designed to be used with automated monitoring tools, and Spirent measurement tools do provide interfaces for scripting, etc. Designing the actual automation software would naturally be a large project, but enabling the study of the aforementioned heuristics and optimization processes could be very beneficial. The measurement system could ultimately even be used to benchmark and optimize the parameters of any system with similar design.

Initially, the plan was also to measure actual power consumption in the system, but

as no suitable equipment for reliably measuring the consumed power was available, this part was dropped. Preliminary tests did however indicate, that the hardware used in the system was rather poor from a power-proportianlity point of view, which seems to still be a prevalent trend, as was discussed in Chapter 2. However, as the services are separated from the hardware by a layer of virtualization, the effects of the actual service load to the power consumption of the hardware might not be as critical compared to its ability to energy efficiently virtualize resources.

4.3 Chapter Summary

In this chapter, the main results of the measurements were presented very compactly in two aggregate plots: one for HTTP, one for HTTPS. The results were combined by picking three clear load regions from the results and plotting them as separate curves according to the predicted behavior of QoE.

The results seemed to match the shape of the plot estimated in Chapter 2 very nicely, and some characteristic figures were calculated from the plot fitting data to a table. The main finding from the results is, that for both cases, but especially HTTPS, the number of application servers after which performance nearly stops improving is surprisingly low.

The significance of these results has to be considered fairly limited, as there are so many parameters that can be altered, that it is nearly impossible to draw reliably generalizable conclusions. However, the results do provide some new insight for testing and benchmarking such scenarios in the future.

5 Summary

In a nutshell, the purpose of this thesis was to cover the topic of data center energy efficiency as a whole, while concentrating on some fields that have previosly received less attention. Also the perspective that was pursued here was in a way reversed from the ordinary, as the the aim was to identify points where data center processing capacity might be decreased with little enough effect to QoE. Novel findings from the field data center energy efficiency related to, e.g., data center location and virtualization were presented and summarized, energy and efficiency trends were presented and analyzed, and Koomey's law and its implications were thoroughly covered. The experimental part of the thesis was a case study into a simple web service from the abovementioned point of view.

The experimental setup used in the case study was designed according to current trends of web service system design but naturally limited by the hardware that was readily available or generously donated. The system was built literally from scratch for the purpose of theses around this subject. This was certainly not the fastest way to gain measurement data from this kind of setup, as there are many service providers offering virtual machines at low costs for systems like this. However, getting to know the details of the cooling and power systems, building the hardware setup, designing the network, installing operating systems and all the software provided us with some very valuable insight about the inner workings of data centers. Building and maintaining your own system also enables measuring practically anything, when outsourced service providers don't provide customers with all the information relevant for the purposes of science.

The measurements in the case study provided us with some valuable information about how this type of systems behave in certain web service scenarios: the amount of application server capacity ceased to be the bottleneck at a relatively low number of VMs. So, in this scenario, i.e., traffic to a blog-style website, becomes limited by some other component of the system, such as database, cache or proxy server, at an early stage.

Initially, also actual measurements of the power consumed by the hardware during the scenarios was also to be carried out, but as there was no sufficient measurement equipment available, this aspect is left for future research. In addition, as video traffic is becoming more and more dominant in the Internet, the energy efficiency of different schemes of streaming video to clients should be evaluated.

References

[1] "Cisco Visual Networking Index: Forecast and methodology, 2013–2018," White Paper, Cisco, Jun. 2014. [Online]. Available: http://www.cisco.com/

[2] A. Bielenberg, L. Helm, A. Gentilucci, D. Stefanescu, and H. Zhang, "The growth of Diaspora – a decentralized online social network in the wild," in *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, March 2012, pp. 13–18.

[3] J. Koomey, "Worldwide electricity used in data centers," *Environmental Research Letters*, vol. 3, no. 3, p. 034008, 2008. [Online]. Available: http://stacks.iop.org/1748-9326/3/i=3/a=034008

[4] ——, "Growth in data center electricity use 2005 to 2010," Aug. 2011. [Online]. Available: http://www.analyticspress.com/datacenters.html

[5] M. Peuhkuri, R. Lääkkölä, J. Costa-Requena, and J. Manner, "Datacenters – energy hogs or helping to optimize energy consumption," in *Smart Grid Technology, Economics and Policies (SG-TEP), 2012 International Conference on*, Dec 2012, pp. 1–4.

