Acknowledgements

We would like to express our gratitude towards the following persons who facilitated the composition and completion of this master thesis. First of all:

⋆ **Gunnar Brataas (IDI, NTNU)**, our adviser, for excellent guidance and constructive feedback throughout the project.

⋆ **Peter Hughes (IDI, NTNU)**, our co-adviser, for value inputs throughout the thesis.

In addition we appriciate the value inputs and moral support from the other performance evalution students at IDI: Erik&Erlend&Geir.

Trondheim, 22nd June 2005

Preface

This paper is a master thesis conducted at IDI, NTNU during the spring 2005. The overall objective in this paper is to look at the technical factors and the cost involved in assesing the scalability for large scale Internet sites.

Below is a short summary of each chapter in our master thesis.

Chapter 1 - Introduction: This chapter introduces the problem definition and the preliminary work.

Chapter 2 - Method: This chapter will present the concepts of the Structure and Performance Specification method (SP-method), which is the method we use to asses the scalability of the system.

Chapter 3 - System: In this chapter we will present the system used for our scalability experiment, both the hardware and the software used. This is done so that the experiments can be easily reproduced either on the same hardware, or on different on hardware.

Chapter 4 - SP: In this chapter the system introduced in Chapter 3 is presented with the SP notation, described in Chapter 2. We will first explain the operations on the components and then explain the connections between the components, which are described as complexity functions.

Chapter 5 - Scalability: This chapter will present the concepts of scalability, and outline the likely scaling scenarios in our thesis.

Chapter 6 - Measurements: This chapter will explain how the measurements of the components are performed, discuss relevant issues connected to this, and give an outline of a measurement plan.

Chapter 7 - Resource Function Workbench: This chapter will present a framework for automatization of the measurements needed in our thesis. This includes both the collection and the post-prosessing of the results.

Chapter 8 - Results: In this chapter we will present an overview of the results from our experiments.

Chapter 9 - SP-parameterization: In this chapter we will parameterize the SP-model identified in Chapter 4, based on the results of the measurements presented in Chapter 8.

Chapter 10 - Dynamic model: In this chapter we will construct and solve a dynamic model for our systems.

Chapter 11 - Discussion: In this chapter we will discuss the main findings in our thesis, and look at them in connection with the scaling scenarios outlined in Chapter 5. We will also explain the implications these findings have on read-intensive web sites like TV2i.

Chapter 12 - Method evaluation: This chapter will evaluate the work and the methods used in this thesis.

Chapter 13 - Conclusion: This chapter gives a summary of the important findings and the evaluation, and highlight areas for further work.

Appendix: The Appendix consists of four chapters:

- The first chapter presents a survey, which investigated the likely growth scenarios for web sites, and it was conducted by interviewing some of the major news sites in Norway.
- The next chapter describes a classification of open source load generators. This was used to find suited load generators for this thesis.
- The third chapter in the appendix, is a detailed walkthrough of the files scripts in connection with the resource function workbench.
- In the fourth chapter in the appendix, we have included three of the most important scripts in our RFW.
- In this last chapter, we presents all the measurement for our systems.

Web site: In connection with this thesis we have produced a web site, available on *http://www.stud.ntnu.no/groups/diplomgisleruud*. This web site contains this report in an electronic format, all the measurements from our experiment, the spreadsheets used and the script used in our RFW.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The trend for read-intensive news sites is to use more and larger multimedia content, especially images. This is driven by the fact that more and more readers use broadband to access the Internet, and in addition the screen sizes have gone up. Increased article size could possibly have wide implications for the web sites, and they should know how this affects their systems. How will improving the hardware of the system help them to cope with these changes?

This paper will look into the technical factors and the cost involved in assessing the scalability for large scale Internet sites. We use a system which resembles read-intensive large-scale Internet sites, to perform this scalability exploration.

This chapter introduces the problem definition and the preliminary work.

1.1 Problem definition

In this section we will present the final version of the problem definition. This version has some slight modifications from the original one, which can be found in Appendix F. The problem definition has been made in cooperation with Gunnar Brataas.

> "In the Ruud & Tveiten project [21] the overall objective was to study the practical and economic feasibility of exploring the scalability of a large-scale read-intensive Internet site for TV2 Interaktiv. Using the SP terminology, a limited number of classes of complexity functions were identified, serving to structure the scalability assessment method. However, the resulting method was not validated with actual measurements.
>
> The main objective of this thesis is to continue the work started in [21]: i.e. to explore the practical and economical feasibility of assessing the scalability of a read-intensive large-scale internet site. Since the TV2 Nettavisen infrastructure was hard to measure, the essence of the infrastructure has to be reproduced in a controlled environment. For practical reasons Open Source Software will be used together with the Clustis2 computational cluster of PCs. The similarities and discrepancies between TV2i's infrastructure and the selected system should be described.

The static properties of the test baseline will be modeled using the SP method. Scaling scenarios will drive the scalability exploration and should be described in detail. In practice, strict and uniform scaling is rare. Some scaling scenarios should therefore consider differential and non-uniform scaling.

For each class of measurements a test plan should be both described and validated by measurements at the level of detail which makes it possible for others to revalidate the actual measurements performed. It is still an open question if it is possible to parameterize a complete scalability model, or if only some examples of each complexity function class should be measured. If a complete set of measurements is performed, the resulting static model should be validated in combination with a dynamic model.

To measure the components needed in the SP model will in the general case require a Resource Function Workbench [25], a tool which aids in the actual measurement and makes it easier to reuse the complexity functions. To develop this is a useful result in its own right. To describe and develop a suitable Resource Function Workbench is in this diploma a secondary objective."

1.2 The TV2i system

In our project [21], we studied TV2 Nettavisen. This is a large-scale read-intensive Internet news site, currently the third biggest site in Norway with just under one million weekly readers [17]. Figure 1.1 shows an overview of the major component in the TV2i system.

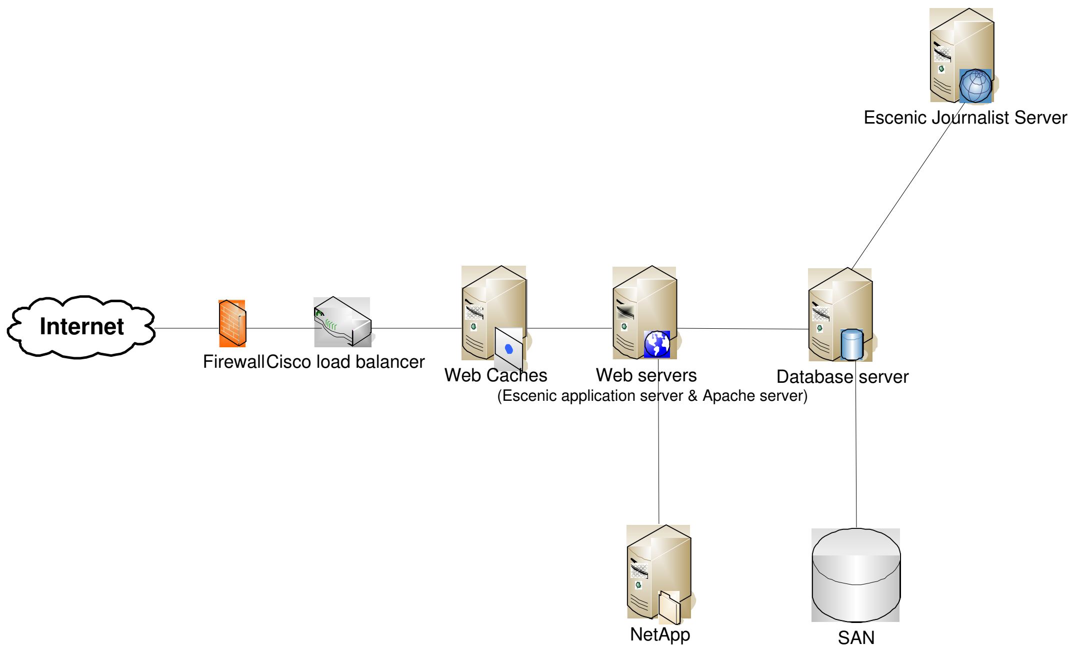


Figure 1.1: The components in the TV2i system

TV2i configuration

All the servers in the TV2i system run Red Hat Linux, and the local network is Gigabit Ethernet.

The web caches in the TV2i system are running Oracle software, and are configured on 3 computers. The web caches speed up the response time for the end-users and minimize the number of requests to the back-end server. The web caches have a hit ratio of around 92%, which is very high. Therefore only a small number of requests hit the back-end server.

Apache is used on the web server. Its job is to serve HTTP requests from the web caches, and to ask the application sever (the Escenic end-user server) for a static file or to retrieve an image from the NetApp and return it via the HTTP protocol back to the web caches. Escenic, which is a content management system, is running in cooperation with Apache. The main task for the Escenic End-user Server is to make queries to the database, and convert this information into a static format (HTML), which in turn is served to the Apache web server. A JDBC driver is also running on the web server. This driver is an open connection between the Escenic End-user server and the database, and improves the performance of this connection.

NetApp is a network storage solution, which is connected to the web servers through a 1 Gigabit local network. It stores the images and graphics used on the web site. It consists of an unknown number of disks, connected together as a RAID solution.

TV2i uses a storage area network (SAN) to allow multiple hosts connect to multiple storage devices. This means that more than one database server can work against the same data. The SAN solution uses a fiber channel to connect the Oracle database servers to the SAN. The Oracle database servers run on two Dell servers, currently only one of the two servers is in operation, with the other in cold standby.

The Escenic Journalist Server is the server the journalist applications interact with, to update and publish articles. In our project, we decided not look further into this part of the system, and rather have a customer focus. The main reason for this is because there are so many more users than journalists in the system.

SP-model of the TV2i system

Based on the SP model of the TV2i system described in [2], we studied in our project the important components in the TV2i system and their connections, and looked for possibilities to simplify the model. In [21] we found that the TV2i system is a large and highly complex system, so that further limitations were needed to make the scalability assessment of the system feasible.

In [21] the system was divided into two main parts, one serving the end-users and one serving the journalists. The focus was on the end-user part, and the reasons for this were that the end-users are the customers, and that they have a higher demand for response time. In addition there was expected a high growth in the number of end-user, and not in the number of journalists. The final SP model of the TV2i system in [21] is shown in Figure 1.2 on next page. This model shows the components of the TV2i system and the connection between them, and a further description the modeling concept is described in Chapter 2.

Because of TV2i's focus on growth in end-users, we have tried to simulate a read-intensive news site, so that our results can be applied to TV2i end-user part.

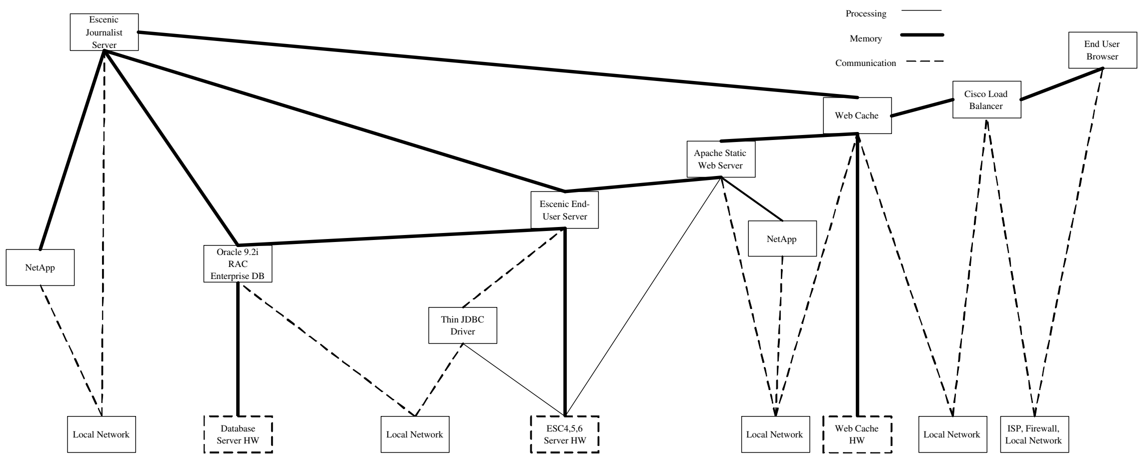


Figure 1.2: SP-model of the TV2i system

Chapter 2

Method

This chapter will present the concepts of the Structure and Performance Specification method (SP method), which is the method we use to asses the scalability of the system.

2.1 Structure and Performance model

The system in this thesis is described using the Structure and Performance Specification method (SP method). This method is a conceptual framework that describes the software and hardware components in the system, and the relations between them. The SP method was developed by Peter H. Hughes in 1988 [9], and its purpose is to make it easier to work with load and performance analysis.

2.1.1 Introduction

SP deals with hierarchical models that span from software on the top to hardware at the bottom. To illustrate the SP-model we can imagine that we have a large system. This system can be considered as a "black box", because we do not know anything about the components it contains. The SP-model aids us to open up this "black box", so we can examine the subcomponents of the system. These subcomponents can be systems, which can be further broken down into new subsystems. Finally, we will be left with components that can not be further decomposed, and these are called devices (hardware). This system breakdown makes the performance analysis easier, as it is easier to calculate the performance on a subsystem than on the overall system. The overall performance can then be found through aggregating the performance of the subsystems.

2.1.2 The SP framework

The SP framework consists of boxes, and lines representing relationships between them (see Figure 2.1 on page 6). The boxes represents both hardware and software components in the system. Each of the boxes has one or more operations, represented by a circle with the name of the operation.

There are software components on all the levels in the hierarchy, except on the bottom

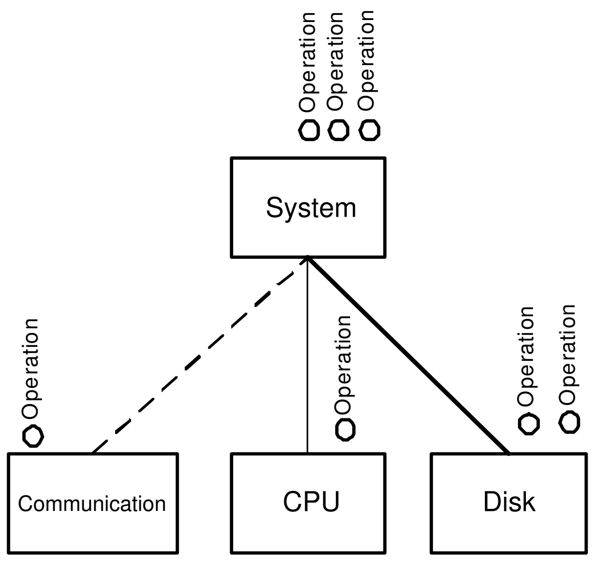


Figure 2.1: Example of a SP diagram

layer where there are only hardware devices. These software components both offers and requires services (operations). The hardware devices offer services, and do not require services from any other components. The hardware devices are components like CPU, disk and network.

The lines are relations between the services the components require from the lower level components, and offer to the higher level ones. In Figure 2.1 *CPU* offer an operation, while *System* requires (invokes) an operation from the *CPU*.

There are 3 different types of relations between the boxes. The thick line illustrates memory storage relations, the thin line process relations, and the dash-dot line communication relations between the boxes. Software components have at least a processing and a memory relation to a component on a lower level in the hierarchy.

2.1.3 Complexity matrix

The SP framework identifies all the components that are involved in the system. To specify the system further, we need identify the relationships between the components. This is done by mapping the operations offered on the lower level with the operations required on the higher level. This is accomplished using complexity matrices. The operations on the higher level are written in the first column, and the operations offered from the lower level are written in the first row. The operations on each component can be real-world operations, but do not need to be real-world. Some operations can be aggregated for the purpose of the SP-modeling.

Each line in the SP-model will be represented by a complexity matrix. In Figure 2.2 module A has six operations that invoke the four operations offered by module B. If the operation A1 uses operation B2 twice, the number 2 should be written at the intersection between the two operations. Vetland et al. claims in [22] that ”each row in a complexity matrix is a vector that represents the mean number of times each service of the sub component is invoked as a consequence of an operation of the superior component being invoked.”

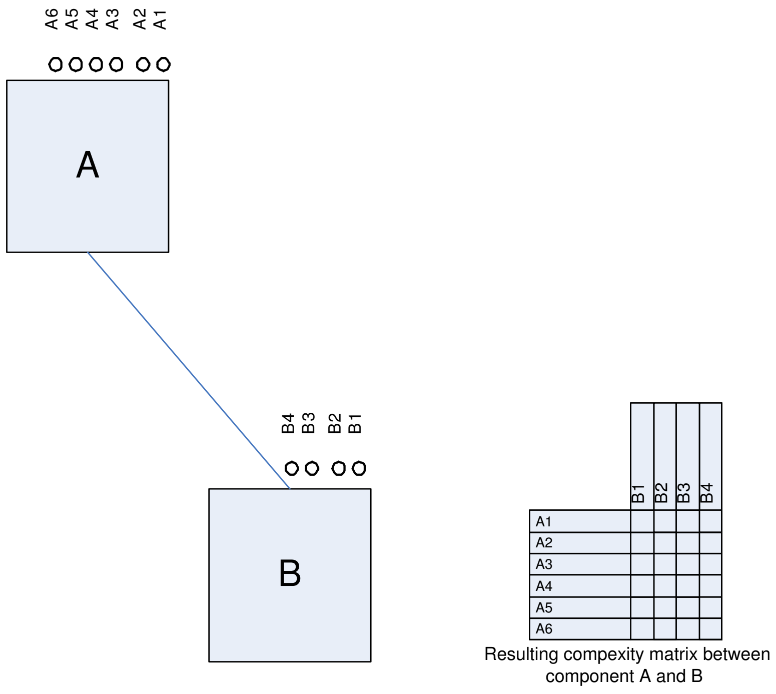


Figure 2.2: Example of a complexity matrix

The matrix element c_{ij} can either be zero, a constant, or a function depending on some parameters. The elements that depend on a number of parameters are called complexity functions. Complexity functions can result in a non-linear scaling effect for the relations, which is of special interest in a performance analysis view.

2.1.4 The static model

A static model describes how the services offered by the system are being used. The SP-method specifies the performance-related attributes of any software component independently of its environment and independently of any particular workload. The static model requires that the work (e.g. the services required) and load (e.g., arrival rate) are separated. We can summarize the SP method in three steps:

- First the system is divided into logical components, and then the relations between the individual components are determined.
- Secondly the services on each component are identified.
- Finally, the properties of the connections are described as matrices. The matrices describe the relations between each connected component's services. When multiplying all these matrices, we get the top level service demand.

2.1.5 Dynamic model

The dynamic model will be thoroughly described in Chapter 10, but we introduce the connection with the static model here. A static model can not alone answer many questions related to scalability, it is only when it is used in connection with a dynamic model it can answer scalability questions and be used for predictions. While the static model captures the work involved in performing the operations, the dynamic model enable us to model how the load (e.g. different number of users) will affect the system. The static model provides parameters to the dynamic model via mapping of service demands to resource demands. Figure 2.3 shows the connection between the static and dynamic model.

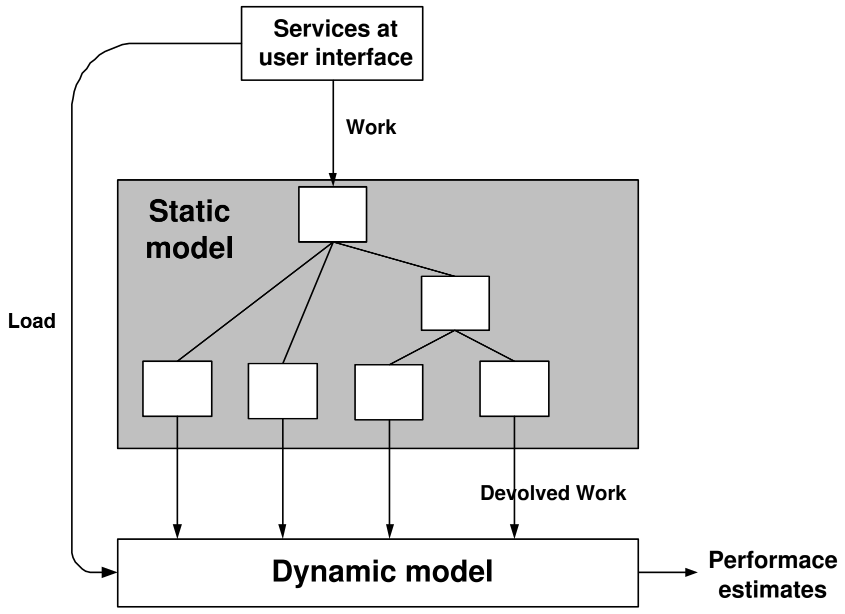


Figure 2.3: The relationship between the static and dynamic model

The load parameters are applied directly to the dynamic model while the mapping between offered services (operations) and resources are modeled by the static model. In a scalability study the dynamic model is made to investigate and experiment with the effects of competition of computer resources.

The dynamic model can either be represented state-based or object-based. The state-based is more solution-oriented than the object-based which is more descriptive. The state-based representation is illustrated by modeling an open source queuing-network, representing the servers and queuing centers of the system. while the object-based representation is illustrated by e.g. activity diagrams.

Chapter 3

System

In this chapter we will present the system used for our scalability experiments, both the hardware and the software used. This is done so that the experiments can be easily reproduced either on the same hardware, or on different on hardware. Section 3.4 will explain how this system resembles the system originally explored at TV2i.

3.1 Introduction

Our mission is to measure on a system fairly similar to the TV2i system, to prove the feasibility of our methodology, so that they can apply it to their system.

To continue the work started in our project work, we have installed a system that resembles the TV2i system. Our system is based on open source products and has fewer components compared to the TV2i system shown in Figure 1.1 on page 2, but all the significant components in the TV2i system are replaced by nearly equivalent components in the new system, except the web caches. Although the new system is highly simplified, the methodology of performance and scalability assessment of the systems will be similar. The major components of the new system are shown in Figure 3.1.