[6] "Key world energy statistics 2012," International Energy Agency, Paris, 2012. [Online]. Available: http://www.iea.org/textbase/nppdf/stat/12/kwes.pdf

[7] Data center images. [Online]. Available: https://www.flickr.com/photos/talk2stu/sets/72157608201783812

[8] K. Christensen, P. Reviriego, B. Nordman, M. Bennett, M. Mostowfi, and J. Maestro, "Ieee 802.3az: the road to energy efficient ethernet," *Communications Magazine, IEEE*, vol. 48, no. 11, pp. 50–56, November 2010.

[9] L. Barroso and U. Holzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, Dec 2007.

[10] D. E. Liddle, "The wider impact of Moore's law," *Solid-State Circuits Newsletter, IEEE*, vol. 11, no. 5, pp. 28 –30, sept. 2006.

[11] J. Koomey, S. Berard, M. Sanchez, and H. Wong, "Implications of historical trends in the electrical efficiency of computing," *Annals of the History of Computing, IEEE*, vol. 33, no. 3, pp. 46 –54, march 2011.

[12] "Telecommunications infrastructure standard for data centers," ANSI/TIA, Arlington, USA, Standard, Apr. 2005.

[13] "The Zettabyte era: Trends and analysis," White Paper, Cisco, Jun. 2014. [Online]. Available: http://www.cisco.com/

[14] J. Hamilton, "AWS innovation at scale," AWS re:Invent 2014 conference talk, Amazon, Nov. 2014. [Online]. Available: https://www.youtube.com/watch?v=JIQETrFC_SQ

[15] V. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z.-L. Zhang, "Unreeling netflix: Understanding and improving multi-cdn movie delivery," in *INFOCOM, 2012 Proceedings IEEE*, March, pp. 1620–1628.

[16] The Elisa Viihde time shifting service. [Online]. Available: https://elisa.fi/viihde/esittely/?

[17] The Sonera Viihde time shifting service. [Online]. Available: http://www.sonera.fi/tutustu+ja+osta/tv+ja+viihde/sonera+viihde

[18] G. Kreitz and F. Niemela, "Spotify – large scale, low latency, p2p music-on-demand streaming," in *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, aug. 2010, pp. 1 –10.

[19] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," in *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '05. New York, NY, USA: ACM, 2005, pp. 303–314. [Online]. Available: http://doi.acm.org/10.1145/1064212.1064253

[20] M. Vouk, "Cloud computing – issues, research and implementations," in *Information Technology Interfaces, 2008. ITI 2008. 30th International Conference on*, June 2008, pp. 31–40.

[21] Q. Huang, F. Gao, R. Wang, and Z. Qi, "Power consumption of virtual machine live migration in clouds," in *Communications and Mobile Computing (CMC), 2011 Third International Conference on*, April 2011, pp. 122–125.

[22] A. Vance, "Amazon's Cloud Is One of the Fastest-Growing Software Businesses in History," Bloomberg Businessweek, Jul. 2014.

[Online]. Available: http://www.businessweek.com/articles/2014-07-15/amazons-cloud-is-one-of-the-fastest-growing-software-businesses-in-history

[23] J. Novet, "DigitalOcean's cloud surpasses Amazon Web Services in one category," VentureBeat, Dec. 2013. [Online]. Available: http://venturebeat.com/

[24] LinuxContainers.org: Infrastructure for container projects. [Online]. Available: https://linuxcontainers.org/

[25] Docker - an open platform for distributed applications for developers and sysadmins. [Online]. Available: https://www.docker.com/

[26] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," *Linux J.*, vol. 2014, no. 239, Mar. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2600239.2600241

[27] B. Rhoden, K. Klues, D. Zhu, and E. Brewer, "Improving per-node efficiency in the datacenter with new os abstractions," in *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, ser. SOCC '11. New York, NY, USA: ACM, 2011, pp. 25:1–25:8. [Online]. Available: http://doi.acm.org/10.1145/2038916.2038941

[28] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM Journal of Research and Development*, vol. 5, no. 3, pp. 183–191, July 1961.

[29] C. Delimitrou and C. Kozyrakis, "The Netflix challenge: Datacenter edition," *IEEE Comput. Archit. Lett.*, vol. 12, no. 1, pp. 29–32, Jan. 2013. [Online]. Available: http://dx.doi.org/10.1109/L-CA.2012.10

[30] American Society of Heating, Refrigerating and Air-Conditioning Engineers, *Thermal Guidelines for Data Processing Environments*, ser. ASHRAE datacom series. ASHRAE, 2012.

[31] R. Tozer and M. Salim, "Data center air management metrics – practical approach," in *Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2010 12th IEEE Intersociety Conference on*, June 2010, pp. 1–8.

[32] M. Patterson, "The effect of data center temperature on energy efficiency," in *Thermal and Thermomechanical Phenomena in Electronic Systems, 2008. ITHERM 2008. 11th Intersociety Conference on*, May 2008, pp. 1167–1174.

[33] R. Zhou, Z. Wang, C. Bash, A. McReynolds, C. Hoover, R. Shih, N. Kumari, and R. Sharma, "A holistic and optimal approach for data center cooling management," in *American Control Conference (ACC), 2011*, June 2011, pp. 1346–1351.

[34] R. Zhou, Z. Wang, C. Bash, and A. McReynolds, "Data center cooling management and analysis - a model based approach," in *Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM), 2012 28th Annual IEEE*, March 2012, pp. 98–103.

[35] T. Malkamäki and S. Ovaska, "Data centers and energy balance in Finland," in *Green Computing Conference (IGCC), 2012 International*, June 2012, pp. 1–6.

[36] X. Lu, T. Lu, M. Remes, and M. Viljanen, "Energy efficiency assessment for data center in Finland: Case study," in *Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on*, June 2011, pp. 54–60.

[37] E. Jaureguialzo, "PUE: The Green Grid metric for evaluating the energy efficiency in DC (data center). measurement method using the power demand," in *Telecommunications Energy Conference (INTELEC), 2011 IEEE 33rd International*, oct. 2011, pp. 1 –8.

[38] G. Warkozek, E. Drayer, V. Debusschere, and S. Bacha, "A new approach to model energy consumption of servers in data centers," in *Industrial Technology (ICIT), 2012 IEEE International Conference on*, march 2012, pp. 211 –216.

[39] "New definitions for inclusion in recommendation P.10/G.100," ITU-T SG12, Geneva, Switzerland, Recommendation P.10/G.100 Amendment 2, Jul. 2008. [Online]. Available: http://www.itu.int/rec/T-REC-P.10-200807-I!Amd2/en

[40] M. Fiedler, T. Hossfeld, and P. Tran-Gia, "A generic quantitative relationship between quality of experience and quality of service," *Network, IEEE*, vol. 24, no. 2, pp. 36 –41, march-april 2010.

[41] J. Nielsen, *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[42] P. Pradhan, R. Tewari, S. Sahu, A. Chandra, and P. Shenoy, "An observation-based approach towards self-managing web servers," in *Quality of Service, 2002. Tenth IEEE International Workshop on*, 2002, pp. 13–22.

[43] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246 (Proposed Standard), Internet Engineering Task Force, Aug. 2008, updated by RFCs 5746, 5878, 6176. [Online]. Available: http://www.ietf.org/rfc/rfc5246.txt

[44] E. Rescorla, *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley, 2001.

[45] L. Zhao, R. Iyer, S. Makineni, and L. Bhuyan, "Anatomy and performance of ssl processing," in *Performance Analysis of Systems and Software, 2005. ISPASS 2005. IEEE International Symposium on*, March 2005, pp. 197–206.

[46] "Transport Layer Security (TLS) Parameters," Internet Assigned Numbers Authority, Aug. 2005, updated in 2014. [Online]. Available: http://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml

[47] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," in *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, Aug 2001, pp. 101–102.

[48] The BitTorrent P2P protocol. [Online]. Available: http://www.bittorrent.org/

[49] J. Brodkin, "Netflix researching "large-scale peer-to-peer technology" for streaming," Ars Technica, Apr. 2014. [Online]. Available: http://arstechnica.com/

[50] The Django web framework. [Online]. Available: https://www.djangoproject.com/

[51] "What powers Instagram: Hundreds of instances, dozens of technologies." [Online]. Available: http://instagram-engineering.tumblr.com/post/13649370142/what-powers-instagram-hundreds-of-instances-dozens-of

[52] The nginx HTTP server. [Online]. Available: http://nginx.org/en/

[53] The MySQL database. [Online]. Available: https://www.mysql.com/

[54] The Memcached memory object caching system. [Online]. Available: http://memcached.org/