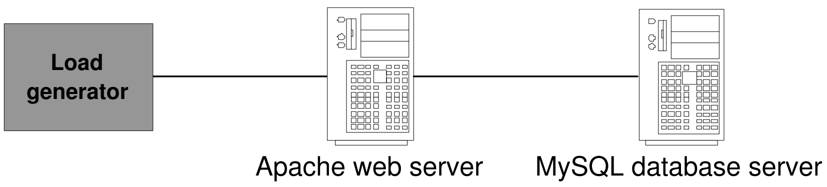


Figure 3.1: The system used in our experiment

3.2 System relationship

This section will explain how the three components in our system work together. Our system works in the same way as a news web site, where the users request articles (news). Because of this, we will first explain the different parts of an *article* in our system.

Article defined

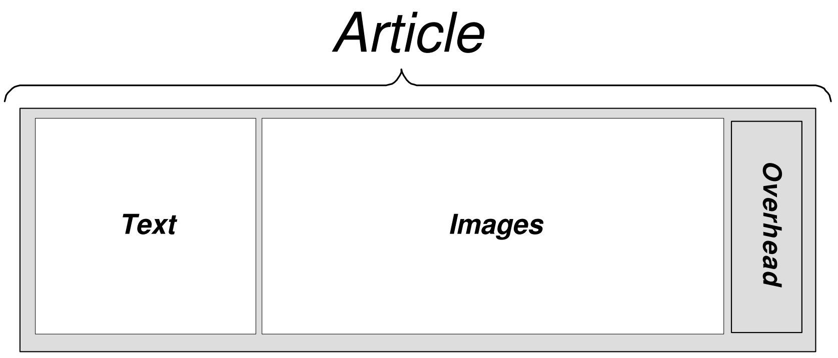


Figure 3.2: The parts of an article in our system

As we can see in Figur 3.2, an article consists of three main parts:

- **Text:** This is the part written by the journalists, and it contains the actual news content. It is stored in a MySQL database on the database server disk. All articles must contain a text part.
- **Images:** An article can contain zero or more images. The number of images is determined by the number of links to an image in the text part. The images are stored on the web server disk.
- **Overhead:** This is the overhead in the article, that is additional information mainly used internally by the web server and the web browser. This is typically some of the HTML code (layout information). On real news sites this overhead can be fairly large, but in our system it is very small, and thus we will not look into it.

Article request

The sequence diagram in Figure 3.3 on page 11 explains how and in which order the different components are involved, when the load generator requests an article. This sequence diagram is slightly simplified , as we have not included the establish and release connection phases.

1. The load generator requests an article from the web server.
2. As an article always contains a text part, the web server requests the text from the data base server.
3. The database processes the request, and text part is returned to the web server. The web server then returns it to the load generator.
4. The load generator sees if there are any image links in the text. And if it is, it sends out one request for each image link.
5. The web server processes the image requests, and return the images.

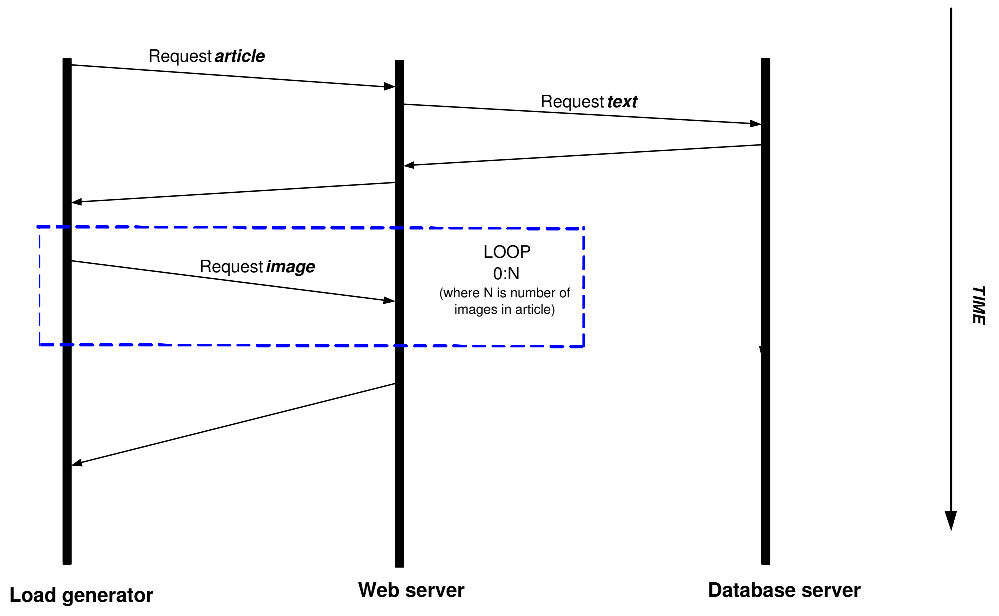


Figure 3.3: Sequence diagram of our system

6. Article request complete.

Note: When we are measuring on the components we have to seperate the system. So we are either just requesting images or just text.

3.3 System configuration

This section will explain our systems in detail. Subsection 3.3.1 will introduce the hardware used, while subsection 3.3.2 will explain the software used.

3.3.1 Hardware configuration

The systems are configured on Clustis2, which is a computational cluster, owned by IDI, NTNU. It consists of 21 computation nodes, which are coordinated by a master node[1]. There are two types of nodes in this cluster, 17 identical new ones, and 4 old nodes. Both types of nodes in the cluster are running the same version of Red Hat Linux. The nodes in the cluster are connected through Gigabit Ethernet. This network is private to the nodes, and only accessible through a front-end/login server.

Originally, we expected to use the old nodes in the cluster, but unfortunately the network card in these nodes did not function properly under heavy load. In addition, these old nodes were down for a long period of time. Our solution to these problems was to get a node with similar hardware specification as the old nodes but with 1Gbit/s network card[2].

[1]For more information about Clustis2 see their website: http://clustis2.idi.ntnu.no
[2]This card was bought and installed by us.

There are some small configuration difference between our old node and the old Clustis nodes. Our old node is a standalone node, on the same subnet, with the same operating system. However, a shared disk area, available in Clustis is not available. This has complicated our process a bit, as the disk area contained the system files, and these had to be copied over to the old node. This should not affect the experiments.

The old servers (our node in Clustis) in our test system, have the following specification:

- A single AMD Athlon MP 1600+ (clocked at 1.4 GHz)
- 1GB of RAM
- A 18GB SCSI disk[3]
- 1GBit network card
- Linux kernel 2.4.21-15.EL

The new servers (nodes in Clustis2) in our test system have the following specification:

- A single 3.4 GHz Intel P4 processor
- 1GB of RAM
- A 36GB SCSI disk[4]
- 1GBit network card
- Linux kernel 2.4.21-15.EL

We will refer to the new servers as Clustis2 nodes, and the old one as Clustis node.

3.3.2 Software configuration

Our system is configured with a web server, a database server and a load generator. An overview of the software used, can be seen below. A more detailed description of each component is described later in this section.

- *Apache web server*
  - Version: httpd-2.0.52
  - Downloaded from: http://httpd.apache.org/download.cgi
- *PHP*
  - Version: 4.3.10
  - Downloaded from: http://www.php.net/downloads.php
- *MySQL*

[3]www.seagate.com/cda/products/discsales/marketing/detail/0,1081,336,00.html
[4]www.seagate.com/cda/products/discsales/marketing/detail/0,1081,541,00.html

  - Version: mysql-standard-4.1.9-pc-linux-gnu-i686
  - Downloaded from: http://dev.mysql.com/downloads/mysql/4.1.html

- *Hammerhead2*
  - Version: 2.1.3
  - Downloaded from: sourceforge.net/project/showfiles.php?group_id=15206&release_id=42907

Apache web server

We use the Apache HTTP web server (Apache), which is the most popular web server[5] on the marked. Apache is developed as a HTTP web server for various desktops and operating system. The goal for Apache is to provide a secure, efficient and extensible server which provides HTTP services in sync with the current HTTP standards.

More information about the Apache project can be found on: *http://httpd.apache.org*

Out of the box Apache is not able to handle more than 256 simultaneous requests, which is not sufficient for our experiment. To overcome this problem we increased this limit in the Apache configuration file.

PHP

Together with the Apache web server we use PHP. In this experiment PHP acts as the content management layer. PHP is an open source server-side scripting language for creating Web applications. It is the most popular dynamic Web content technology for use with Apache servers. PHP offers also excellent connectivity to most of the common databases, including MySQL as we use. PHP was downloaded as source code, and compiled into binary files, with no special options, and then copied into the Apache folder. We use it to fetch text from the database and build it into HTML and deliver it to the Apache server.

More information about PHP can be found on: *http://www.php.net*

MySQL

The system is configured with a MySQL database server, which is the world's most used open source database server. The database is easy to customize and configure, and is also known for its high performance and reliability. The MySQL comes as a binary file, and it was set up using the installation guide.

More information about MySQL can be found on: *http://www.mysql.com*

Load Generator

A load generator is used to generate a large number of HTTP requests to the web server. There are many such tools available on the marked today, and much work has been put

[5]Apache serves more than 40 million sites, which is around 70 percent of the available web sites (According to the February 2005 Netcraft web server survey)

into choosing the one. We were dependent on a well working load generator to get reliable and reproducible measurements.

To choose the best suited load generators, we tried to classify the ones available by some requirements, both to get an overview over the available load generators on the marked, and to aid in choosing the ones to look further into (see Appendix B). After this classification, we were left with a few promising alternatives. Grinder was the one which was initially chosen, with Siege and Hammerhead2 close by.

During the initial phase of our project we used *Grinder* as load generator. Grinder was our first choice, as it has previously been used successfully in similar projects at NTNU. Another reason for choosing Grinder was that it is fairly simple to write system specific test in JYTON (Java PYTHON). The Grinder is a downloaded as a jar file (Java Archive file), and was trivial to install and well documented.

Unfortunately, Grinder proved to be highly CPU and memory intensive. It handled a "few large" connections very well, but was unable to generate many fairly small connections. Even when running Grinder on 6 nodes in parallel, we were not able to generate a high stable number of connections. Many of the servers collapsed under its own load, which made the load on the target machine drop abruptly. Because of this, another load generator was needed. It was clear that something written in a programming language like C, which is the native language on Linux, was preferable, since it runs much more efficient than a JAVA solution like Grinder.

Siege and *Hammerhead2* came out favorable from the load generator classification (see Appendix B), and both of these tools were installed and tested thorough.

Siege is an HTTP regression testing and benchmarking utility. It was designed to let web developers measure the performance of their code under stress, to see how it will stand up to load on the internet. It is written in C, and designed to be used on a Linux platform, although it has been successfully ported to other UNIX flavors. Siege is easy to install and use, but unfortunately Siege had some flaws. It was hard to control what resources it requestet and in addition the output produced by the program was poor.

Hammerhead2 is a stress testing tool designed to test out web servers and web sites. The rate at which Hammerhead2 attempts to pounds a site is fully configurable. It manages to create a stable number of request on the system. There are however, some small statistical variation, but the number of users in the system are more or less constant.

Hammerhead2 is not as trivial to install as Siege, its documentation is not as well written, and there have not been released a new version in 2 years. However, Hammerhead2 works very well in practice.

One of the strengths with Hammerhead2 is that the behavior is very configurable. Hammerhead2 loads a set of scenarios, and these scenarios states what resources that should be targeted on the target machine.

More information about Hammerhead2 can be found on: http://hammerhead.sourceforge.net

3.4 Similarities and discrepancies from the TV2i system

The new system in this thesis is meant to reflect the TV2i system, and although the final result of our study can not be directly used for the TV2i system, the method (SP) and the approach for deciding scalability problems can however be directly applied to the TV2i system. This is because the similarities between the systems are sufficient.

At first sight it is not easy to see the similarities between a highly costly and complex system like the TV2i system, and our three servers with open source software. This section will give a thorough description of the similarities and discrepancies between the two systems.

The main difference between the systems is the degree of complexity in terms of; costs, number of components and software. Their basic function is however the same. They both serve end-users with static web pages through a Web server (Apache) and both systems fetch the content for the web pages from a database and convert it into HTML, in other word they are both a *read-intensive website*.

As was shown in the Ruud&Tveiten project [21], many of the components in the TV2i system could be ignored in a scalability study. The reason these components could be dropped is because they were not regarded as a possible scalability problem. The major components in the initial TV2i system are illustrated in Figure 1.1, while Figure 3.4 shows the components we ended up with looking at. If we compare this to the major components in the new system (see Figure 3.1), we see the similarities quite clearly. We are not looking at the Web caches in this study, although the web caches is an interesting component, it is hard to recreate these in our system (because of license cost). As long as the cache hit on the web caches is relatively constant, the scalability of the web caches is independent of the scalability of the web system.

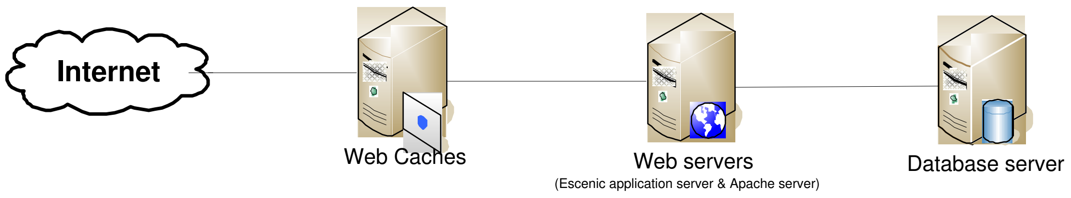
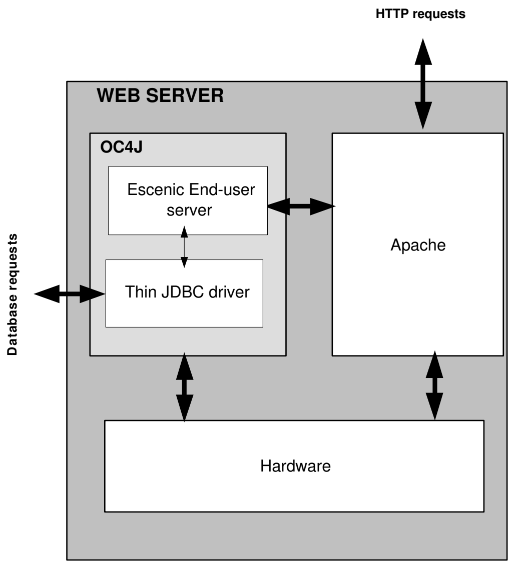


Figure 3.4: Major components in the TV2i system

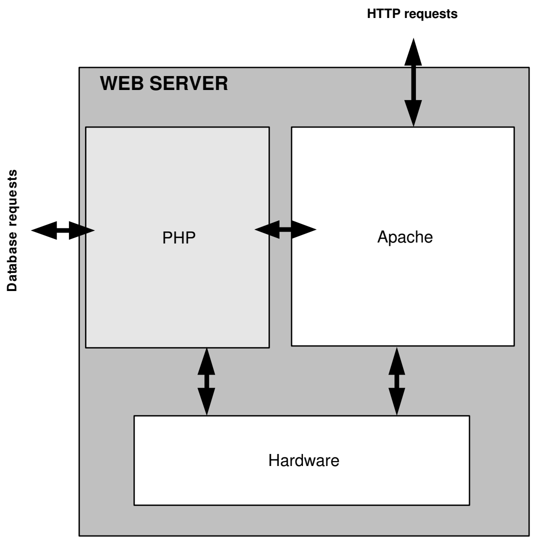
In the TV2i system the web server ran a content management solution called Escenic based upon Java, but we use PHP. The different between the two solutions can be seen in Figure 3.5 on next page. As we can see there is not a big difference, except that the complexity in Escenic is believed to be larger than in PHP. The measurement techniques used on our web server could be applied to the TV2i server.

The database server in the TV2i system was an Oracle Real Application Cluster, but we have chosen to use MySQL. The approach for deciding scalability problems is however believed to be the same. The Oracle RAC is much more complicated, as it consists of several nodes attached to a SAN solution.

What we have ended up with, is a system which is similar enough to the TV2i system, so that the approach can be directly applied in the TV2i system. The software components are not the same and the components in our system is less complicated. But the



(a) The configuration of the TV2i web server



(b) The configuration of the new web server

Figure 3.5: Comparison of our web server and the TV2i web server

overall approach for deciding scalability problems will be the same, and this increases the feasibility for TV2i to use the SP method for assessing the scalability of the system.

Chapter 4

SP

In this chapter the system introduced in Chapter 3 is presented with the SP notation described in Chapter 2. This terminology is used to break down the system in order to perform a scalability analysis on it. Figure 4.1 on next page shows the system presented with SP notation. We will explain the operations on the components and then explain the connections between the components. For a deeper introduction to the relationship between the web server and the database and how the articles are build up, see Chapter 3.2.

There are three main components in our system (introduced in Chapter 3.3); the load generator, the web server and the database server. Each of these components offers one or more operations. In addition the system has components on a lower level, namely CPU, disk and the network. All the operations shown in Figure 4.1 are explained in the following subsections.

4.1 Operations

4.1.1 Load generator

The load generator sends *Request Article* to the Apache web server through the local network (LAN), and it is the only operation on the load generator. The *Request Article* operation will always invoke one *Get Text* operation on the web server. In addition, one *Request Article* will invoke no, one or several *Get Image* operations. This is dependant of how many images the article contains, and all this is further explained in Chapter 3.2.

4.1.2 Web server

On the Apache web server component there are two operations, namely *Get Text* and *Get Image*. When the web server receives a *Request Article* from the load generator it determines which of the *Get Text* and *Get Image* operation to use. The *Get text* operations will invoke the database, where the **text**s are stored. The *Get image* operations invoke the CPU and disk on the web server, where the **images** are stored. See Chapter 3.2 for further explaination.

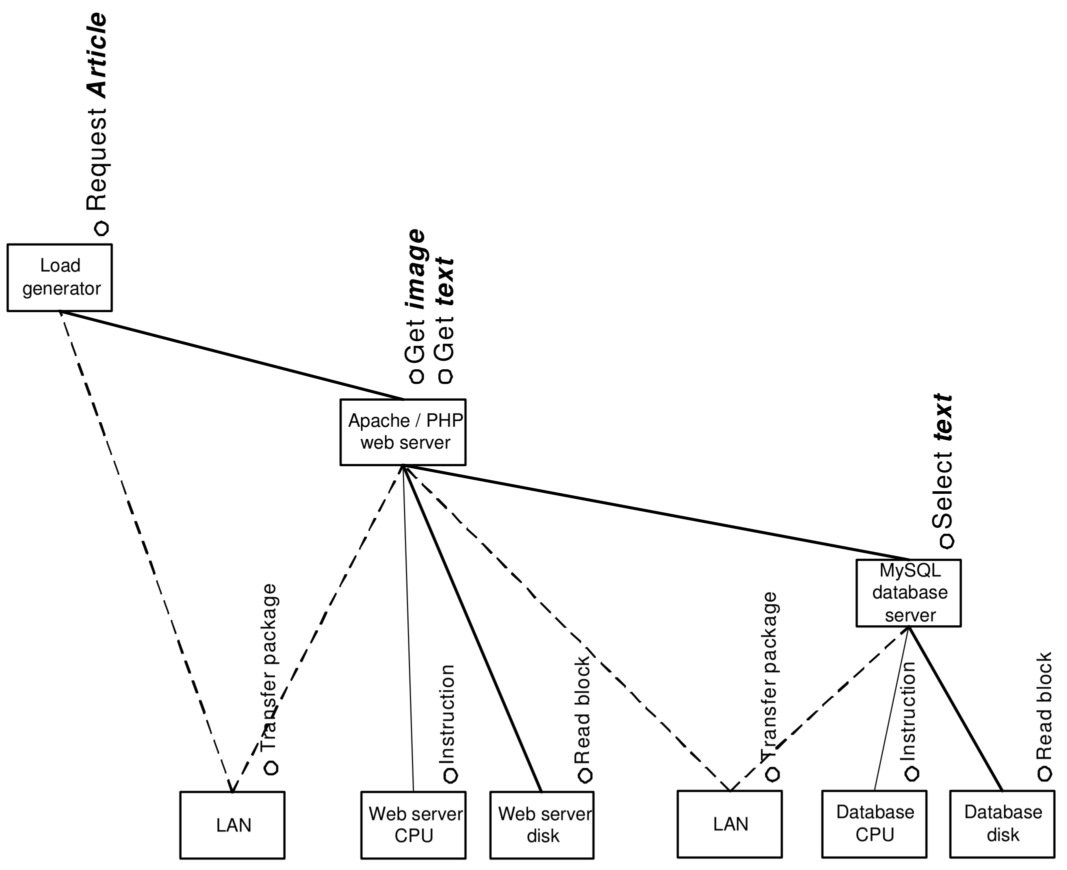


Figure 4.1: SP model of the system

We assume that no images are deleted, so we have dropped such an operation. The Apache web server is running on hardware, consisting of a disk and a CPU, where the memory is included in the CPU component. The web server communicates with the Load generator and the database server through LAN.

4.1.3 Database server

The operation on the MySQL database server is *Select text*, which invokes the *Read block* operation on the disk and the *Instruction* operation on the CPU. We have defined our system to only be a readable web site, thus we only need the *Select text* operation on the database server. *Insert text* and *Delete text* operations would have been necessary if we included a journalist part that published, changed and deleted articles on the web site.

4.1.4 CPU, disk and LAN

The CPU is a device on the lowest level in the SP-model, another device on the lowest level is memory. These two components work very close together, so it is very hard to separate one from the other in our measurements, thus we have decided to include the memory in the CPU unit and have defined its only operation to be *Instruction.* A CPU executes several types of instructions, but we will not distinguish between different kinds of instructions. It is hard enough to perform measurements on the CPU, and we will not make it harder by distinguish between different operations. The CPU is measured by how many CPU cycles it can process in a second, and not by how long it takes to process a single instruction.

The disks are also devices on the lowest level, and they offer the operations *Write block* and *Read block.* The *Read block* and *Write block* operations on disks work with blocks of data. As a natural consequence of our focus on the read intensive part on the web site, we have dropped *Write block* operations on the disks.

The Local Area Network (LAN) is the physical connection between the main components, and it offers a *Transfer package* operation. The load generator, the web server and the database server sends and receives packages through the LAN. The LAN component is an important component, but can be hard to parameterize.

4.2 Complexity functions

The SP-model of the complete system, shown in Figure 4.1 contains ten relations between the components. According to the SP method presented in Chapter 2, this will result in ten complexity matrices. However, for the purpose of our scalability analysis is it useful to do some simplifications to the model.

The LAN components and the connections to and from the LAN components will not be considered as separate components in the further scalability analysis. This is because they are not believed to be important in our scalability analysis, as they are not thought of as bottlenecks device. In addition is it hard to measure on the LAN components.

The reduced model is presented in Figure 4.2 on page 20. It should be noted that this figure is not a valid SP model, but it illustrates which relations in the SP-model shown in Figure 4.1 we will construct complexity matrices for.

We will in the next subsections present the construction of the complexity matrices for the relations shown in Figure 4.2.

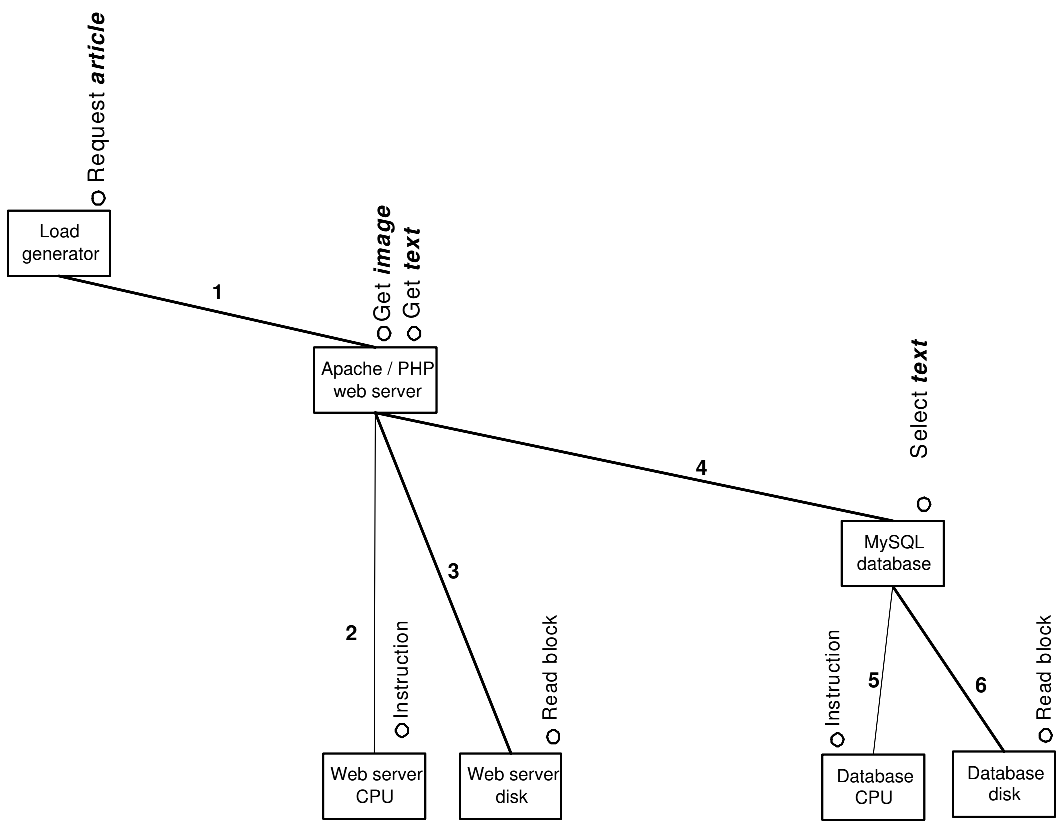


Figure 4.2: Simplified model to show which relations that are investigated further

4.2.1 Parameters used in the complexity functions

The lines in Figure 4.2 describe the dependencies between the components. For each link between the components there exists a complexity matrix, which describes the work invoked on the inferior component by the superior component in the model. One operation on the superior component requires either a constant number of operations, or a complexity function describing the number of operations invoked on the inferior component.

The complexity functions introduces a set of parameters which are explained in Table 4.1. These parameters will help us in grouping the functions and aid us in making a test design for each of the classes. The next subsections introduces all the complexity functions for each matrix, and then Chapter 4.3 summaries the functions and finds patterns of the complexity functions. These complexity function patterns will then after the measurements be parameterized in Chapter 9.

Parameter name	Description of parameter
C_{Get_Text}	Average number of CPU cycles to process one *Get Text* operation
C_{Get_Image}	Average number of CPU cycles to process one *Get Image* operation
C_{Select_Text}	Average number of CPU cycles to process one *Select Image* operation
$C_{Readblock}$	Average number of CPU instructions to process one *Read block* operation
I	Average number of Images in the articles
DV_{Text}	Average data volume of a Text file
DV_{Image}	Average data volume of an Image file
DB	The data block size on the disks

Table 4.1: Parameters used in the complexity functions

4.2.2 Load generator - Web server

C1	Get Text	Get Image
Request Article	**1**	$f(I)$

Explanation of complexity functions
The *Request article* operation on the load generator will always invoke 1 *Get Text* operation, see Chapter 3.2. The number of *Get Image* operation invoked is determined by the number of images in an article (I). This number is found through measurements. The complexity matrix contains one complexity function (one pattern), and the complexity function is shown in Equation 4.1. The parameterization is explained in Chapter 9.1.

$$f = \Big(I\Big) \tag{4.1}$$

4.2.3 Web server - CPU_web server

C2	Instruction
Get Text	$f\Big(C_{Get\_Text}\Big)$
Get Image	$f\Big(C_{Get\_Image}, DV_{Image}\Big)$

Explanation of complexity functions
The *Get* operations invoke the *Instruction* operation on the CPU, and the number of *Instructions* that is invoked for each of the *Get Text* and *Get Image* operations have to be found through measurement. The *Get Text* operation forwards a request for an article from the load generator to the database, and this operation requires a number of CPU cycles to process. Equation 4.2 describes that number. As this function only forwards a standard request, it is not dependent on the data volume, thus it is assumed to be fixed.

The parameterization of this function is shown in Chapter 9.2.

$$f = C_{Get\_Text} \tag{4.2}$$

The number of *Instructions* invoked by the *Get Image* operation is determined by the number of instructions required to process an average image request (C$_{Get\_Image}$), the number of instructions required to read an image from the disk (per image size) (C$_{Read}$) and the data volume of the image (DV$_{Image}$). This function describes the number of CPU cycles needed to process an operation for reading an image, and is shown in Equation 4.3. The parameterization of this function is shown in Chapter 9.3.

$$f = \frac{C_{Get\_Image}}{DV_{Image}} \tag{4.3}$$

This complexity matrix contains of two different functions, which makes two patterns.

4.2.4 Web server - Disk_web server

C3	Read block
Get Text	0
Get Image	$f\left(DV_{Image}, DB\right)$

Explanation of complexity functions
The *Get Text* operation does not invoke any operations on the disk, since there are no articles stored on this disk, see Chapter 3.2. The images are stored on the disk and the number of *Read block* operations invoked by *Get Image* is determined by the data volume of the image (DV$_{Image}$) and the data block size (DB). The function describes the number of read block operations on the web server, and is shown in Equation 4.4. The parameterization of this function is shown in Chapter 9.4.

$$f = \frac{DV_{Image}}{DB} \tag{4.4}$$

4.2.5 Web server - Database server

C4	Select Text
Get Text	**1**
Get Image	0

Explanation of complexity functions
The operation *Get Text* on the web server invokes 1 *Select Text* operation. No other operations are invoked in this matrix, and since there are no complexity functions there are not any patterns.

4.2.6 Database server - CPU_Database

C5	Instructions
Select Text	$f\left(C_{Select\_text}, DV_{Text}\right)$

Explanation of complexity functions
The *Select Text* operation invokes a number of *Instructions* which is determined by the number of instructions required to process an average article request (C$_{Select\_text}$), and the data volume of the image (DV$_{Text}$). The function describes the number of CPU cycles needed to process an operation for reading an article, and is shown in Equation 4.5. The parameterization of this function is shown in Chapter 9.3.

$$f = \frac{C_{Get\_Text}}{DV_{Text}} \tag{4.5}$$

This complexity matrix contains of one function, which makes 1 pattern.

4.2.7 Database server - Disk_Database

C6	Read block
Select Text	$f\left(DV_{Text}, DB\right)$

Explanation of complexity functions
A *Select Text* operation invokes a number of *Read block* operations that are determined by the data volume of the requested article (DV$_{Article}$), and the block size (DB). The function describes the number of read block operations needed on the database disk, and is shown in Equation 4.6. The parameterization of this function is shown in Chapter 9.4.

$$f = \frac{DV_{Text}}{DB} \tag{4.6}$$

This complexity matrix contains of one function, which makes 1 pattern.

4.3 Classification of the complexity functions

Table 4.2 on next page gives an overview over the matrices elements in the matrices presented in the previous subsections. Each row gives a brief summary of each matrix. The second column states the number of matrices element, while the third column shows how many elements which are non-zero, and the fourth column shows how many of the non-zero element which are ones. The last column *patterns*, show how many distinct function patterns there are in each matrix. A pattern is a group of complexity functions which are dependant on the same type of parameters. Zero and ones in the matrices are not regarded as patterns. Table 4.2 shows that there are 6 patterns totally.

Matrix	# elements	$\neq 0$	1	Patterns
C1	1 x 2 = 2	2	1	1
C2	2 x 1 = 2	2	0	2
C3	2 x 1 = 2	1	0	1
C4	2 x 1 = 1	1	1	0
C5	1 x 1 = 1	1	0	1
C6	1 x 1 = 1	1	0	1
SUM	9	8	2	6

Table 4.2: Size and properties of the matrices used

The complexity functions found can be grouped for designing different measurement test, based on what variables to be measured. We have in Table 4.2 identified 6 different patterns of complexity functions. A further study of the functions shows that some of the patterns in different matrices are pretty equal. Some of these patterns can have a common solution to a function that occurs in many matrices, and these patterns are placed in the same group. In Table 4.3 identical function patterns are given the same denotation (group) in the first column.

Matrix C1, in Chapter 4.2.2:	
Pattern	Function
f_1	$f\Big(I\Big)$
Matrix C2, in Chapter 4.2.3:	
f_2	$f\Big(C_{Get\_Text}\Big)$
f_3	$f\Big(C_{Get\_Image}, DV_{Image}\Big)$
Matrix C5, in Chapter 4.2.6:	
f_4	$f\Big(DV_{Image}, DB\Big)$
f_3	$f\Big(C_{Select\_Text}, DV_{Text}\Big)$
Matrix C6, in Chapter 4.2.7:	
f_4	$f\Big(DV_{Text}, DB\Big)$

Table 4.3: Function patterns

We have four pattern groups in total, and the next subsections will explain the function patterns according to P. H. Hughes classification presented in [10]. A pattern is placed in a test design determined according to its dependence or independence of three properties:

- *Load* - which can affect overheads such as context switching, garbage collection, connections etc.

- *Data name accessed* - which can affect cache hits.
- *Data size accessed* - which can affect message lengths, buffer occupancy, memory utilization, etc.

Pattern f_1

This pattern describes how many *Request Article* operations from the load generator that are for images per article request. This pattern is load independent, data name independent and data size independent and is the core test in Hughes' test design.

> ”It has simple and compound variations, according to the context required by the particular operation being measured.” [10]

Pattern f_2

This pattern describes the number of CPU cycles required to process a *Get Text* operation. The pattern only forwards a request of a constant size, and thus it is data size and data name independent. The pattern is assumed to be load dependent. To parameterize this pattern we have to run the test described under pattern f_1 with m multiple threads. m should be varied from 1 to some way beyond the planned operating point. All the load dependent cases have to be run with m multiple threads.

Pattern f_3

This pattern describes the number of CPU cycles needed to process the *Get Image/Select Text* operation. Since this pattern is determined by the file size, reading it is dependent on data size and data name access. It is also assumed to be dependent on the load.

To parameterize this function we have to control the cache effects, which we will do and that is described in Chapter 6.6. An other important factor is *.. to eliminate the effect by operating with specific data sizes, or size distributions.*[10] This means that the results are only valid the data size tested.

Pattern f_4

This pattern describes the number of *Read block* operations on the disk. The pattern is data size dependent, data name dependent and load dependent, which leads to the same test design as described in function pattern f_3.

Chapter 5

Scalability

This chapter will present the concepts of scalability, and outline the likely scaling scenarios in our thesis. Scalability is a well known term, and there exists several different definitions of the word. In its simplest form it can be defined as an IT systems ability to handle growth. Peter Hughes, Gunnar Brataas et al. have come up with the following definition [12]

> *"An architecture is scalable with respect to an IT profile and a range of desired capacities if it has a viable set of instantiations over that range"*

By IT profile we mean all other requirements than capacity, e.g. quality of service and functional requirements, and with viable we mean that it is feasible both economical and technical.

Scaling dimensions

Hughes, Brataas [8][12] has identified three important scaling dimensions a typical e-business application would contain.

- *Processing capacity* describes the rate at which specific work can be performed. E.g. disk service rates.
- *Storage capacity* describes the amount of capacity that can be stored at some level. E.g the capacity of a database.
- *Connectivity* the total number of access points to a system or subsystem. E.g. can be the maximum number of simultaneous users to a web cache or the maximum connections of an Ethernet cable.

"The physical resources used are different for each dimension. For ease of exposition we will refer to them collectively as *size*."[12] This enables us to define a scaling function for each dimension, which delivers a certain capacity for a certain size.

Scaling functions

A scaling function plots capacity and size in a particular scaling dimension. Some things scale linear and some things scale non-linear. Hughes, Brataas et.al.[12] identifies three different scaling functions. One linear, one super linear (indicating economies of scale) and one sub linear (indicating diseconomies of scale).

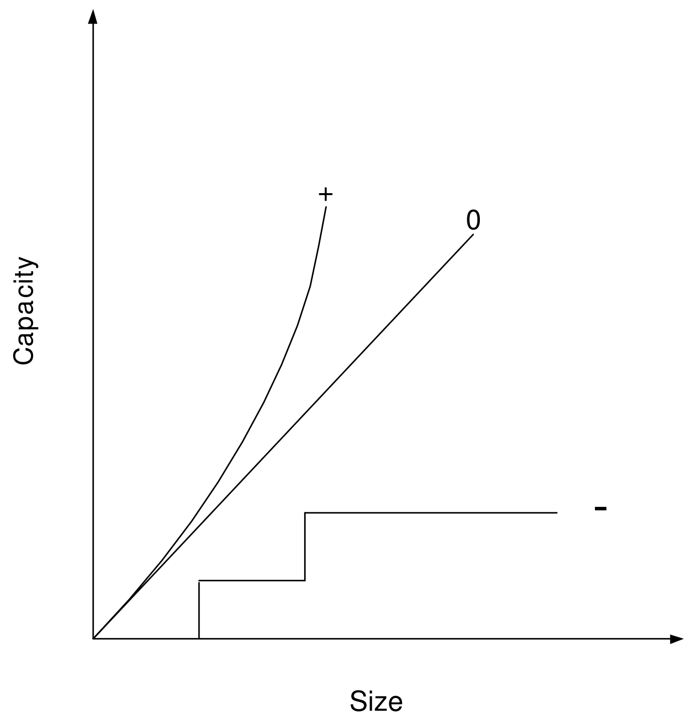


Figure 5.1: Different types of scaling functions: + super scalable, 0 linear scalability, and – non-scalability

Scalability and requirements

As we have seen, a system is scalable in respect to an IT profile that is a set of requirements other than capacity. Typical requirements for a web system can be divided into functional and non-functional requirements (reliability, security, cost, quality of service).

5.1 Scaling scenarios

To evaluate the scalability of a system, several scenarios are needed to test the system against. The system is scalable for a given scenario if it can handle the growth we imagine in the range of that scenario and still fulfill the requirements. A scaling scenario is a set of functions, one for each of the scaling dimensions, which indicates the future growth of the system.

The way a system is scaled can be considered both with respect to a particular scaling dimensions or all scaling dimensions together. When the size changes by the same factor in all dimensions, it is called *strict scaling*[3] . The term *uniform scaling* is used with respect to only one dimension, when all the sizes of the subsystems are changed by the same factor.

Scaling is closely related to growth of a system, and in a large system it is unlikely that *uniform scaling* is possible, because of different architectural limitations. One might imagine that one wants to change the storage capacity of the different server types with different ratios. According to Hughes, Brataas et.al. in [12] *non-uniform* or *skewed* scaling may complicate the formulation of scaling functions and make the analysis harder.

To continue the process started in [21], we choose to set up a system which resembles the TV2i system. In other words, our system models a Norwegian news site and with this starting point we will outline some likely scaling scenarios.

To help us in predicting the likely growth scenarios, we have contacted leading Norwegian news sites, and asked them for their opinion regarding the future of news sites. Based on their input, we will try to outline the most likely scaling scenarios. The entire survey can be found in Appendix A.

User definition

It is important to establish the terms *simultaneous users* and *concurrent users.* In this thesis concurrent users describes virtual users that are visiting a site without requesting any file. Think of all the people reading a internet news site, but not actually clicking to request a resource. Simultaneous users or simultaneous requests describe the users that execute requests within the system within a small time frame.

Increased data volume of articles

In the past Internet articles has usually been fairly short, with a small amount of low-resolution pictures, and this has kept the total data volume of an article fairly small. The main historic reason for this, was the low amount of bandwidth available for the Internet user, but this has changed with the introduction of broadband. According to our survey, almost every participant believes that the articles will experience a massive increase in size.

Some expect an increase in the text length, but this will not contribute so much to an increase in total size. It is the use of multimedia content (image, video, sound) which will account for the massive increase. One of our participants expects that an increase in the resolution of the images is likely because the average screen size has gone up. TV2i has seen that articles with more multimedia content are more popular, than articles with little or no multimedia content. Aftenposten and Dagbladet have started to use video in each article from the World championship in Nordic Sports, and this indicates that multimedia content will be more common.

Increased use of video is highly interesting, but because video on the Internet is normally streamed, thus running on dedicated hardware, we do not look into that. The reason video on internet is streamed rather than downloaded, is because the user does not have to download the entire clip, before viewing it. For one possible approach on assessing the scalability of a video streaming architecture see [11].

These trends make it possible to think that the total size of each article (text size + multimedia content) will increase in the years to come, even a tenfold increase is not impossible. The reason for this is because multimedia content is much larger than text.

Today most of the articles on a web site system are in the region from 50 KB to 100 KB, this is the total article size (text, images, overhead). Images are currently between 10-30KB, but we choose to start at 100KB, because we believe based on our survey that this will be a normal size in near future. We will look at images in the interval from 100KB to a 1000KB.

Text size is currently between 5KB-20KB, and we choose to look at the interval from 10KB to 300KB. 300KB might sound very large, but we choose to include this, as we might imagine that larger documents can be put our for viewing, e.g. goverment reports etc.

The impact of an increased data volume is believed to be significant, each request will take more resources to process, the archive will increase much faster and the caches will need more memory.

Increased number of users

There are a constant growth in the Internet usage, and according to TNS Gallup[17] approximately 47% of the Norwegian people over thirteen have daily access to Internet. This is a doubling in the coverage in just 4 years, and this growth is expected to continue. This means that more and more people have access to the Internet, and thus we can expect a general increase in Internet usage.

The largest web site in Norway has increased its number of daily reader by 280% since 2001, from 307.000 to 820.000[17]. Similar growth has been experienced by other large and medium sized sites as well. This growth is expected to continue, although it is important to understand that there is an upper limit on the number of Internet users in Norway.

Number of daily readers is just one way of measuring users. We are more interested in number of simultaneous requests. This is because simultaneous requests say something about the load on the website, while daily readers do not give this information.

Most of the web sites, predicts that the user behavior will change, and that each user will access the web sites more frequently. Pål Nisja in TV2i predicts that each user will spend three to five times more time on the Internet. This means that the number will simultaneous requests will increase, but there will not be an increase in the number of unique users. Each user will put more strain on the system.

In addition to an organic growth, we have to think about non-organic growth (mergers or acquisitions). We might see a consolidation in the Internet news segment, where we might experience that larger web site buys or merges with smaller web sites, thus increasing their customer base over night. As such growth happens over a very short time span, assessing the scalability is ever more important.

There is in other words very likely, that the number of simultaneous request will increase in the years to come, through an increase in the number of daily readers and/or an increased in the time spendt by each daily reader.

Increased number of articles

An increase in the number of articles published per day can have a large impact on a system. The storage will reach it limits faster. According to our survey, there is however no indication that the number of articles published per day will increase a lot. To begin with, the number of articles published a day varies from 10 to 250, between the different sites. So the number of articles per day seems to be more site specific, so a general trend about an increase in this number can not be drawn.

Increased use of "pay per click"

To earn more money, web sites may want to demand payment for reading articles and accessing the archives. According to our survey, most of them agree that this will not common, but it might happen for the archive. This aspect is interesting when looking at scalability, because the user would have to log into the site using a secure connections. The use of secure sockets leads to increased work on the CPU, because it takes more CPU power to establish and maintain a secure connection than a normal one [14].

Increased number of web robots

A web crawler is a program that visits web sites and reads their pages and other information in order to create entries for a search engine index. The reason that web crawlers can decrease hit rate is that they do not necessarily ask for the most popular objects. Menascè, Almeida and Riedi conclude in [1] that "the presence of robots causes a significant increase in the miss ratio of a server side cache. Crawlers have a referencing pattern that completely disrupts locality assumptions". A decrease in web cache hit will lead to increased load on the back-end servers.