[55] The Chef IT infrastructure automation suite. [Online]. Available: https://www.getchef.com/

[56] The Eventlet concurrent networking library for Python. [Online]. Available: http://eventlet.net/

[57] T. Brisco, "DNS Support for Load Balancing," RFC 1794 (Informational), Internet Engineering Task Force, Apr. 1995. [Online]. Available: http://www.ietf.org/rfc/rfc5246.txt

Appendices

A Detailed Load Time Graphs

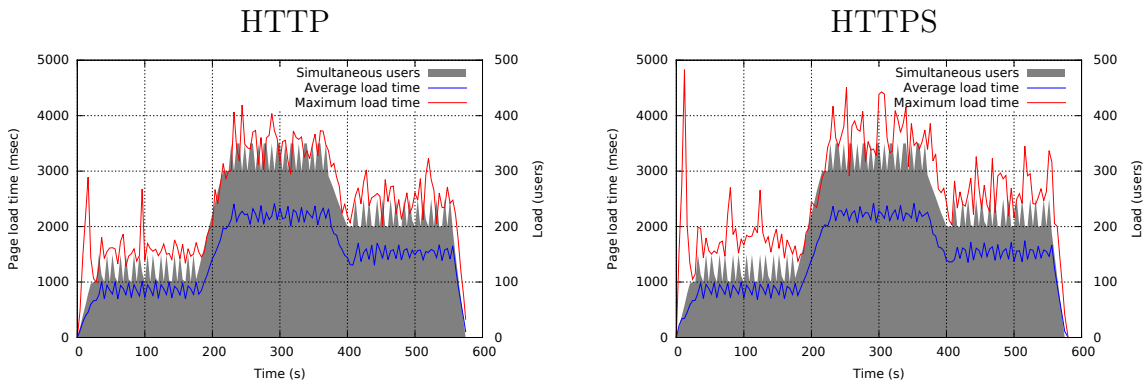

Figure A1: Load times for 1 app server.

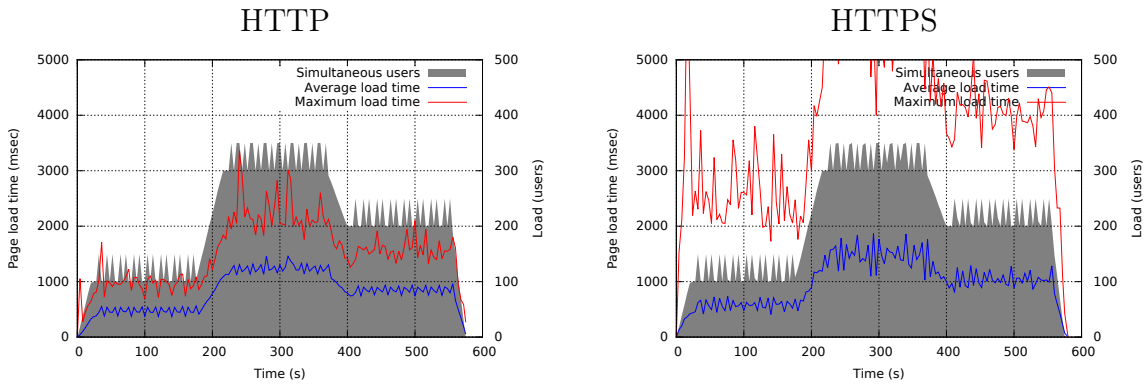

Figure A2: Load times for 2 app servers.