5.1.1 The focus in this thesis

Based on our survey we choose to focus on one scaling scenario, namely *increased article size*. This scenario has to subscenarios: *increased text size* (from 10KB to 300KB) and *increased image size* (100KB to 1000KB).

We will measure on how these subscenarios affect the performance of the system. The reason we focus on these scenario, is because it is the most likely one, and because it is unkown to what degree they will affect the system. When we know how they affect the system, we will be able to say something about how the system handles *increased number of users.*

Chapter 6

Measurements

This chapter will explain how the measurements on the system components are performed, discuss relevant issues connected to this, and give an outline of a measurement plan. The purpose of our measurement is two folded; first of all it is to investigate the implications of increased image size on the web server and increased text size on the database component. Secondly it is to investigate how the increased image size, acts on different hardware (a scale-up experiment).

6.1 Scale-up

Our experiment will use two types of nodes, and we will compare the component performance (web server and database) between them. This is a so called *scale-up* experiment, where the hardware is upgraded. We will not look into replication, which is the other way a system can be scaled. When dealing with a scale-up experiment it is important to establish the scale factors between the different hardware.

We will first measure on the web server component installed on the old node, and then perform the same measurement on a new Clustis2 node with the same operating point (see section 6.2).

CPU

In our experiment we will measure the CPU utilization, and from that together with the clock speed derive the CPU usage of a component. It is crucial to find the scale factor between our two CPU's, that is how much faster the new one is than the old one. Our two nodes have different CPU architecture, one is an Intel CPU and the other an AMD processor. This complicates the process of finding the scale factor, because the CPU speeds given by the manufactures are not directly comparable.

The old node is running with an AMD Athlon 1600+ processor which is clocked at 1.4 GHz[1], and the new node is running with an Intel Pentium 4 processor clocked at 3.4 GHz. If we just divide the clock speed on each other; 3.4 GHz divided by 1.4 GHz we obtain a scale-up factor of 2.4.

[1]Found on, http://tinyurl.com/b4qut

However, AMD processors are performing more instructions per clock cycle than an Intel processor[2]. AMD has a rating system, which converts their processor into an Intel equivalent. According to this rating system our AMD processor corresponds to a 1.6 GHz Intel Pentium 3 processor. This gives a scale-up factor of 2.1. However, since our Intel processor is a Pentium 4, which is believed to be faster than the Pentium 3 at the same clock cycle time, the estimate of 2.1 seems a bit to low.

Based on this we chose to opt for a scale-up factor of 2.4[3].

IO

CPU is one important factor, another one is disk performance. Both disks run at the same disk speed, namely 10.000 RPM. The new node have a disk seek time of 4.7 ms, while the old node has a seek time of 5.2 ms. The disk transfer rate is the third important parameter for disk performance, the new disk has a formatted internal transfer rate of maximum 78MB/s, while the old one only has 63.2MB/s. So the new disk has a scale-up factor on 1.23 on transfer rate, and 1.1 on seek time. We will there use an overall scale-up factor of 1.1 on the entire disk, as the smallest scale-up factor will be the deciding one. In addition, we believe that the transfer is high enough and never fully utilized even on the old disk.

Storage space could be an interesting scale factor to look at, but we will not look into it as we are not directly looking into any scaling scenarios concerning disk growth.

Network

Both nodes have a 1Gbit network card. Initially the old node only had 100Mbit card, but as we wanted to be sure that the network was not the bottleneck, we upgraded it to a 1Gbit card. Because the network bandwidth is more than sufficient, further measurement on network will be neglected. The network traffic will generate some load on the CPU, but this will be hard to isolate from the rest of the web component.

Scale-up factors

The scale factor between our standalone Clustis node and the Clustis2 nodes, varies as seen between the different components. We have in other words non-uniform scaling. The nodes are more or less the same, the main difference is the CPU performance (2.4), and a moderate better disk with a scale-up factor of 1.1.

6.2 Operating point

Under a scale-up the both systems must run on the same operating point. Operating point is a way of ensuring that it is possible to compare to different system under a scale-up. In [12] Brataas, Hughes et.al proposes that the operating point can be determined according

[2]Found in article available from http://pclt.cis.yale.edu/pclt/PCHW/clockidea.htm

[3]This consists with preliminary findings in Geir Bostad's (IDI, NTNU) master thesis

Component	*Clustis*	*Clustis2*	*Scale factor*
CPU	AMD 1600+	Intel 4 3,4GHz	2.4
Mem	1GB	1GB	1
Disk			1.1
Network	1Gbit	1Gbit	1

Table 6.1: Scale factors between Clustis and Clustis2

to two criterias, either *equivalent utilization* or *equivalent response time*. In our experiment we choose to use equivalent utilization, mainly because it was more feasible to get reliable measurements on the CPU than on the response time.

The operation point should be fixed on the bottleneck device, in our case the CPU is the bottleneck device. This means that the CPU utilization on the old node, and the new node should be equal during the experiments. This implies that we expect to put a higher load on the new web server, as that probably will be able to cope with more requests on the same utilization. We have chosen to have the same operating point on the database server and the web server.

Choosing the operating point is mainly a pragmatic question, as the operating should reflect the load on the system in its natural operating environment. Normally someone with good domain knowledge on a given system would say what operating point the system would run on, and that would have been chosen. It our case, we are free to choose the operating point.

We wanted the operating point to be fairly high, but it must not be so high that the CPU is saturated. To support us in choosing the operating point, we performed a series of measurements on the old node with different image size. These measures gave us result from 0% CPU utilization to 100% (see figure 11.3 on page 84). As the graphs were fairly linear, we could have chosen almost any point. We ended up with using an operating point of 60% on the nodes. Ideally we would have performed the measurements on more than one operating point, but this was not possible because of time constraints.

6.3 How to isolate resource demand

When trying to quantify the parameters in the complexity function, we are actually trying to measure the resource demand. That is the work the operation puts on a given resource, e.g. we are trying to find the number of CPU cycles an operation uses. On real live system this can be very hard as many programs use the same resource simultaneously. One of the main missions in our experiment is to be able to isolate the resource usage by the operation we want to parameterize. To be able to truly isolate the resource demand for a single operation is almost impossible, but it should be possible to get a good approximation.

The way we isolate the resource demand, is to run a given number of identical operations on the computer simultaneously, so the CPU reach the operation point. We can then calculate the CPU usage for a single operation. During our experiment CPU load should ideally only be coming from the web component or the database, however we might imagine that some other components can consume CPU time during our measurement. However, since the CPU idle is 100% before we put any load of the system, and quickly returns to

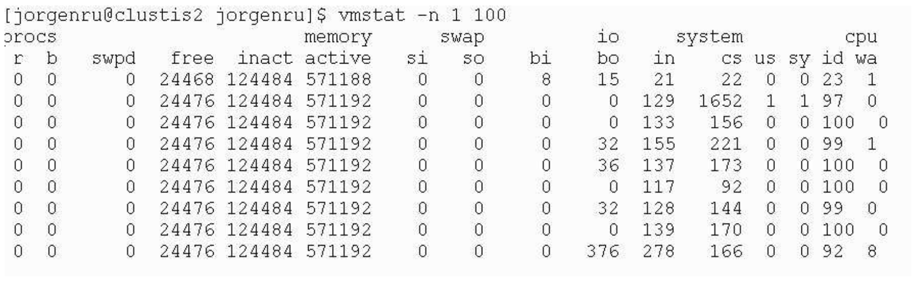
100% after the experiment is completed, we believe that we have managed to isolate the web and db components in our experiment.

6.4 Measuring Tools

For our experiment we have relied on standard measuring tools available on the Unix/Linux platform. The advantage with these is that they are unobtrusive, that is they do not put a high load on the system, thus not interfering with the measurement results. They take a snapshot of the system state every second, and this is recorded in a file.

6.4.1 CPU tools

To capture the CPU utilization we use the standard Linux tool, *vmstat*. It manages to capture the CPU idle utilization every second. We can then find the CPU utilization by subtracting this result from 100. In addition to the CPU idle time, vmstat reports a lot of other performance statistics see Figure 6.1. We are however not looking into any of them.

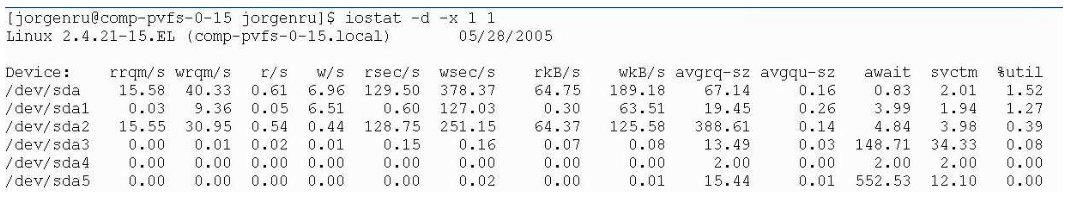


```
[jorgenru@clustis2 jorgenru]$ vmstat -n 1 100
procs                          memory      swap          io     system         cpu
 r  b   swpd   free  inact active   si   so    bi    bo   in    cs us sy id wa
 0  0      0  24468 124484 571188    0    0     8    15   21    22  0  0 23  1
 0  0      0  24476 124484 571192    0    0     0     0  129  1652  1  1 97  0
 0  0      0  24476 124484 571192    0    0     0     0  133   156  0  0 100  0
 0  0      0  24476 124484 571192    0    0     0    32  155   221  0  0 99  1
 0  0      0  24476 124484 571192    0    0     0    36  137   173  0  0 100  0
 0  0      0  24476 124484 571192    0    0     0     0  117    92  0  0 100  0
 0  0      0  24476 124484 571192    0    0     0    32  128   144  0  0 99  0
 0  0      0  24476 124484 571192    0    0     0     0  139   170  0  0 100  0
 0  0      0  24476 124484 571192    0    0     0   376  278   166  0  0 92  8
```

Figure 6.1: Example output of the vmstat command

6.4.2 IO tools

To capture IO usage, namely disk writes per second and disk read per second, we used the standard Linux tool *iostat*. It captures these parameters with a granularity of one second. When performing this command on the nodes, we get a lot of unnecessary information. It captures statistics for each device, (see Figure 6.2), but we only need to look at /dev/sda2 which is the device where our temporary data is located. We are only looking at the rKB/S (KB read per second), wKB/s (KB written per second), and the %util, for the /dev/sda2.



```
[jorgenru@comp-pvfs-0-15 jorgenru]$ iostat -d -x 1 1
Linux 2.4.21-15.EL (comp-pvfs-0-15.local)       05/28/2005

Device:    rrqm/s wrqm/s  r/s  w/s  rsec/s  wsec/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await  svctm  %util
/dev/sda    15.58  40.33 0.61 6.96  129.50  378.37    64.75   189.18    67.14     0.16    0.83   2.01   1.52
/dev/sda1    0.03   9.36 0.05 6.51    0.60  127.03     0.30    63.51    19.45     0.26    3.99   1.94   1.27
/dev/sda2   15.55  30.95 0.54 0.44  128.75  251.15    64.37   125.58   388.61     0.14    4.84   3.98   0.39
/dev/sda3    0.00   0.01 0.02 0.01    0.15    0.16     0.07     0.08    13.49     0.03  148.71  34.33   0.08
/dev/sda4    0.00   0.00 0.00 0.00    0.00    0.00     0.00     0.00     2.00     0.00    2.00   2.00   0.00
/dev/sda5    0.00   0.00 0.00 0.00    0.00    0.02     0.00     0.01    15.44     0.01  552.53  12.10   0.00
```

Figure 6.2: Example output of the iostat command

6.5 Workload

According to P. H. Hughes [8] the following information is needed to characterise the *workload*:

- *work*
  The set of operations or commands invoked upon the system from the environment
- *load*
  - In a closed system, this is the number of independent processes which are concurrently generating work.
  - In an open, the arrival rate of work at system
- *sequence*
  This is needed for very detailed studies, involving behavior which is dependent on the system state, and thus dependent of the sequence of the operations.

When we are measuring on a system, we must determine the workload put on the system. On existing system, this can be quite difficult. E.g. If we are trying to determine the workload on an Internet system, the characteristics of Internet traffic must be understood. Internet traffic is known to exhibit some characteristics; inter-arrival time distribution, burstiness, heavy-tail distribution [16][15][7][19], and these must be understood in order to make a workload.

On our system it is quite easy to determine the workload, because our system is used for experimental purposes. As for the *work*, we are requesting the same operation over and over again[4]. This is either *Get image* or *Get text* (see Chapter 4, depending on whether we are testing on the web server or the database server.

Our system is a closed model, because the number of users in the system is constant (see discussion in Chapter 2.1.5). The number of independent processes varies between the different image and article sizes, and the number of processes used for each can be seen in section 6.8.5 on page 41. The think time Z, that is the time the process spend after completing a request before sending a new one is zero.

In our model we do not need to worry about the *sequence*, as we only run identical operations.

6.6 Controlling the caches

A big issue in our experiment has been how we were going to control the cache effects. Caches exist on several different levels in a modern computer, both on hardware and software. Caches speed up the process, and we have to control these caches in order to find out how much they account for. Because there are so many caches on so many different layers it is very hard to fully understand the full effects of the caches.

There are several methods the cache can be controlled.

[4]This was the only way, we could isolate the resource demand

1. The caches could be turned off. In other words the request always experience a cache miss.
2. The caches can be filled with all the data which are used. (Always cache hit)
3. A combination of method 1 and 2.

Controlling these caches is inherently difficult. If it had been feasible, the ideal solution would have been to perform all the three solutions outlined in order to determine the effects of cache. The method with full cache hit and cache miss, would have enabled us to get an upper and lower band on the performance. While the third method, is the one which resembles real news sites.

In our system, we have managed to turn the caches on program level off, that is any caching within Apache and MySQL. We are however unable to turn off the disk cache and the primary memory. So in order to 100% cache miss ratio, we would have to requests images/articles sequential in order of 2 times the primary memory (2GB). This is possible on large image sizes, where we would have to have 2000 images for 1000kB which were read sequential. On small image/text sizes, e.g. 10KB we would need 200.000 images/texts, and this is not feasible.

It would have been possible for set the system up so that every requests would experience a cache hit, but this implies that there would have been no reading on disk. Since parameterization of disk was one of our goals, we would have lost valuable information.

This leaves us with only one feasible solution, a mix between cache hits and cache miss. The most important thing for us was that the experiments between the different systems (the new and the old) and between the image sizes were comparable. Our solution was to use a fixed number of images/articles, and look at it from a real-life web server perspective. This is the most likely scaling scenario from the news sites standpoint.

We chose to use 2000 images/text files on each size, 2000 was chosen because this is about one month worth of images for a news server[5]. More images/articles than this is not likely to be stored on the same disk, it is much more likely to be put in an archive solution. Just to be sure, we checked what happened it we used 4000 images/articles instead, but it gave more or less the same results. A further doubling was impossible because of storage space on the disk.

A drawback with this method is that the cache effects will be larger on smaller image/text sizes, as more of them can be found in the primary memory. However, since the Apache connections have to be in the memory as well, this decrease the available cache space available on small image/text sizes as they have more connections which needs memory. The cache hit ratio will not be constant, but vary between the different image/text sizes. However, this solution should be the most realistic and the best way to simulate a real web site.

The timeline in this project has been extremely tight and, and more investigation into the cache effects would have been desirable. The chosen method is however a very realistic method and gives a good impression of the effect of increased article size on real systems. The server can be seen as being in an operating state, where some articles may be found in

[5]We wanted to have the same number of images and articles, because of this we use 2000 articles as well

the cache but other may be fetched. But because we are going to run each test 10 times, and for a 100 seconds this should even out.

6.7 Replications and run length

There are many similarities between simulations and measurements, as both experience statistical variations and noise which is desirable to remove. There are many ways to perform a measurement / simulation, concerning run length and the number of replications. One solution is to have one long run, another is to choose many individual replications and run them for a shorter time.

We opted for a solution with a fairly short run length and where each experiment was executed 10 individual times, as this gives us confidence that the results were reproducible. In addition, if the experiment for some reason was invalid, less time has been lost than if we had one long test that was invalid. If an experiment of 3 hours is invalid, 3 hours is lost, while if a shorter test of 10 minutes is, invalid only 10 minutes is lost.

Each individual experiment had a run time of 100 seconds. This may seem fairly short, but it was apparent that the system reached a steady state fast, so a longer runtime would not give any more precision. In addition time was of the essence, as we were performing a large amount of measurements.

In most measurements only the steady-state performance is of interest that is the performance after the system has reached a stable state. The initial state before this one is called the transient state, and results from this part should not be included in the final results. The problem of identifying this initial state is called removing *end effects*. Raj Jain[13] has proposed six methods for identifying and solving this problem. We have gone for the *initial data deletion*, where we delete the end effects before and after the steady state. This was accomplished by looking through the logs. Figure 6.7 show an excerpt from such an experiment log, and from this the end effect phase can easily be identified (the first 7-8 seconds). Our solution is to remove the 20 first and the 20 last second of an experiment. This ensures that we only use the results from the steady state, this method is perhaps a bit crude and a more sophistical method could have been used, but we wanted to make sure that we only used data from the steady state.

6.8 Measurement plan

Based on the considerations we have outlined in this chapter we will device a measurement plan for our experiment. In the next sections we describes a measurement plan[6]. This is not an extremely detailed plan, as most of the measurement is done automatically.

[6]Loosely based on the IEEE std 829 standard for test planning. According to the IEEE 829 test planning process there are eighteen factors to consider, and for that reason we have to do some appropriate modifications to the standard to fit the purpose of our measurement plan.

```
FS800-S42-RT100
{...}
VMSTAT LOG
99
100
100
100
100
100
100
62
49
38
41
35
43
37
{...}
```

Figure 6.3: Excerpt from an experiment log

6.8.1 Introduction

This measurement plan specifies all system testing activities to be done on the old Clustis system, and the new Clustis2 system.

6.8.2 Test system

The system is configured on Clustis2, which consist of 17 new nodes and one old node. Both type of nodes in the cluster are running Red Hat Linux, and for a throughout description of the system, see Section 3.3. For measurements on the web component two nodes are needed, and for the measurement on the database component five nodes are needed.

6.8.3 What to be measured

We will measure transactions per second, CPU and disk statistics for each measurement. All measurements are performed with operating point on CPU equal 60% and with the test harness explained in Chapter 7.

- *web measurements on old node* to simulate increased image size.
  Image sizes in KB: 10, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000.
- *web measurements on new node* to simulate increased image size
  Image sizes in KB: 10, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000.
- *database measurements on new node* to simulate increased article size
  Text sizes in KB: 10, 50, 100, 200, 300.

The image/text sizes used will be the range which our scaling scenarios are viable over. This range was determined in Chapter 5.

The test scenario

For an experiment to be valid, the median utilization during the test run must be between 55% and 65%. We allow this because it is almost impossible to get an operation point of exactly 60%.

We have chosen to use the median to find the operating point, and not the mean average CPU utilization. Our measurements logs show both the values, and in most cases both the average and the median give approximately the same result. All the "good" measurements with a stable CPU utilization give almost the same value, and we could have chosen either of the values. However, some times there is a difference, that is in those cases were some of the samples in the measurements can be considered as outliers. For some reason the CPU drops to 0% utilization some times, and outliers like that affect the average much more than the median. If the difference between median and average is too large the measurement is not accepted.

6.8.4 What not to be measured

We will not measure on the network, as that is believed to be more than sufficient when it is all 1Gbit/s.

6.8.5 Schedule

In this section the parameters used for each image/text size is listed. In our experiments, there is only one parameter in addition to image/text size and that is number of sessions. Sessions is a parameter used by the load generator to determine how many request per second it should put on the target machine.

Numbers of sessions for each image/text size where found through trial and error. First we established the number of sessions needed to get a valid measurement for the largest image/text size and the smallest image/text size. We had then found the upper and lower bound, and then we found it on the middle image/text size. From these three measurements it was pretty easy to determine the other image/text sizes, as the number of sessions increased pretty fixed with increased image/text sizes. To find the number of sessions needed, we had to try about three for each size.

It proved much harder to measure on small image sizes than on large image sizes. For the web measurements on the new node we use the number of sessions from Table 6.2 on next page, for web measurements on the old node those from Table 6.3, and for database measurements on new node the sessions from Table 6.4.

It is important to note that in Table 6.4, the number of sessions is the total number of sessions. Since measurements on the database require two load generators, each of the two load generators should be initiated with half of the total. E.g. if total sum is 1800 sessions, each load generator should have 900 sessions.

Image size (KB)	10	50	100	200	300	400	500	600	700	800	900	1000
Sessions	3000	1700	1038	537	372	290	160	88	60	42	35	30

Table 6.2: Number of sessions needed on new node to produce valid web mesurement for a given image size

Image size (KB)	10	50	100	200	300	400	500	600	700	800	900	1000
Sessions	312	225	221	135	118	80	63	57	52	43	37	34

Table 6.3: Number of sessions needed on old node to produce valid web mesurement for a given image size

Text size (KB)	10	50	100	200	300
Sessions	$\approx$ 4000	$\approx$ 3700	$\approx$ 3000	$\approx$ 2400	$\approx$ 1900

Table 6.4: Number of sessions needed on new node to produce valid database mesurements on a given text size

Chapter 7

Resource Function Workbench

This chapter will present a framework for automatization of the measurements needed in our thesis. This includes both the collection and the post-processing of the measurements. Capturing resource functions on computer systems is a highly time consuming and complicated activity, thus creating the need for a workbench which aids the process. This chapter will outline our framework for such a workbench, inspired by Vetland, Woodside, Bayarov and Curtouis [25], and Vetland and Hughes [22]. The workbenches outlined in these articles are a highly general one, but the one we have designed is more domains specific. The reason for this is because it would be too costly to make a domain independent workbench.

This process of designing and implementing this framework has in itself been one of the most time consuming tasks in the entire thesis. Because of this we will explain our solution in detail in this chapter, hoping that future projects within this area can reuse our framework.

All the scripts introduced in this chapter, are presented in further detail in Appendix C, some of the source code can be found in Appendix D and the entire source code is available on the thesis website [20].