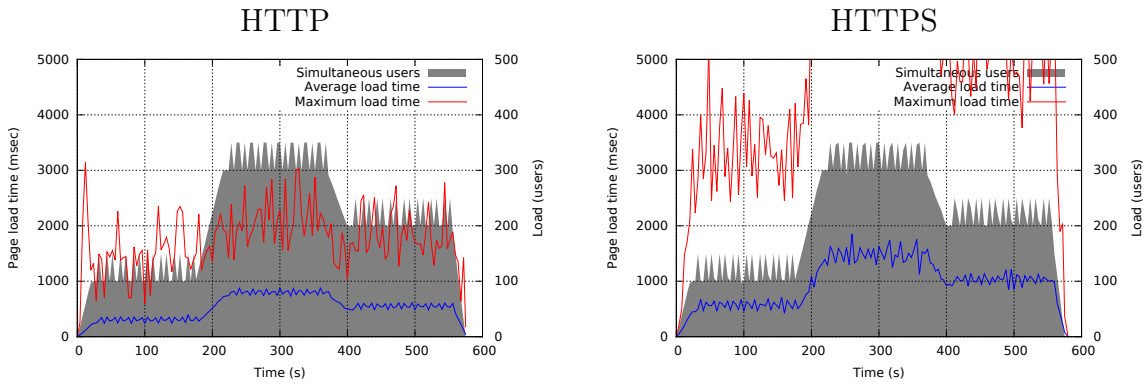

Figure A3: Load times for 3 app servers.

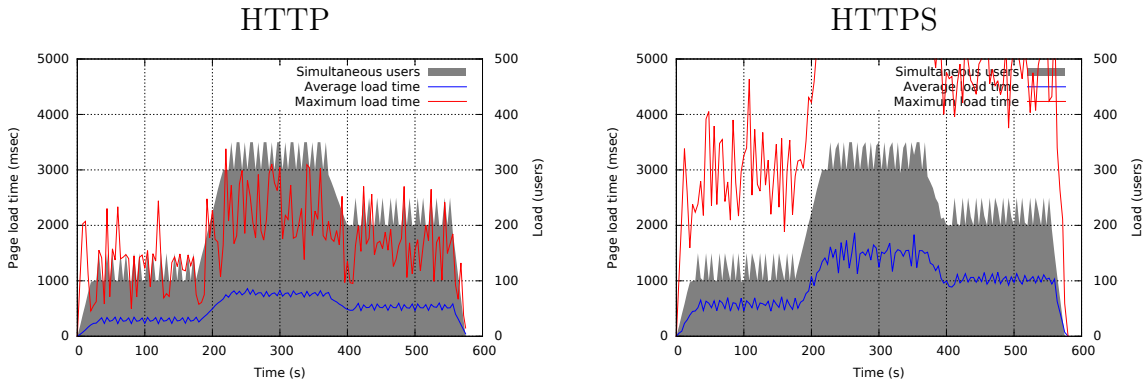

Figure A4: Load times for 4 app servers.

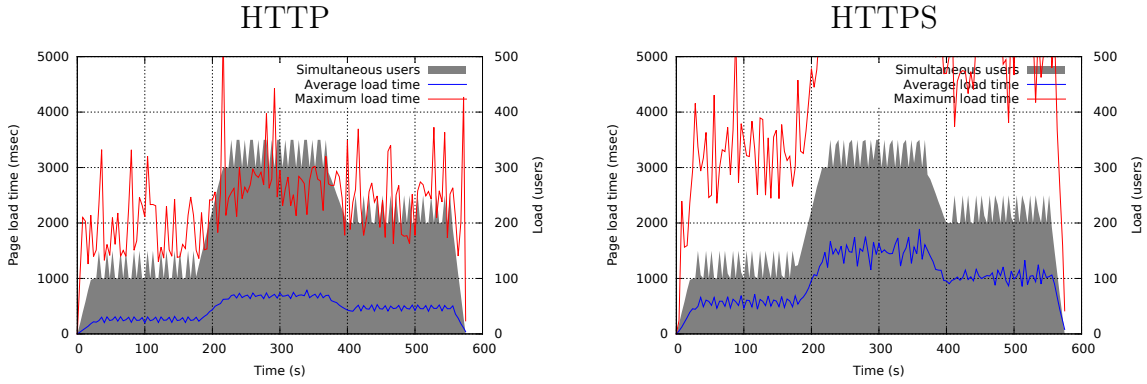

Figure A5: Load times for 5 app servers.