7.1 Background theory

Vetland et. al. came up with the following six requirements on resource functions, the process and tool for measuring them in [22][25].

1. A test configuration is required, which should be reproducible and representative.
2. It should be easy to create a test for a new component, since software technology is constantly changing.
3. It must be possible to run the measurement experiment automatically, over different processing environments, so that the information can be maintained as models and versions change.
4. It must be possible to apply the parameter changes in the experiments and run them many times over ranges of parameter values.

5. A compact representation is needed for the function which captures the variation of the demand, as the parameters change.
6. It should be possible to combine resource functions from different sources. E.g. theoretical reasoning and expert judgement.

These requirements leads to an application that can be illustrated by Figure 7.1. The first four requirements are covered by the left hand side of the figure. Requirement 3 to 6 illustrates the need to store the results in a proper manner, and this is solved by the components in the right hand side of the figure.

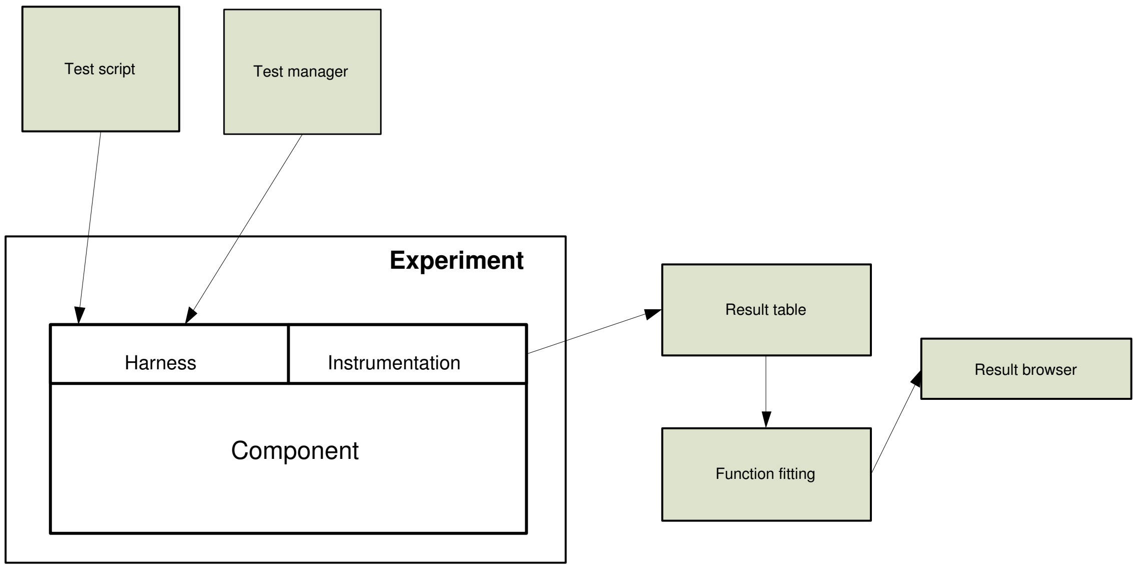


Figure 7.1: Elements of a Resource Function Workbench [25]

7.2 Description of workbench

Based on the requirements outlined and Figure 7.1, we designed our workbench. Our main goals for constructing such a workbench were:

- Automate the measurement process in order to save time and make it feasible.
- Make it easy to measure on different baselines (hardware).
- Save the results in a manner which is human readable, parsable and reusable.
- Save time for future projects in this area, by making it possible for others to reuse our workbench.

7.2.1 Outline of solution

Figure 7.2 on 45 shows the three main scripts in our workbench. *Startexperiment* and *stopexperiment* sets up our test harness (system), while *measure* is the test manager and

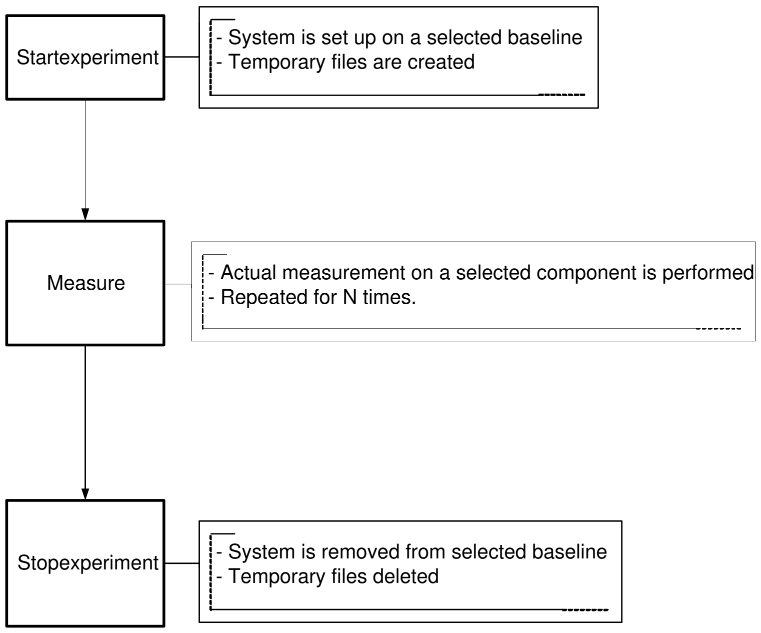

Startexperiment
- System is set up on a selected baseline
- Temporary files are created
Measure
- Actual measurement on a selected component is performed
- Repeated for N times.
Stopexperiment
- System is removed from selected baseline
- Temporary files deleted

Figure 7.2: Main scripts in the workbench

test script. The instrumentation used in this workbench is two standard measuring tools for the Unix/Linux platform, namely vmstat and iostat. More information about our instrumentation can be found in Chapter 6.4. The resources we measured on was the CPU and IO usage for a given component, either database or web server.

An experiment is a series of individual measurements on a given component for a given set of parameters. The parameters are set in the *measure* script, together with the experiment name. The experiment name should be descriptive, and indicate whether measurement was done on web server, database or old or new nodes.

The output for each individual measurement is saved in a folder, named after the experiment name and the date (*experimentname-date*). This is done to increase the tracability of the measurement, and to keep all the measurements from a single experiment in the same place.

The output for each measurement, has a filename which indicates the parameters used. E.g.: FS900-S35-RT100, indicates that 900 is the file size[1], 35 is the number of session and 100 is the runtime of the measurement. An excerpt from such a file, can be found in Figure 7.3 on page 46, and this format is both human readable and easy to parse with a script.

These individual result files are parsed with *interpretmeasurements.sh* to create a Excel friendly format for the entire experiment. In Excel we use spreadsheet to automatically calculate the standard deviation and confidence interval and plot graphs. Much of this calculation could have been done with this script, but we did not find it necessary.

[1]Either text or image size, and this should be clear from the experiment name.

```
FS900-S35-RT100

################   GENERAL DATA #####################

 Filesize:900
 Total number of connections:2012
 Average number of connection per sec:33.53
 Failed request:0

################   VMSTAT DATA #####################

 Utilization CPU, Median:41
 Utilization CPU, Max number:71
 Utilization CPU, MIN number:13
 Utilization CPU, Arithmetic mean:40.48



################   IOSTAT DATA   #################
 Disk utilization [Arimetric mean]:11.049
 Disk utilization [Median]:19.05
 Disk utilization [MAX]:20.51
 Disk utilization [MIN]:4.29
 --------------------

 Disk KB read [Arimetric mean]:41344.403
 Disk KB read [Median]:30687.5
 Disk KB read [Max value]:149228.57
 Disk KB read [Min value]:14394.12
 --------------------

 Iostat KB write [Arimetric mean]:148.96
 Iostat KB write [Median]:0
 Iostat KB write [MAX value]:1220.00
 Iostat KB write [MIN value]:0.00



----------- LOG FILES ------------------
```

Figure 7.3: An example result file for an individual measurement. In this case with image size 900, 35 session on the load generator, and with a runtime of 100 seconds.

7.2.2 Detailed description

This section will include a walkthrough of an experiment, how it is done, and what must be done.

The first thing which must be done is to reserve the number of nodes needed. This is because Clustis is a shared cluster. This reservation ensures exclusivity on the nodes used.

If we are measuring on the web server component, two nodes are needed, one for the load generator and one for the web server. When we are measuring on the database component, five nodes are needed; two load generators servers, two web servers and one database server. This is because the database server handles much more, and smaller connections which creates the need for more hardware to generate enough load.

The nodes are reserved using the *reserve-nodes.sh* script[2]. This script check for available nodes, reserves the number needed, and binds the IP addresses of the nodes requested to global variables. This way, the scripts only need to use variables instead of IP addresses. E.g. *$WEBSERVER* instead of the 10.255.255.128.

After the nodes are reserved, the *startexperiment* script is executed. The components are then automatically installed on the nodes, and the system is up and running. This script is only runned when changes to the baseline is needed, e.g. when changing from measuring on old nodes to new nodes. The scripts initiated by the *startexperiment* can be seen in Figure 7.4 on page 48. First it copies local files over to local node disk (*mysql-data.sh* and *Images-data.sh*), then it transform the skeleton files into updated config files (*transform-skel-files*). E.g. *transform-skel-files.sh* transform the IP address for the web server in the load generator configuration file.

After the experiment baseline is set up, we use the *measure* script to set the parameters for the experiment, and this script is manually edited before each experiment according to the measurement plan. The parameters that needs to be set is what file sizes that should be measured on, what sessions that should be used, and the experiment name.

After the parameters are determined, the *measure* script is executed on the load generator node. The file runs sequential through a series of individual measurements, where one of these measurements with a runtime of 100 seconds[3] takes about 5 minutes to complete. This part of the script is indicated by the loop in Figure 7.5 on page 49. We usually measure about 150 combinations of file size and sessions at a time. When this is completed, the measurements are outputted into a Excel friendly format vy the *interpretmeasurements* script.

The scripts initiated by the measure script can be seen in Figure 7.5. *Generateload* and *formatmeasurement* is the actual measurement process, and this is run in a loop for each individual measurement. A further breakdown of the *generateload* script can be seen in Figure 7.6 on page 50. The most important thing to notice here is the three script which run in parallel, namely *start hammerhead*, *capturecpuusage* and *captureiousage*. These run in parallel because the actual measurement has to be done, while the component is under load from the load generator.

At that time we usually run *stopexperiment*, to clean up the server logs. The scripts initated by the stopexperiment can be seen in Figure 7.7 on page 51.

[2]Thanks to Geir Bostad for providing us with an almost complete script

[3]which is the run time decided in Chapter 6

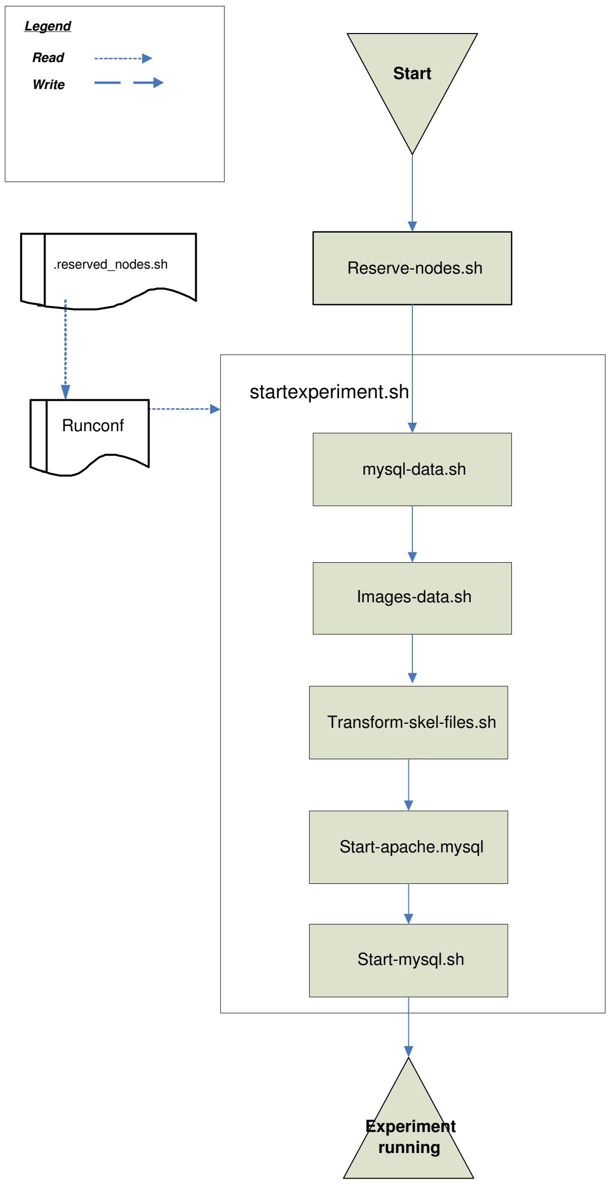


Figure 7.4: How the experiment is started

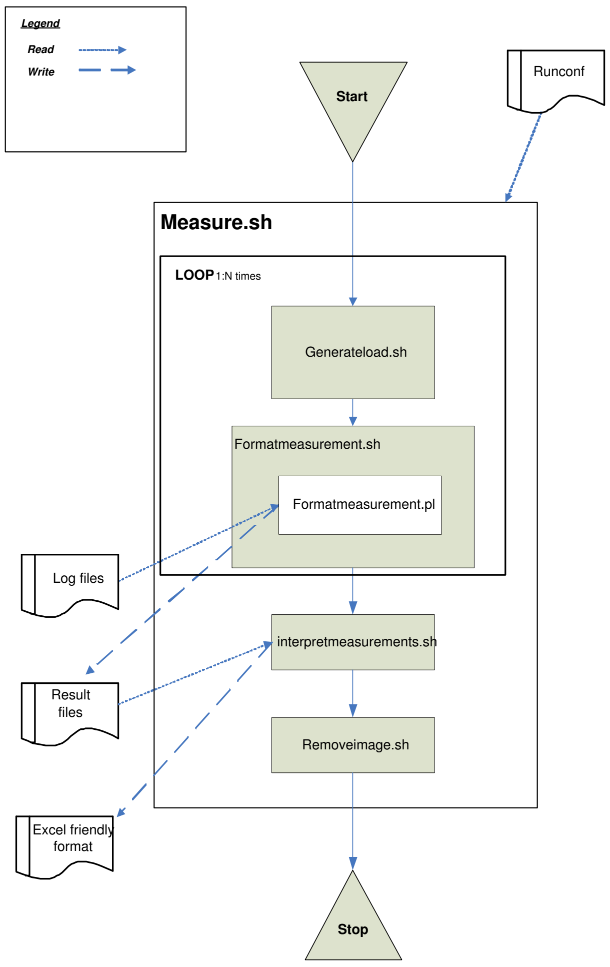


Figure 7.5: Breakdown of the measure script

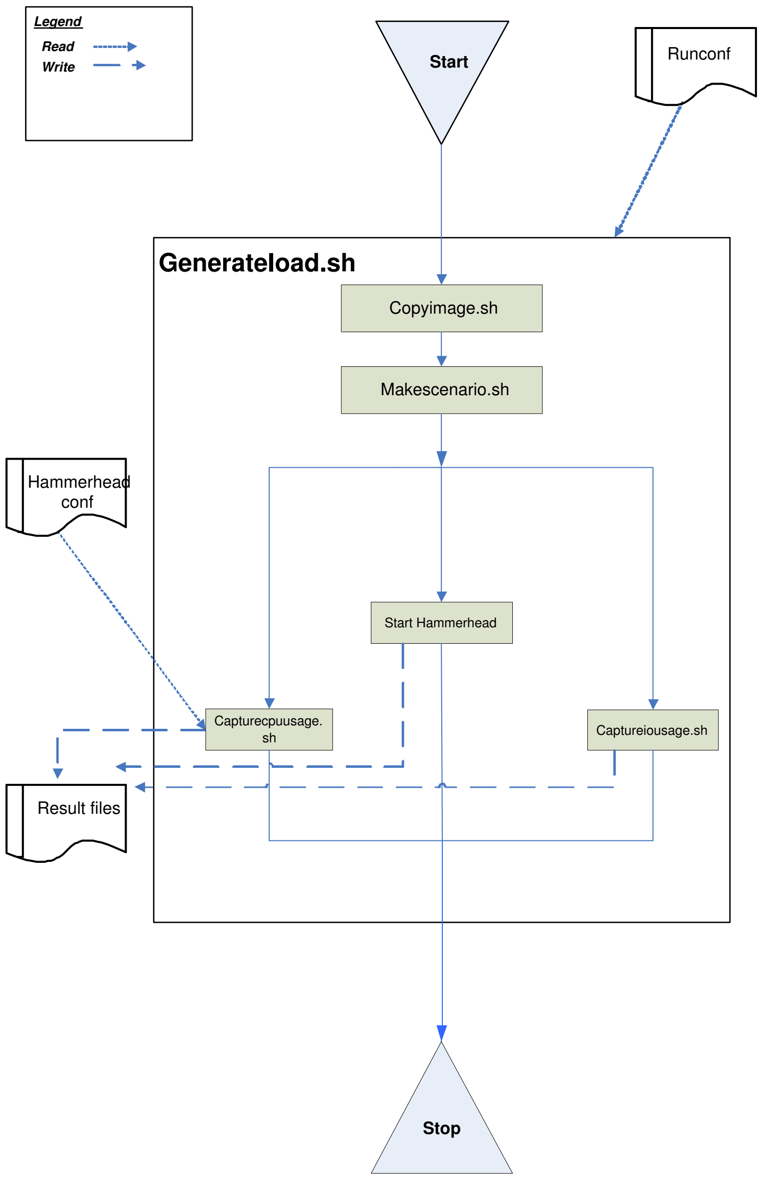


Figure 7.6: Generateload breakdown

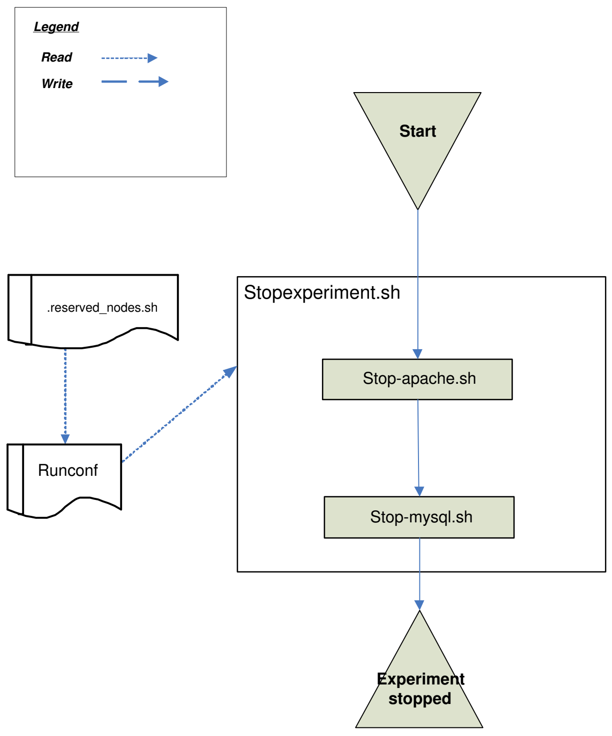


Figure 7.7: How the experiment is stopped

The workbench is set up so that it can be easily moved from one installation to another, all scripts read a single configuration file called *runconf*. This file set all the global variables, any changes which affect the program should be set here. This should make it fairly easy to reuse our workbench, however since the Clustis cluster is a fairly odd architecture, there is probably some scripting which needs to be done in order to reuse all of our scripts. The one script which is most likely to be easily reused is the *measure* scripts, because this is more or less hardware (system) independent. This was the script which took the longest to make, so it is good that it is the most reusable.

It should also be noted, that the entropy of the workbench increased during the phase of the thesis. The reason for this was the old node used, was a stand alone Clustis node. It did not have a shared home area, as the other nodes. So some alternations to the scripts were needed to make it work. It should however be noted, that it is mainly hacks on the start and stop experiment which was performed, and no alternations to the measure script.

Chapter 8

Results

In this chapter we will present an overview of the results from our experiments. A detailed presentation of results for each individual image size or text size can be found in Appendix E. In addition to this, the spreadsheets with all the measurement data used to calculate the confidence interval and means can be found on our web site [20].

As mentioned in our measurement plan (see Chapter 6), the measurements are runned with a run time of 100 seconds. For a measurement to be valid the CPU utilization has to be between 55% and 65%, and on each image or text size we want 10 individual valid measurements. This was accomplished for most of the measurements, but because it was hard to get a stable load on the small image sizes, some of them have fewer valid measurements.

To find the throughput on exactly 60% CPU utilization, we used linear regression. We also calculated a 95% confidence interval for this result, which shows how reliable and reproducible the measurements are. The confidence interval for each image or article size are shown within the graphs as vertical error bars.

8.1 Experiment results on old web server

Figure 8.1 on page 54 shows the overall results for the experiments on the old web server. The graph shows that as the image image size increases the throughput on the web server decreases, and this behavior was expected.

There was an unexpected interval in the graph, and that is the shape of the curve between image sizes 10 KB and 100 KB. The result for image size 50 KB is lower than for image size 100 KB. We had expected a smooth curve and throughput (Tps) around 250 for image size 50 KB, but this odd result can be explained with our problems of getting valid measurements on image sizes below 100 KB. This is reflected in the large confidence interval for these image sizes.

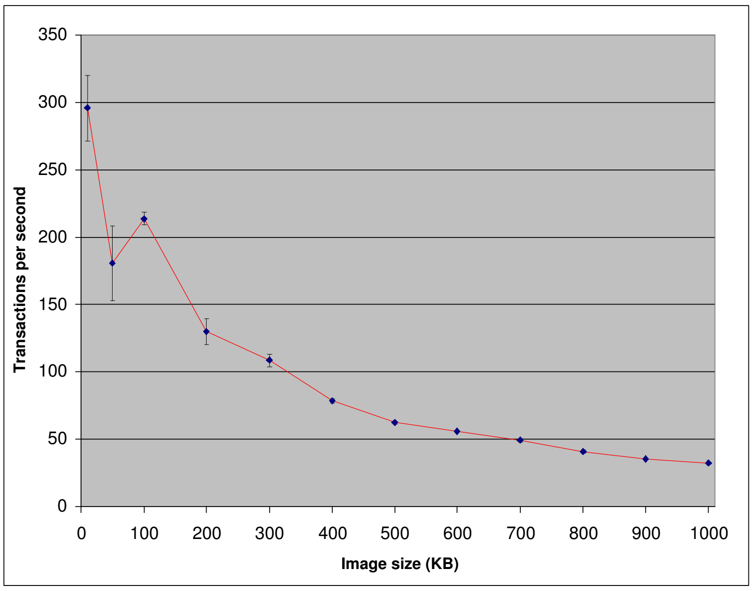


Figure 8.1: Throughput (Tps) versus image size on old web server

8.2 Experiment results on new web server

Figure 8.2 shows the overall results on the new web server. The graph has a smooth curve, and is pretty much as we expected it to be. However, the curve decreases faster than expected for the small image sizes. The graph shows a large confidence interval for image size 50 KB, and on image size 10 KB there is a infinite confidence interval that not is shown in the graph. The infinite confidence interval is a result of few measurements on this image size (we were only able to get two valid test runs).

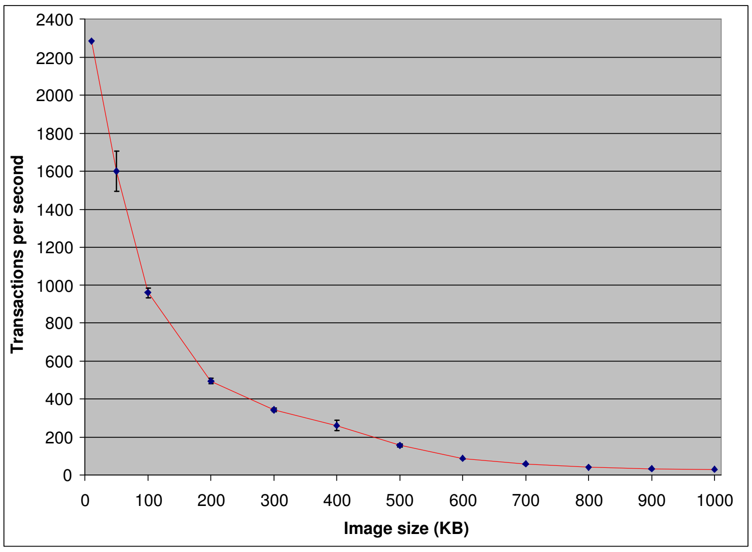


Figure 8.2: Throughput (Tps) versus image size on new web server

To see the effects of the scale-up on the CPU, we have compared the two web components against each other in section 8.5.

8.3 Experiment results on the database

Figure 8.3 shows the overall results from the experiments on the database server, which were performed on the new nodes. The database was able to handle much higher number of sessions than the web servers, so we used two load generators and two web servers to perform the measurements on the database. We did not measure on larger article text size than 300 KB since an article text not is likely to be any bigger, see Chapter 5.

The graph in Figure 8.3 shows that the measurements have a large confidence interval. It was problematic to generate a stable load with the load generators for such a high number of simultaneous connections as we needed on the database. Even though the large confidence interval, the curve is as expected and the curves shape is decreasing and relative linear.

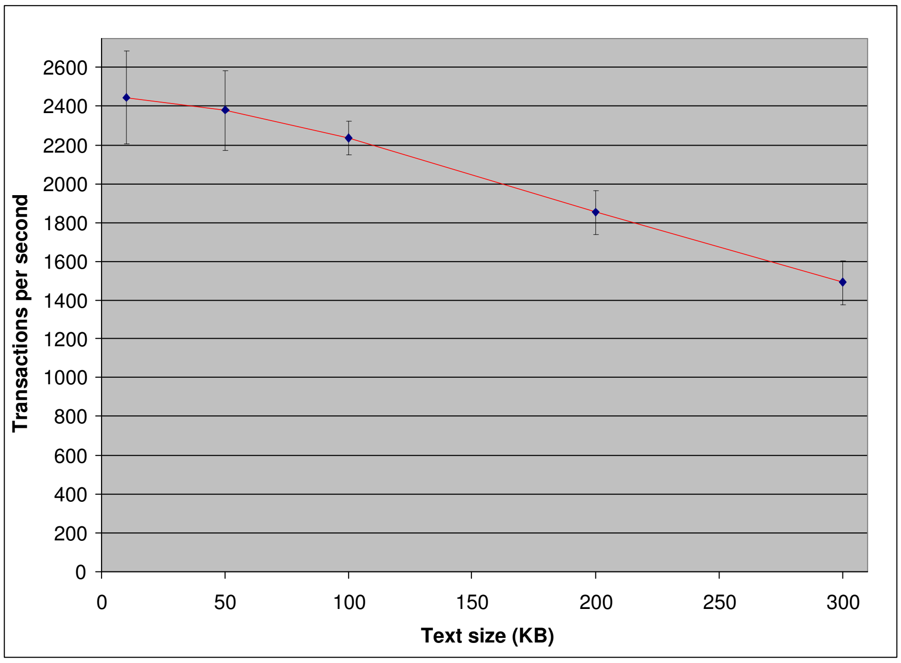


Figure 8.3: Throughput (Tps) versus text size on the database

8.4 Disk measurement

We have experienced some problems with the disk measurements, and main reason is that the measuring tools have not been functioning as expected.

Old node

On the old node the measuring tools[1] did not report any reading what so ever. This is extremely strange, and even when we tried to copy images with image size of 1000 MB, the tool reported no disk activity.

New node

On the new nodes, we managed to get results for disk, for image sizes over 400KB. The reason we only get reading above 400KB, is most likely caching. On the small image sizes all the images are in memory.

We have 2000 images on each image size, and for image size 400 KB gives this a total image volume of 800 000 KB (≈0.8 GB). The memory has 1 GB storage space, so all the images for this image size fits in the memory. For image size 500 KB the total image volume is larger than the memory and the CPU have to read from disk. This fits well with the fact, than the reported reading increases as the image size increases, as the cache miss ratio is increasing.

The results obtained on the new web server is shown in Figure 8.4. The values in the graph are the average reading during a test period for each image size. It should be noted that, there seems to be disk activity only a few times during a measurements. So we consider these measurements to be quite unreliable. However, to get any results we have decided to use the average read during the test period.

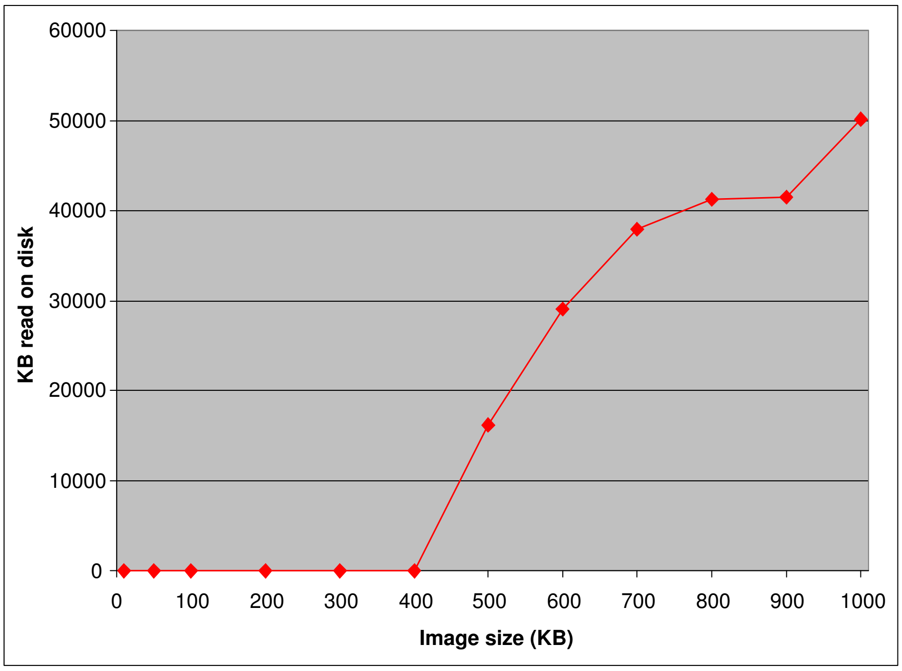


Figure 8.4: KB read on disk read vs. image size on new web server

[1]We tried several of the tool available on Linux, we even compiled them up ourself

Comment

It is very strange that we do not get any disk reads on the old server. Everything is the same between the measurements on the old and the new node. The number of images and the primary memory are the same, so there should be no difference in caching. The test harness and the measuring process is the same, so there are no difference in how the results are collected.

8.5 Comparison of the web servers

The graph in Figure 8.5 compares the throughput between the old and the new web server. As we see the new web server performs much better on the small image sizes, where the CPU is the bottleneck. On large image sizes the throughput is more or less the same. We believe this is because on larger image sizes, the disk becomes the bottleneck. And as both disk has more or less the same performance, the number of transaction are more or less the same.

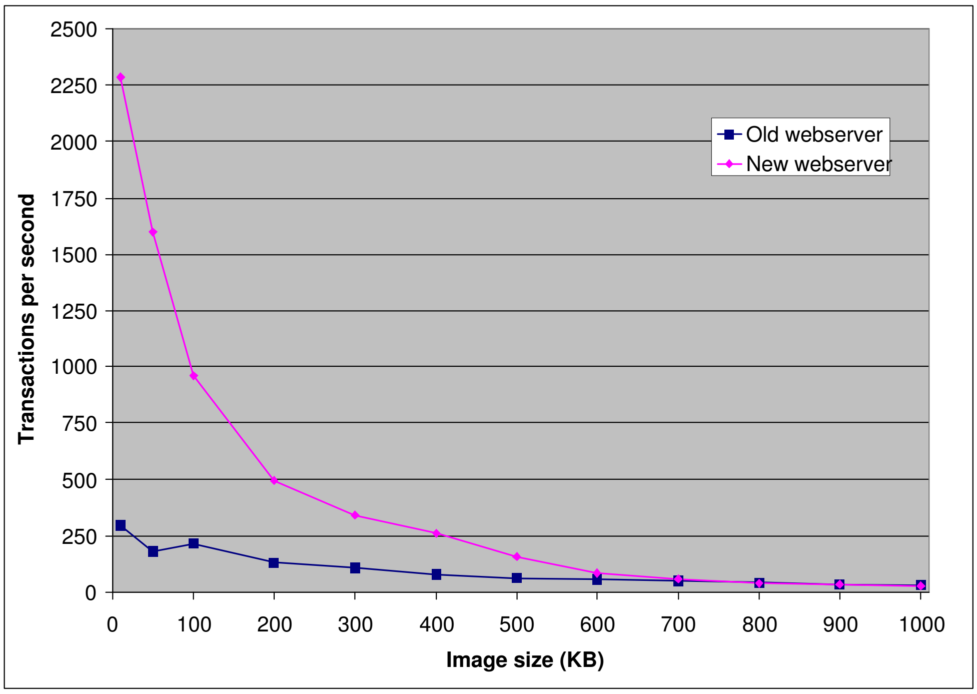


Figure 8.5: Comparison of Tps vs. image size between old and new web server

Chapter 9

SP-parameterization

In this chapter we will parameterize the SP-model identified in Chapter 4, based on the results of the measurements presented in Chapter 8. Parameterization is very hard to get exact due to several reasons; it is hard to manage to truly isolate the resource demand and the available measuring tools are relatively coarse. As a result of this, we have to base the parameterization on some assumptions and approximations, which will be explained in the following subsections. All the parameters used in the complexity functions is shown in Table 4.1 on page 21.

9.1 Function pattern $\mathbf{f}_1$

To parameterize this function pattern we need to find the average number of images per article at the measured web system. This function pattern is only dependant of how many images an article consists of. If the articles consists of 3 images on average, this function states that one request from the load generator will request three image requests. As we have measured on a test system, we have not modeled a web site with an average number of images per article. We have varied the number of image per article, but the parameterization of the function in Equation 9.1 should be easy to complete. I in the equation is the average number of images in the articles, see Table 4.1 on page 21.

$$f_1 = I \tag{9.1}$$

9.2 Function pattern f_2

Function pattern f_2 is the number of CPU cycles used by the web server to process a *Get Text* request to the database server. This request fetches the text. To parameterize this function we need to measure the Utilization on the CPU when running the *Get Text* operation. This function is extremely hard to determine, as it requires almost no CPU. Our measurements showed almost no utilization when performing a large number of this operation per second. From this we assume that this operation use a minimal number of CPU cycles.

We assume that the cost for a *Get Text* operation is similar to the *Get Image* operation.

We can then use the parameterization of the *Get Image* operations in the next section 9.3 to calculate the number of CPU cycles used for a *Get Text* operation. We assume that *Get Text* operation is maximum 1 KB, we can then find the cost of this operation by inserting 1 KB into equation 9.11 (old node) and equation 9.8 (new node) on page 65.

$$f_{2,oldws} = 2.02 * 10^6 \tag{9.2}$$

$$f_{2,newws} = 0.50 * 10^6 \tag{9.3}$$

This is a quite crude approximation, but it is believed to be a worst case approximation.

9.3 Function pattern f_3

To parameterize this function we need to measure the CPU utilization when running the *Get text*/*Get image* operations, and the text size / image size has to be known. This function pattern describes how many CPU cycles the web server use to execute the *Get text* or *Get image* operation.

Since it was impossible to measure how many CPU cycles exactly one operation uses, we had to estimate it. We measured how many identical transactions (operations) the CPU served per second (Tps) for a given CPU utilization. Thus we were able to find the number of CPU cycles used for one operation by using Equation 9.4, which we believe gives a good approximation.

$$\sharp CPU\_Cycles = \frac{Speed_{CPU} * Utilization_{CPU}}{Tps_{image\_size\_or\_text\_size}} \tag{9.4}$$

Let us explain the equation we have introduced:

1. Speed$_{CPU}$ is the speed of the CPU, and this is known through the configuration of the system. This speed is given in Hz and gives us the total number of CPU cycles executed in a second.

2. Utilization$_{CPU}$ is the operating point we have chosen, which is 60%. By multiplying this value with the total number of CPU cycles executed in a second, we get the total number of CPU cycles that are used in a second to execute operations.

3. Tps$_{imagesizeortextsize}$ is the throughput (Tps) for each image size or text size at operating point 60%, which have been found through measurements and regression analysis. These values are presented in Appendix E. We know from item 2 how many CPU cycles that are used in a second, and by dividing this number with the number of operations in a second (Tps) we find how many CPU cycles that is used by one operation. This will distribute any overhead evenly between the operations.

   We assume that no of the CPU cycles used in a second, is used by other operation than the operation we have measured. This assumption can be made, because we are pretty sure we have managed to isolate these operations (see Chapter 6.3).

We have measured on three components, respectively web servers with a CPU speed at 3.4 GHz and 1.4 GHz, and database server with a CPU speed 3.4 GHz. Equation 9.4 is the basis for the parameterization of function pattern f_3 for each of the measured components.

Old web server

We use the formula from Equation 9.4 to calculate the number of CPU cycles used on the old web server. The calculation with image size 1000 KB is shown in equation 9.5, while the results for the other image sizes is shown in Table 9.1. The Speed$_{CPU}$ for the old web server is 1.4 GHz and the Utilization$_{CPU}$ is 60%. This applies for all the image sizes on the old web server, while the throughput is individual for each image size, and can be found in Appendix E.

For image size 1000 KB the Tps is measured to be 32 transactions per second, if we Insert this into Equation 9.4 we find the CPU cycles used for one *Get image* operation on image size 1000 KB, see Equation 9.5.

$$\sharp CPU\_Cycles_{Old,1000KB} \approx \frac{1.4 * 10^9 * 60\%}{32} \approx 25.96 * 10^6 \tag{9.5}$$

Image size	10KB	50KB	100KB	200KB	300KB	400KB
$CPU\_cycles \approx$	$2.84 * 10^6$	$4.61 * 10^6$	$3.92 * 10^6$	$6.47 * 10^6$	$7.75 * 10^6$	$10.71 * 10^6$

Image size	500KB	600KB	700KB	800KB	900KB	1000KB
$CPU\_cycles \approx$	$13.52 * 10^6$	$15.09 * 10^6$	$17.12 * 10^6$	$20.65 * 10^6$	$23.83 * 10^6$	$25.96 * 10^6$

Table 9.1: CPU cycles used on the old node

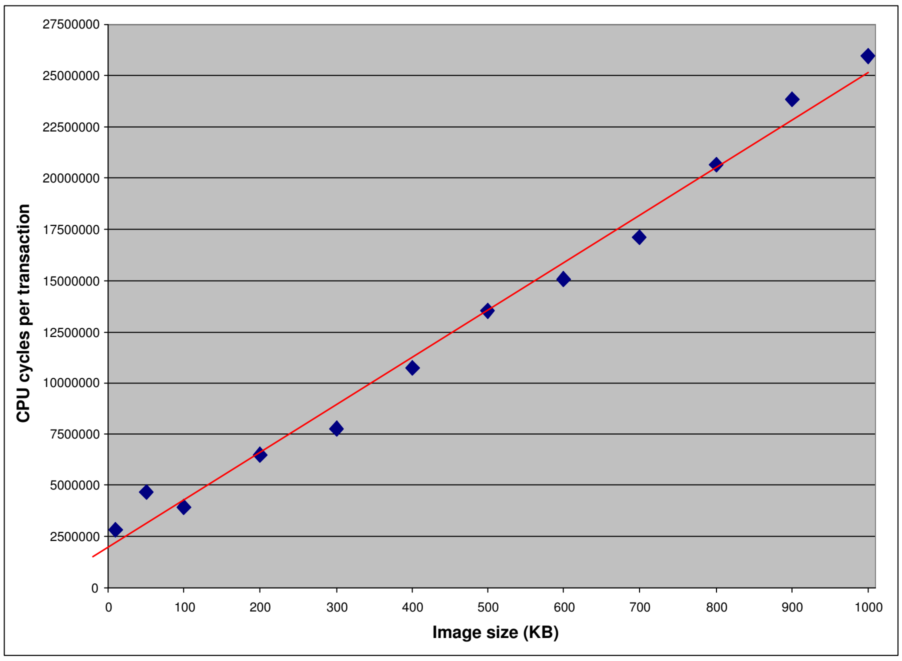


Figure 9.1: CPU cycles used vs. image size on old web server

The graph in Figure 9.1 shows an increase in CPU cycles as the data volume increases. This growth is approximately linear, and by linear regression we have found a function matching the red line in the graph. This line gives the parameterization of function pattern f_3 for the old system, and is presented in Equation 9.11.

$$f_{3,oldws} = 23207 * Image\_size + 2 * 10^6 \tag{9.6}$$

New web server

The calculation of the CPU cycles used on the new web server is done in the same way as the old web server. The Utilization$_{CPU}$ is still 60%, but the Speed$_{CPU}$ for the new web server is 3.4 GHz. These parameters apply for all the image sizes on the new web server, while the throughput is individual for each image size.

For image size 1000KB the throughput, Tps$_{Image\_size}$ is measured to be 28 transactions per second, see Appendix E. Inserted in Equation 9.4 we find the CPU cycles required for one *Get Image* operation on image size 1000 KB in Equation 9.5. The results of the calculations of the other image sizes are shown in Table 9.2.

$$\sharp CPU\_Cycles_{New,1000KB} \approx \frac{3.4 * 10^9 * 60\%}{28} \approx 72.94 * 10^6 \tag{9.7}$$

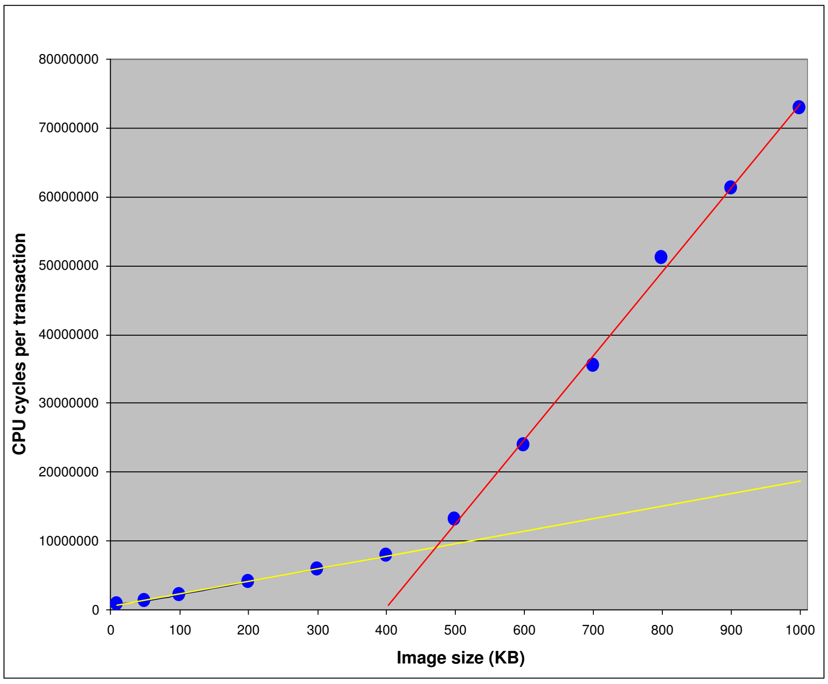


Figure 9.2: CPU cycles used vs. image size on new web server

The graph in figure 9.2 shows the CPU cycles used for one operation on each image size for the new web server. The shape of the graph is very interesting as it have a clear break and the graph can be divided into two linear parts. The clear break is between image size 400 KB and 500 KB. These two linear parts create the function pattern for the new web

Image size	10KB	50KB	100KB	200KB	300KB	400KB	500KB	600KB	700KB	800KB	900KB	1000KB
$CPU\_cycles \approx$	$0.89 * 10^6$	$1.27 * 10^6$	$2.13 * 10^6$	$4.13 * 10^6$	$5.96 * 10^6$	$7.81 * 10^6$	$13.12 * 10^6$	$23.89 * 10^6$	$35.48 * 10^6$	$51.19 * 10^6$	$61.31 * 10^6$	$72.94 * 10^6$

Table 9.2: CPU cycles used the new web server

server. We could have used polynomial regression, but since the two parts are so clearly linear we choose linear regression, given in Equation 9.8.