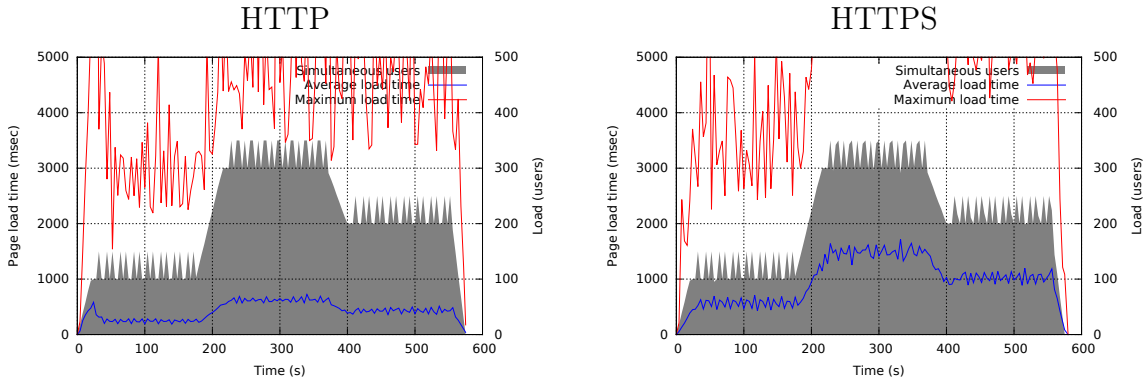

Figure A6: Load times for 6 app servers.

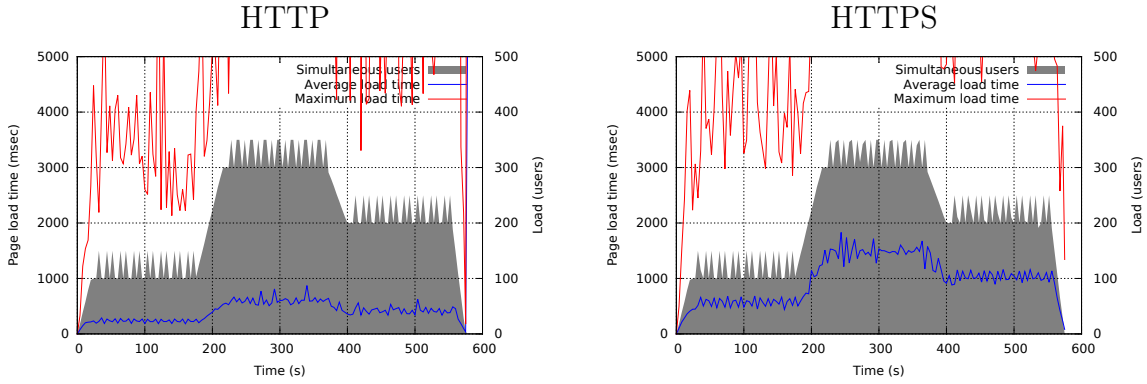

Figure A7: Load times for 7 app servers.

B Detailed First Byte Arrival Graphs

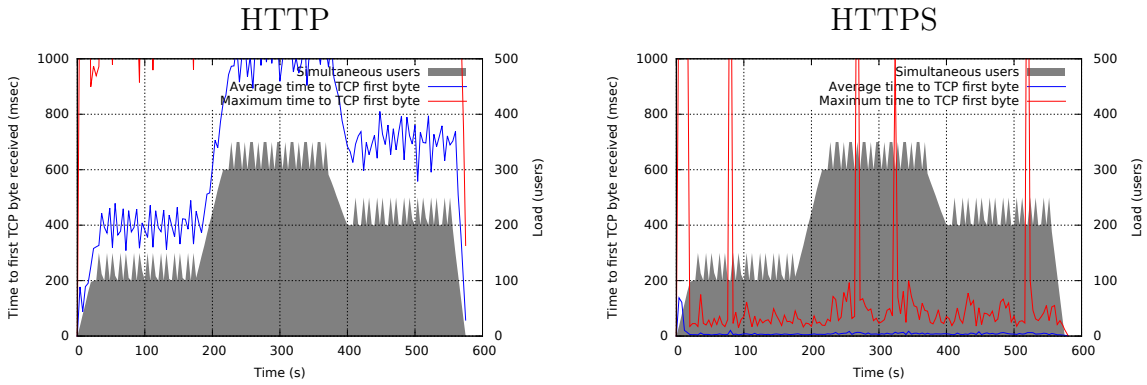

Figure B1: First data byte arrival times for 1 app server.