This parameterization function found is different from what we expected, as we had expected a linear graph like that one for the old web server in the last section. Now we have two linear functions where the slope of the regression on the large image sizes is 6.7 times bigger than the slope for the regression for the small image sizes. There is something odd about this function, and in the next paragraph we will give a more plausible parameterization for the CPU on the new web server.

$$f = \begin{cases} 18227 * Image\_size + 479791 & \text{for } Image\_size \leq 486KB \\ 122025 * Image\_size - 5 * 10^7 & \text{for } Image\_size \geq 487KB \end{cases} \tag{9.8}$$

Alternative parameterization If we study Figure 9.3, we see that the new web server seemingly use more CPU cycles than the old web server on image sizes from 500 KB to 1000 KB. This is unexpected and odd behavior, as there is no reason why the new web server should use any more cpu cycles that the old one. We have indentified two two plausible explainations for this behavior.

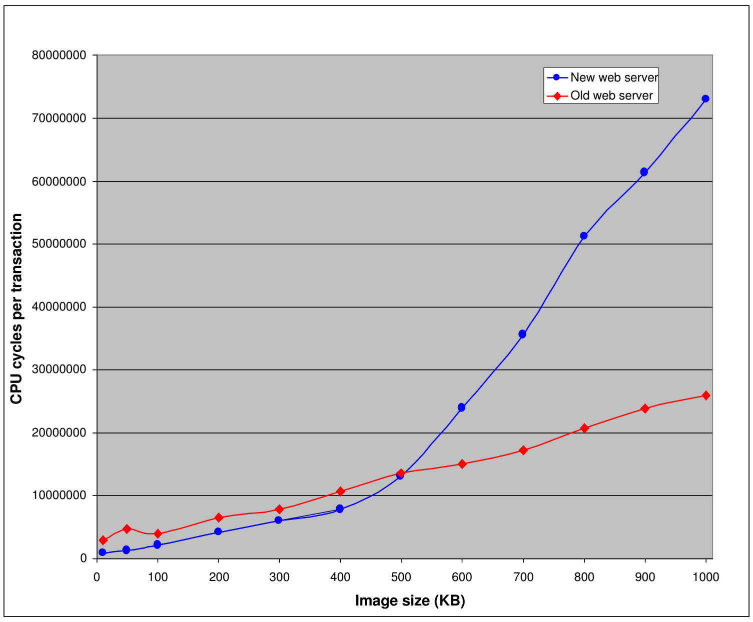


Figure 9.3: Comparison of the CPU cycles used vs. image size on the old and new web server

It could be some kind of experimental error. However, since all the experiments are performed exactly the same way and the software used are identical between the two, this is not very likely.

The other plausible explanation for the break is because our approximation (Equation 9.4 on page 60) of CPU cycles is based on the fact that all the reported CPU utilization is used by the *Get image* operation, and this is the case as long the CPU is the bottleneck.

On the small image sizes up to 400 KB the CPU is the bottleneck, while the disk is the bottleneck for image sizes from 500 KB to 1000 KB. This is supported by measurements performed on the new web server, see Figure 8.4 on page 56. As can be seen from this figure there are reading on disk from image size 500 KB and upwards. From the same figure we can see that there is no reading on image sizes below 400 KB, this indicates a cache hit ratio of a 100% in this interval. When the image size increases above 400 KB the cache hit ratio will decrease and this leads to disk reads.

So when the bottleneck is changing from the CPU to the disk between image size 400 KB and 500 KB, the approximation of CPU cycles starts to get wrong. When the disk is bottleneck not all of the reported CPU utilization is used by the *Get image* operation. Some of the reported utilization is IO wait, since our measurement tool report IO wait as an active CPU state. This means the reported utilization is higher than the number of CPU cycles used by the actual operation. We use Equation 9.4 on page 60 to calculate the CPU cycles used by the operation, in this equation is Tps and the CPU speed right while the CPU utilization is to high and this leads to an overestimate for the CPU cycles used.

This happen on the old web server as well, but not nearly as evident. The reason for this is because the old CPU is slower and that the degree of overestimation is higher on the new CPU.

Because of this, we believe that a better approximation for the CPU cycles used on the new node, is to continue the liner regression function for the interval 10 KB to 400 KB all the way up to a 1000 KB. This can be seen in Figure 9.3. We then see that the relation between the CPU cycles used on the new and the old web server is relative constant. The parameterization for the new CPU is then given as function in Equation 9.9.

$$f_{3,newws} = 18227 * Image\_size + 479791 \tag{9.9}$$

Database

The CPU cycles used on the database server are calculated the same way as the CPU instruction on the web servers. The calculation on the database system for text size 300 KB is shown in equation 9.10. The values in the equation are $1.4 * 10^9$, which is the speed of the CPU, 60% which is the utilization on the CPU, and 1493 which is Transaction per second found in Chapter 8

$$\sharp CPU\_Cycles_{DB,300KB} \approx \frac{(1.4 * 10^9) * 60\%}{1493} \approx 0.562 * 10^6 \tag{9.10}$$

By replacing $TPS_{textsize}$ in equation 9.4 with the Tps for each text size, the CPU cycles

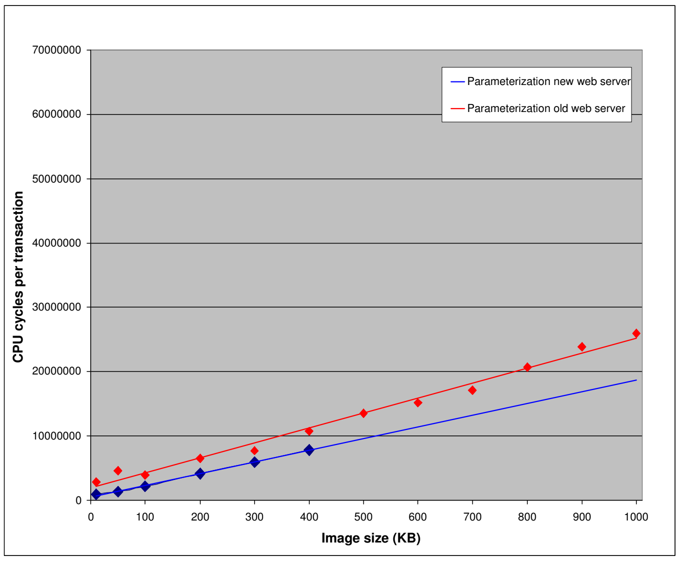


Figure 9.4: Parameterization of the new and old web server

used for one operation are found. The results of the calculations for the other text sizes are shown in Table 9.3.

Text size	10KB	50KB	100KB	200KB	300KB
$CPU\_cycles \approx$	$0.344 * 10^6$	$0.353 * 10^6$	$0.376 * 10^6$	$0.453 * 10^6$	$0.562 * 10^6$

Table 9.3: CPU cycles used on the database

The graph in Figure 9.5 shows that the number of CPU cycles increases as the data volume increases, and we can see that linear regression fits the measured values quite good. The red line in the graph is found through linear regression, and this line match the equation 9.11, which is the parameterization of function pattern f_3 for the database on the new node.

$$f_{3,DB} = 764.97 * Text\_size + 316605 \tag{9.11}$$

9.4 Function pattern f_4

To parameterize this function we need to measure how much data that is read on disk, and we need to know the data block size used on the disk. The disk measurements proved to be very hard as we had problems with our measurement tools, and this is explained in Chapter 8.4. As a consequence of this we were unable to perform a complete parameterization for this pattern for all the test components.

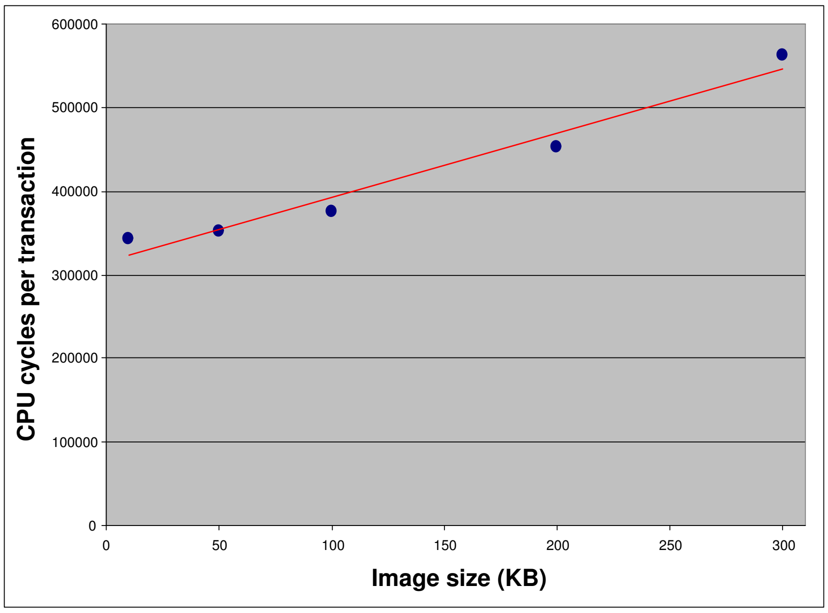


Figure 9.5: CPU cycles used vs. text size on database server

New web server

We have some results for disk reading on the new web server, see Figure 9.6. The graph shows that there are readings on disk for the large image sizes. The reason there is only reading on the large image sizes and that may be because on the small image sizes all the images are in memory during a test run, but the memory is not big enough to have all the large images in the memory during a test run (this is further explained in Chapter 8.4). Ideally, we should have eliminated the caching completely to get read result for all image sizes, but this was unfortunately not feasible, and this is further discussed in Chapter 6.6.

Since the measurements are uncertain, the parameterization is uncertain as well. However, we make a parameterization based on the measurements on the new web server. The values on the image size from 10 KB to 400 KB are all zero, and therefore we can only do a parameterization within the range from 500 KB to 1000 KB. Through regression in this range we have found the regression function to be: $y = 60.2 * Image\_size - 9120.6$. By dividing this by the block size on disk (512 KB), we found the number of read blocks by the complexity function in Equation 9.12, which should be the parameterization of function pattern f_4.

$$f_{4,newws} = \frac{60.2 * Image\_size - 9120.6}{512} \text{for } Image\_size \geq 500KB \tag{9.12}$$

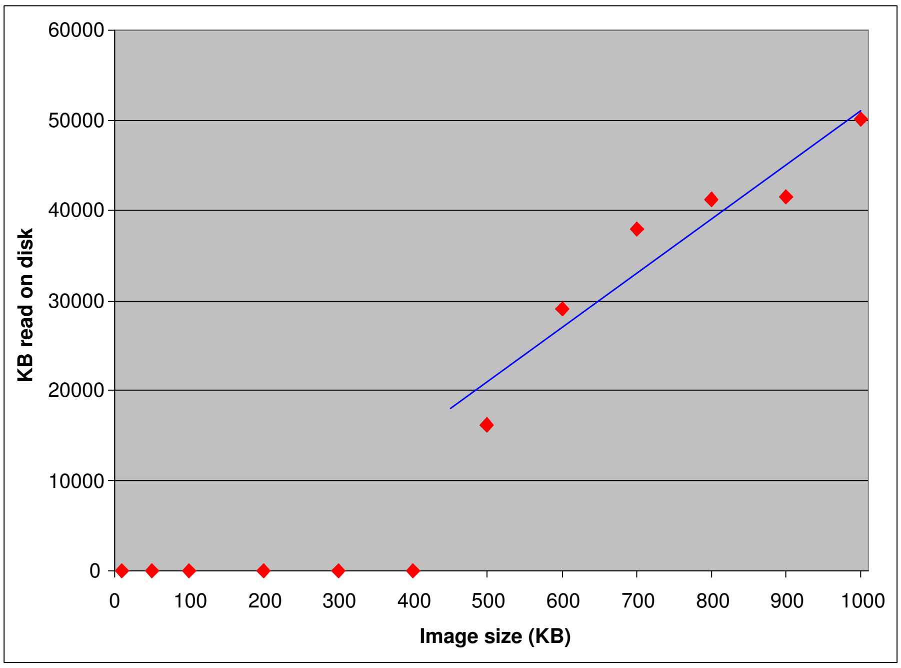


Figure 9.6: Disk read on new web server

Chapter 10

Dynamic model

In this chapter we will construct and solve a dynamic model for our systems. While a static model devolves the resource demands for the components in a given system, it can not be used for predicting the performance for a given load. A dynamic model combines the static work with a load, and can be used to predict the performance for a system. We will use a method outlined by Peter Hughes and Gunnar Brataas.

10.1 Background theory

We will model our dynamic model as a Queue Network (*QN*), that is a network of interconnected queues, where a queue stands for a resource centre (e.g. disk, CPU or network) and the queue for that resource. A queue is characterized by a function *S(N)*, which represent the service time for a given queue length n. There are three types of resources.

- *Load-independent resources*, where the service time, *S(n)* is independent of the number of users in the queue.
- *Load-dependant resources*, where the service time, varies according to the number of users in the queue.
- *Delay resources*, which represent situations where there is no queuing. And where the total time spent in the resource is the requests service time.

Not all of the requests in a QN network need to be identical in terms of resources used and resource demand. It is therefore possible to have different classes of requests, each with its own resource demand and different workload intensity parameters (e.g. inter-arrival time). E.g. one might imagine that one type of requests are more CPU resource consuming than other, e.g. large SQL queries compared to small ones.

The classes can either be open or closed. If all classes in a QN are open then we call it an open QN, if all are closed then it is a closed QN. It is also possible to have a mixed QN, where some classes are open and some closed. An open QN does not put any constraints on the maximum number of requests present in the system, while the closed have a finite number of requests in the system. According to [16] a closed QN can arise when we want to model a system with a maximum degree of multiprogramming. Whether a QN is open or closed affects the way the model is solved, either way it is solved through fairly easy operational laws.

One of the main advantages with QN models are that they are fairly easy and quick to solve. In order to solve our QN, we have to assume that it is *separable.* According to Peter Hughes [8], a separable network is one where the customers and the resource centres performs in a way which the performance of each centre can be found in isolation from the rest of the network. "In practice most systems violate the assumptions required for separability, but the resulting inaccuracies are often less significant than other errors"[8]

10.2 Method

Peter Hughes identifies in [8] the three normal stages in using a system model for scalability studies as; *construction stage*, *projection stage* and *validation stage.*

In the *construction stage* the baseline model is build, parameterized and its workload identified. We then validate the model against performance data, in our case the overall response time for one *Get image* request. The *projection stage* contains a modification analysis, where we identify the difference between the baseline system and a projected system. We then try to asses how these changes affect the input parameters for the projected model. The final stage in the process is the *validation stage*, where we verify any changes in the model.

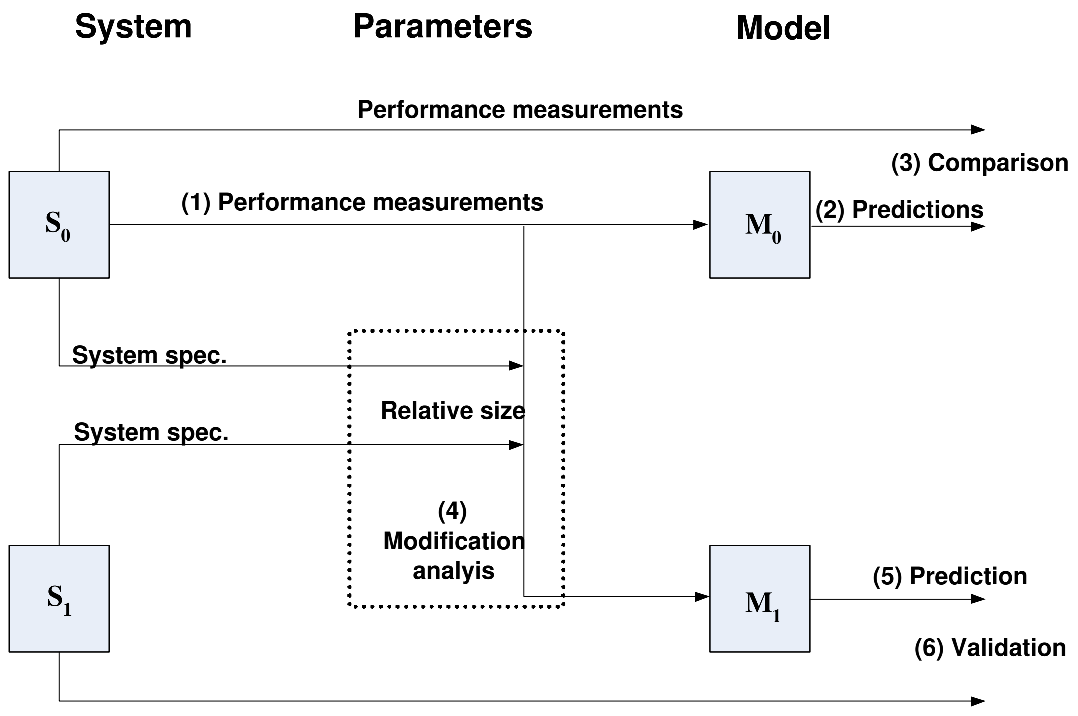


Figure 10.1: Scalability method (Peter Hughes and Gunnar Brataas, 2005)

Figure 10.1 is a more schematic overview of the processes involved in a scalability assessment using dynamic models, and it gives a clear overview of the processes involved in the *projection stage* and the *validation stage.* We will use the method outlined in this figure to aid us in assessing the scalability using dynamic models. As we can see from the figure the method consists of six steps.

In our thesis we have two different systems, one running on our old node; from now on denoted as S_0 (baseline system), and one system running on new nodes (S_1).

Unfortunately we do not have the disk parameterization for S_0, and this prevents us from performing stages *(2) Prediction* and *(3) Comparison.* We can however calculate an approximation of the disk residence time for S_0, to use in the later stages of the method. This can be done because we have all the input parameters to model M_0 but one. In addition, we have the output parameter for M_0, and then are left with a simple equation with one unkown (disk residence time), which can be solved.

Because we have found an approximation for the disk on S_0, we can find the disk residence time for the disk on system S_1, since we know this disk is 10% faster (see Chapter 6.1) . We will then have a complete set of input parameters to model M_1.

The steps we will perform are described in the list below.

- *Model construction stage:* We will construct a dynamic model for the systems; S_0 and S_1 (see section 10.3)
- Find the input parameters to M_0
- Perform a modification analysis between system S_0 and S_1 stage (4) in Figure 10.1.
- Execute stage (5) in the figure, and calculate the predicted output values for model M_1.
- *(6) Validate* the predicted output parameter[1] from model M_1, against the performance measurements for system S_1.
- If the output created by M_1 does not match the performance measurements for system S_1, we will try to identify the reasons for this.

10.3 Model construction

Model construction is always a trade off between precision and cost. A more precise model is more costly to make and solve, while an imprecise model is easy to make and solve, but not might not be accurate enough to use for predictions.

Based on the discussion in Chapter 3.2 on page 9 about how our system prosesses a request, the system can be represented as the QN shown in Figure 10.2 on page 72. We have chosen to represent the system as a closed model, because the load generator produces a fairly constant number of requests. This implies that in our test runs, the number of users in the system is more or less constant. It should be noted that Internet sites normally are modelled as open systems, but because of the way we conduct our experiment it must be closed. The LAN is represented as a load dependant device, because the service time in the network increases at high network utilizations.

[1]The overall response time for one image request

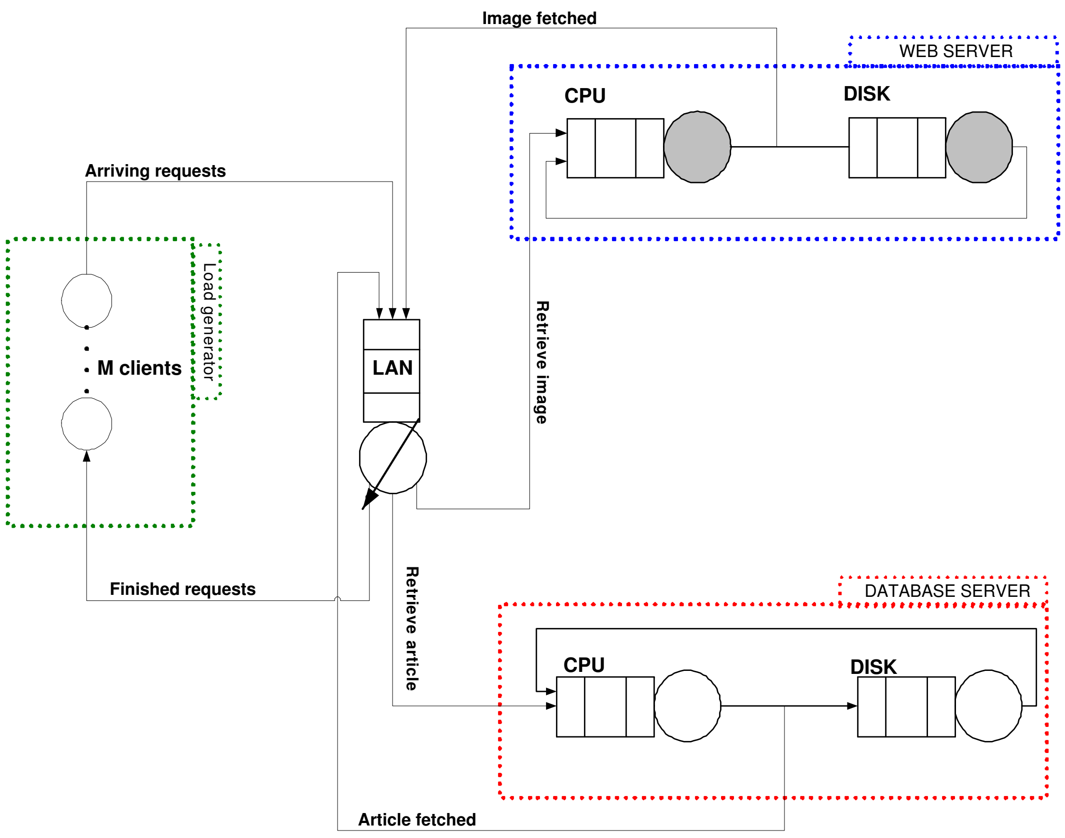


Figure 10.2: Closed QN model of our system

This is a fairly complete model, but unfortunately we do not have enough information to solve it, as we lack information about the IO on the database. We also know that the database server handles more requests, and is therefore less likely to be the bottleneck, and we will therefore choose to focus on the request image part.