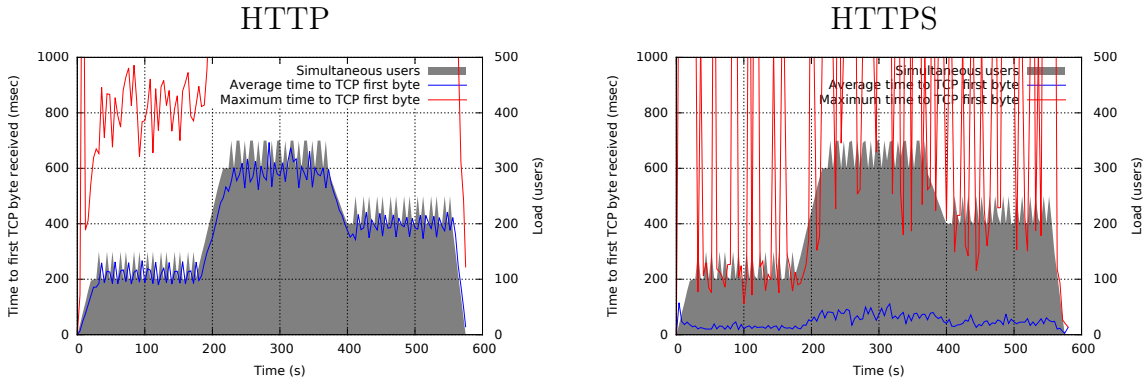

Figure B2: First data byte arrival times for 2 app servers.

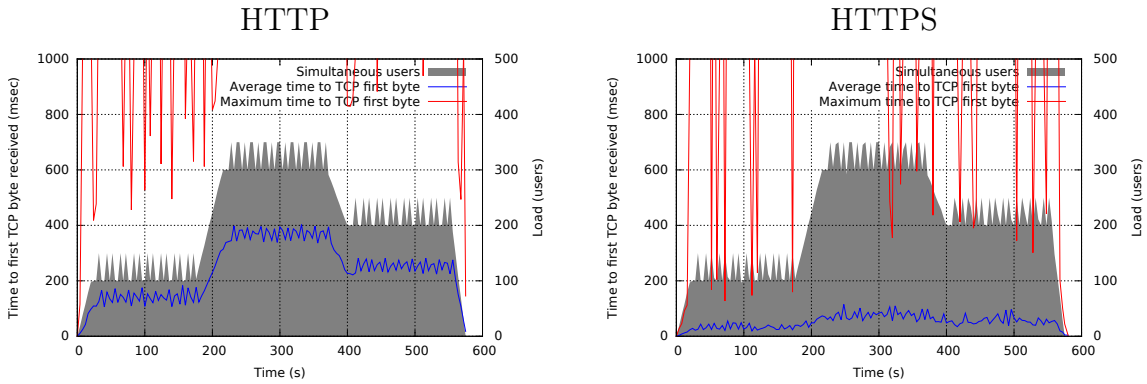

Figure B3: First data byte arrival times for 3 app servers.

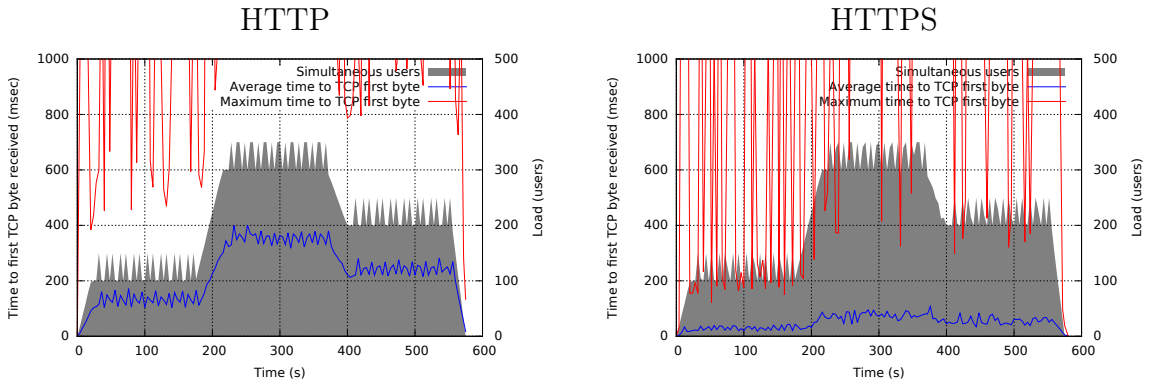

Figure B4: First data byte arrival times for 4 app servers.

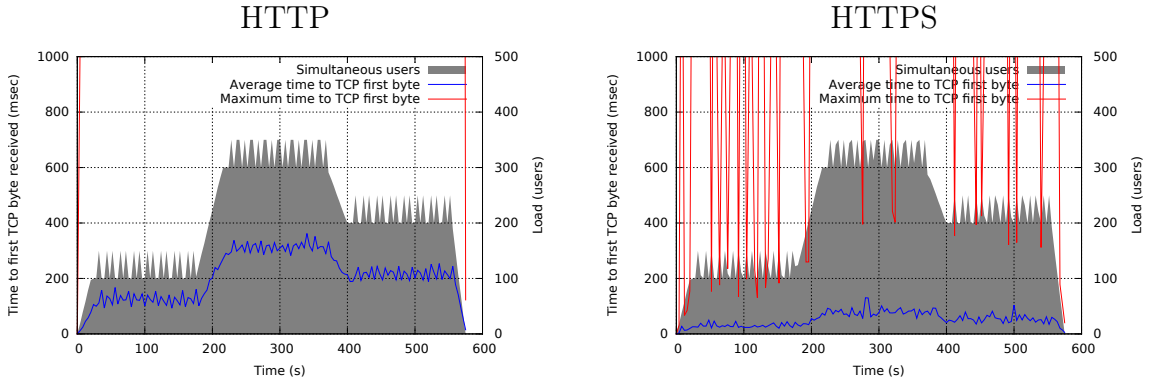

Figure B5: First data byte arrival times for 5 app servers.

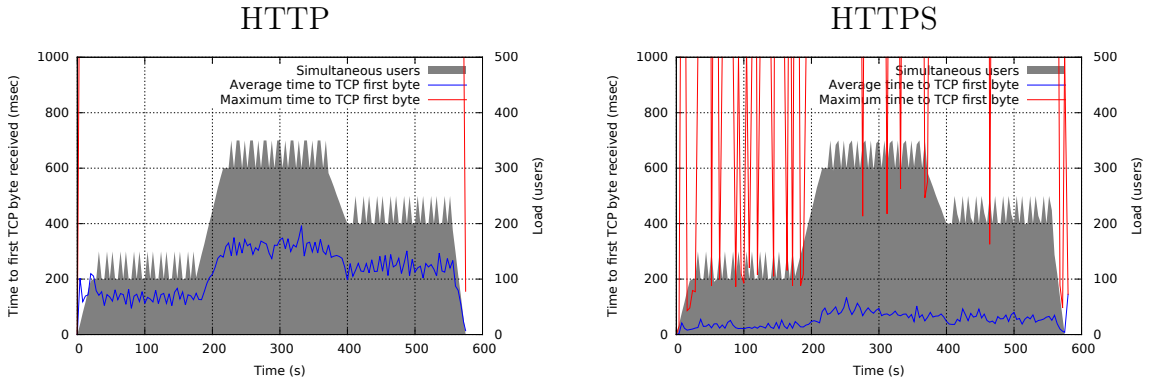

Figure B6: First data byte arrival times for 6 app servers.

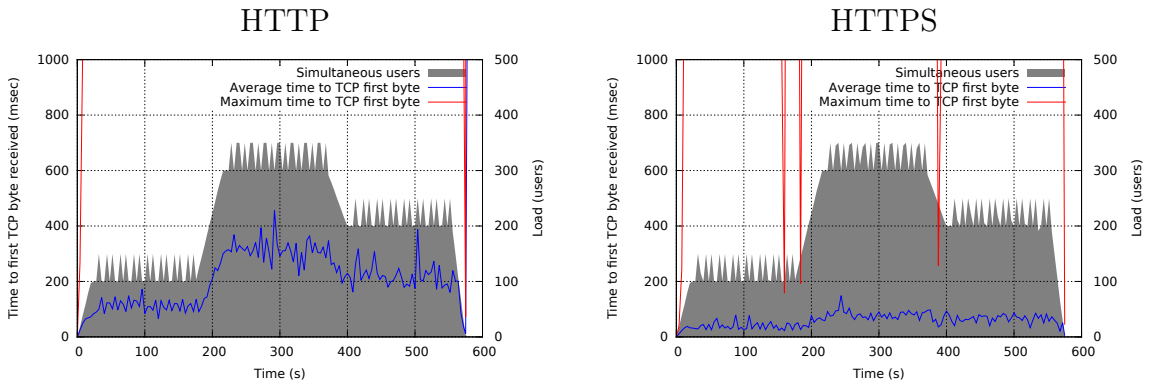

Figure B7: First data byte arrival times for 7 app servers.