The system breakdown can be seen in Figure 10.3, and this represent our system model for system S_0 and S_1.

We have a single-class network, with 10 sets of input parameters. This is because we have 10 image sizes from 100KB to 1000KB, all with different parameters

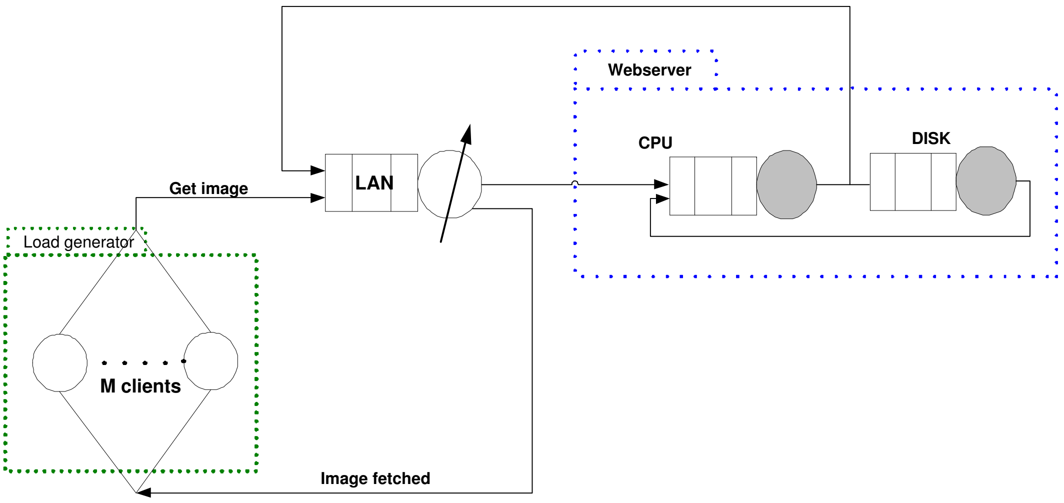


Figure 10.3: Breakdown of the contention model

Since our QN is closed, it can be solved through *Mean Value Analysis* (MVA). MVA is a recursive process based on three equations (*Residence time equation* 10.1, *Throughput equation* 10.2 and *Queue length equation* 10.3)[2].

$$R^{'}(n) = \begin{cases} D_i & \text{for delay resource} \\ D_i[1 + n_i(n-1)] & \text{for queuing resource} \end{cases} \tag{10.1}$$

$$X_0(n) = \frac{n}{\sum_{i=1}^{K} R_i^{'}(n)} \tag{10.2}$$

$$n_i(n) = X_0(n) x R_i^{'}(n) \tag{10.3}$$

10.4 Model solution

In this section we will follow the stages identified in Figure 10.1, for assessing the scalability using dynamic models. We can however, not perform stage *(2) Prediction* and *(3) Comparison*, because we need to use the performance measurements and the parameters measurements for system S_0 to approximate the disk demand.

[2]To see how these equations are derived, see Chapter 9 in [16]

(1) Parameters measurements

This stage involves determining the input parameters to the dynamic model M_0, and it is important to note that there is one set of parameters for each image size. Our model consists of three resources which needs parameters, and these have been found through measurements and approximations.

We will now explain the input parameters for model M_0, and how they were quantified. All the input parameters are given as residence time, $R^{'}$.

- $R^{'}_{cpu}$: This one is found by dividing the number of CPU cycles used for a given image size on the CPU clock speed. These results are discussed in Chapter 9.
- $R^{'}_{LAN}$: This is the time spent in the network by one request. This time consists of two main parts; the time used for establishing and releasing the connection, and the transfer time. From the load generator we get the establish and release time. We assume the transfer time is close to zero, as it is small files which are transferred over a 1Gbit/s Ethernet.
- $R^{'}_{disk}$: This is found through approximation, as we do not have any measurements for it. We find this by subtracting $R^{'}_{cpu}$ and $R^{'}_{LAN}$ from the overall response time. This is a bit cyclic, but it is the only way we can find an estimate. The impact of cyclic calculation is that the approximation for the disk is highly dependant on the accuracy of the other parameters.

The values for these input parameters to M_0 can be seen in Table 10.1.

Image size (KB)	$R^{'}_{disk}$	$R^{'}_{cpu}$	$R^{'}_{LAN}$
100KB	1.70 ms	2.80 ms	7.00 ms
200KB	2.38 ms	4.62 ms	12.50 ms
300KB	6.97 ms	5.54 ms	11.00 ms
400KB	6.35 ms	7.65 ms	7.50 ms
500KB	6.34 ms	9.66 ms	6.50 ms
600KB	10.22 ms	10.78 ms	9.50 ms
700KB	11.77 ms	12.23 ms	10.00 ms
800KB	10.75 ms	14.75 ms	8.50 ms
900KB	10.98 ms	17.02 ms	7.50 ms
1000KB	13.46 ms	18.54 ms	7.50 ms

Table 10.1: Input parameters to model M_0

(2) Predictions and (3) Comparison

According to the method these stages should compare the performance measurements (overall response time), against the output of model M_0 calculated in stage **(2) Prediction**. Since we are missing the disk demand we are unable to do a complete prediction, we have to use performance measurements R to find the residence time on disk ($R^{'}_{disk}$).

(4) Modification analysis

The modification analysis describes the relative size of the system specifications between systems S_0 and S_1. This analysis is performed in Chapter 6.1 on page 34. The CPU is expected to be 2.4 times faster, and the disk is expected to be 10% faster in system S_1 than in S_0. The network performance (R'_{LAN}) is believed to be the same, as they have the same network card. However, some differences between the two systems might occur as the new system has a higher throughput on small image sizes, which can lead to more queuing in the network. Network performance can also be dependant on the CPU, so we might see that system S_1 have better performance than S_0.

(5) Prediction

Based on the input parameters to M_0 (see Table 10.1) and the modification analysis we can predict the input parameters to M_1. This is done by dividing the input parameters for M_0 with the scale-up factor for each parameter found in the modification analysis.

E.g. the R'_{CPU} for M_1 is found by dividing the R'_{CPU} for M_0 by 2.4, as we assume that the new CPU will execute 2.4 times faster. Based on these calculated input parameters for M_0, we can solve the model, and find the predicted output parameters for system M_1.

(6) Validation

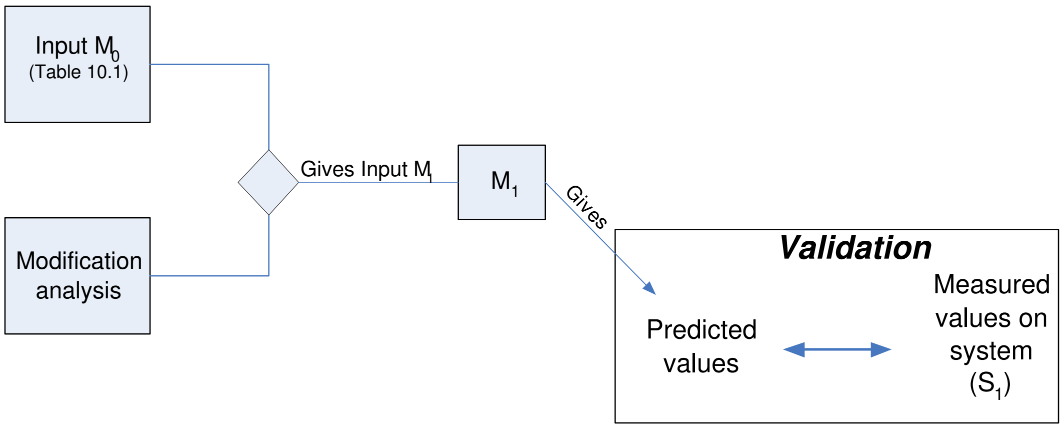


Figure 10.4: Explanation of validation process

In this stage we compare the predicted output values for M_1, with the measured values on system S_1. This is done to check to what degree we have managed to find a model which can produce sound predictions for the system (S_1). The output parameter is the overall response time for a single *Get image* request.

Table 10.2 on page 76 shows the predicted overall response time and the measured response time on S_1. As we can see, the deviation between the two system are quite large, thus our model can not be seen as very accurate. There are many possible error sources for these deviations, and one of the most probable is the measured overall response time itself. These measurements are quite uncertain as they are produced by Hammerhead2, and because we do not have total control on how Hammerhead2 calculates them.

Image size	Predicted overall response time	Measured overall response time
100KB	10ms	22 ms
200KB	17ms	23 ms
300KB	20ms	34 ms
400KB*	16ms	N/A
500KB	16ms	19 ms
600KB	23ms	13 ms
700KB	22ms	12 ms
800KB	24ms	12 ms
900KB	24ms	13 ms
1000KB	27ms	14 ms

*Table 10.2: Comparison between measured and predicted overall response time for model M_1 (*We did not manage to get any good measured values for 400 KB)*

To locate the probable errors sources, we can compare the predicted input parameters to M_1 to the one we have parameterized for CPU and LAN.

Validation of R'_{CPU}

Table 10.3 shows the predicted and the measured R'_{CPU}. And as we can see the predictions are quite good, it is the same magnitude. As the measured values is lower than predicted, the CPU in system S_1 is performing better than expected.

Image size (KB)	Predicted (ms)	Measured (ms)	Deviation (ms)
100KB	1.17 ms	0.63 ms	0.54 ms
200KB	1.93 ms	1.22 ms	0.71 ms
300KB	2.31 ms	1.75 ms	0.56 ms
400KB	3.19 ms	2.30 ms	0.89 ms
500KB	4.03 ms	2.82 ms	1.21 ms
600KB	4.49 ms	3.36 ms	1.13 ms
700KB	5.10 ms	3.89 ms	1.21 ms
800KB	6.15 ms	4.43 ms	1.72 ms
900KB	7.09 ms	4.97 ms	2.12 ms
1000KB	7.73 ms	5.50 ms	2.23 ms

Table 10.3: Comparision between predicted and measured R'_{CPU} M_1

Alternative scale-up on CPU

The results for the CPU presented in Table 10.3, shows that the predicted values are higher that the measured values. Because there is a almost constant error ratio, this can idicate that the actual scale-up factor of the CPU should have been higher.

In Table 10.4 we propose new scale-up factors between the input parameters to S_0[3] and

[3]These can be found in Table 10.1 on page 74

the measured value for system S_1. From this we can conclude that the actual relation between them is around 3.4.

Image size (KB)	Real scale-up factor based on measurments on S_0 and S_1
100KB	4.4
200KB	3.8
300KB	3.2
400KB	3.3
500KB	3.4
600KB	3.2
700KB	3.1
800KB	3.3
900KB	3.4
1000KB	3.4

Table 10.4: Real scale-up factor between system S_0 and S_1

Validation of R'_{LAN}

Table 10.5 shows that for most of the image sizes the prediction for the residence time (R'_{LAN}) is quite poor. From Table 10.5 we see that the measured residence time, drops abrubtly from image size 400KB to 500KB. It would be wrong for us to draw any conclusions on the reason for this drop, as the measured values are so uncertain.

It is quite clear that much of the inaccuracy in the dynamic model is caused by the measurements on this component. The reason we have presented the data in Table 10.5 is to show that they can account for much of the error in the dynamic model, and not because the results are useful.

Image size (KB)	Predicted (ms)	Measured (ms)
100KB	7.00 ms	14 ms
200KB	12.50 ms	9.75 ms
300KB	11.00 ms	11.50 ms
400KB	7.50 ms	14.20 ms
500KB	6.50 ms	1.00 ms
600KB	9.50 ms	2.00 ms
700KB	10.00 ms	1.00 ms
800KB	8.50 ms	1.00 ms
900KB	7.50 ms	$\approx$ 0.00 ms
1000KB	7.50 ms	$\approx$ 0.00 ms

Table 10.5: Comparison between predicted and measured R'_{LAN} for M_1

10.5 Evaluation

It was an open question whether or not it would be possible for us to construct and solve a dynamic model for the systems, and we think it still is an open question. The precision of dynamic QN models are totally dependant of the quality of the input parameters and the control parameters. We feel that while some of our input parameters are good (like the CPU), some are highly uncertain. E.g. the total response time for system S_0 and S_1.

We believe it would have been easier to get good results if we had used equal response time instead of equal utilization, because then we would had far more reliable measurements on the overall response time.

The project time line has also been very tight and this has put a limit to the number of measurements we were able to do. And the overall response time measurements has suffered of this. However, we feel that since our experiment is easily reproduced, it should be quite easy for someone to finish the job with the dynamic model.

Chapter 11

Discussion

In this chapter we will discuss the main findings in our thesis, and look at them in connection with the scaling scenarios outlined in Chapter 5. We will also explain the implications these findings have on read-intensive web sites (E.g. TV2i which was looked at in our pre project [21]).

11.1 Impact of increased article size

We have performed our experiments on three different components:

- A web server running on an old node
- A web server running on a new node.
- A database server running on a new node.

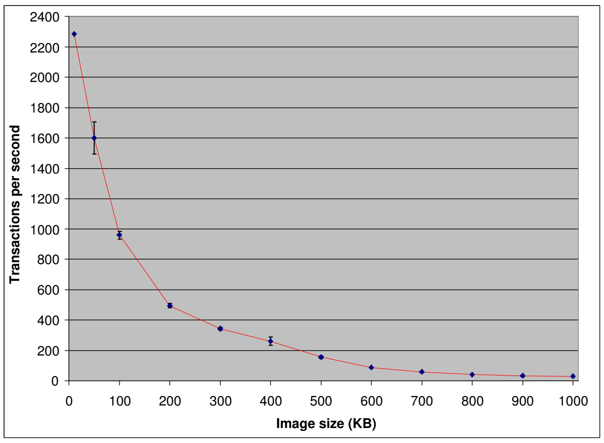
On all of these three components, we can see a distinctive reductions in the throughput[1] (Tps) as the image/text size increases. This can be seen Figure 11.1 on page 80. This effect was expected, but it is larger than we initially though, especially on the small image sizes on the web servers.

In Chapter 5 we pointed out that we will use image size 100 KB and text size 10 KB as a starting point for our discussion. The measurements on the new web server, see Figure 11.1(a), show that throughput is reduced to one fifth when the image size is increased from 100 KB to 500 KB. This is a considerable reduction in the throughput for a web site. On the old web server the throughput is reduced by almost one fourth, when the image size increases from 100 KB to 500 KB. The reason why we use choose to focus on this interval is because this is most probable image size in near future.

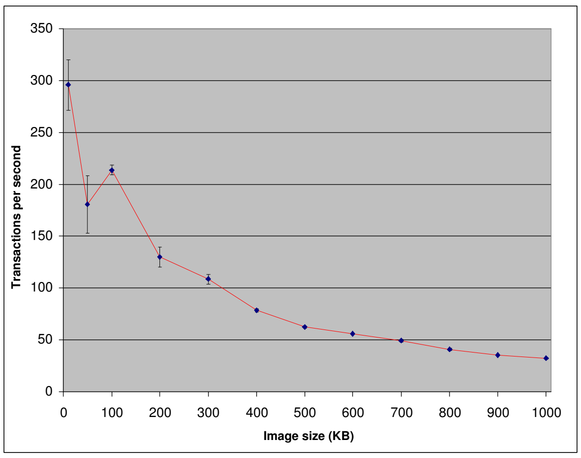
As we can see from the Figures 11.1(a) and 11.1(b), the throughput is reduced more for the small image sizes than the large ones.

On the database the throughput is reduced by about 25% when the text size is increased from 10KB to 200KB (see Figure 11.1(c)). This indicates that the database are much

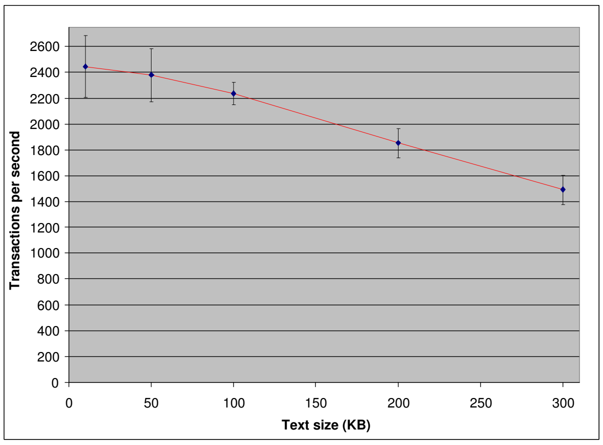
[1]Measured by number of request served by second (Transactions per second)



(a) Tps per image size on new web server



(b) Tps per image size on old web server



(c) Tps per article text size on the database

Figure 11.1: Throughput on the old and new web server and the database

more robust for future increases in text size than web servers in image size. To confirm this, we can compare the decrease in throughput between the two in the same interval, 100 KB to 300 KB. In this interval the database were reduced by 33% while the web server is reduced by 62%.

Implications for news sites

Most of the web sites interviewed in our survey (see Chapter A) predicts an increase in the article size, and particular in the image size. Although, it is common knowledge that increased article/image size leads to decreased throughput on the database/web servers, this has previously not been quantified.

Our findings shows that the first step from 100 KB to 200 KB have an bigger impact than the next step from 200 KB to 300 KB. The trend is that the impact of increased image size is getting smaller as the image size increased. So a increase from 900 KB to 1000 KB has less impact on the throughput.

At the same time as the image size is increased, the web site is expected to serve the same number of customers (throughput). This implies that unless the news sites have spare capacity on their current system, they will have to make alternations to their system, which can be costly.

The need for larger articles should be closely weighted up against the cost involved in handling the increased article sizes. There are two main solutions on how a system can be altered, that is either through *replication* or *scale-up*. Replication is to use more hardware at a given level (e.g. install a second CPU in the server), while a scale-up is to improve the hardware at a given level (e.g. more memory, better CPU, etc). We have in our thesis explored the effects off the latter solution, namely a scale-up. Looking at scale-up is more interesting, as replication is easier.

11.2 Scale-up

In our experiment we measured on two web server components, where the main difference between them was the CPU performance. In Chapter 6.1, we identified that the new CPU had a scale-up factor of 2.4, while the disk had a scale-up factor of 1.1. This does not mean that we expected the new web server to have 2.4 times higher throughput, as the throughput could depend on several factors (e.g disk and memory size). However, the throughput on the new web server was expected to be higher than on the old web server.

As can be seen in Figure 11.2 on page 82, the difference between the throughput on the new and the old web server decreases as the image size increases. On the largest image sizes the throughput are more or less the same.

If we study the interval from 100 KB to 500 KB, there is a considerable difference in the throughput between the old and the new component, with an average difference factor of 3.4. The throughput in this interval is mainly dependent on the CPU as there is no reading on disk, see Chapter 8.5, and the CPU is the bottleneck. This explains this factor, however a factor of 3.4 is 70% larger than our expected scale-up factor of 2.4. This indicates some super linear effects. It is interesting to note that that this factor of 3.4 was found in

Chapter 10.4 as well, these two findings indicates that perhaps the scale-up of 2.4 is too low.

The reason for the large difference on the small image sizes is because the new web server execute each connection more effectivly than the old web server. Since each connection is read more efficiently from memory, the difference in the throughput will be much larger with many connections than few connections. The difference in throughput increase proportionately with the number of connections. There is no difference on the large image sizes since the disk is the bottleneck device, as there is almost no scale-up on disk.

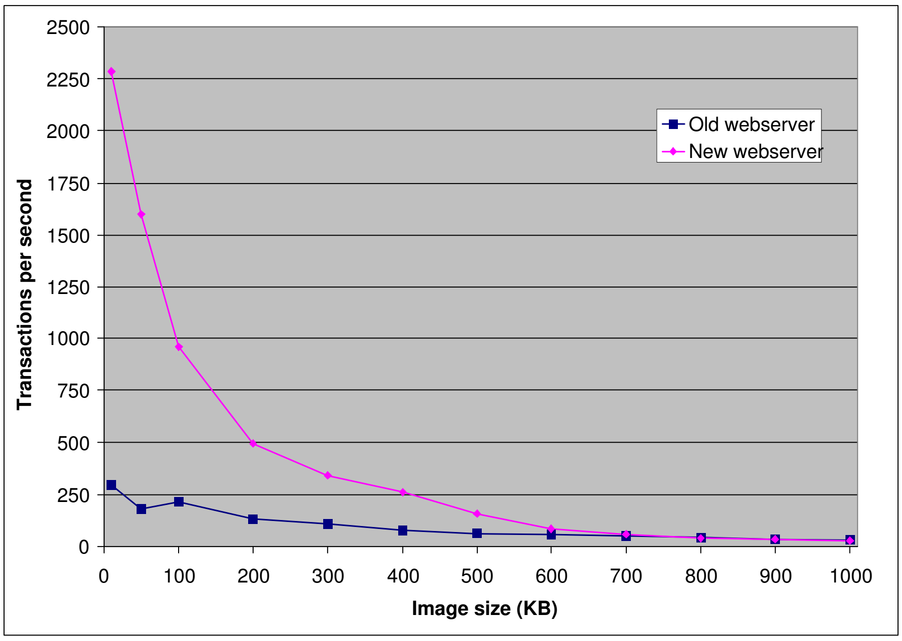


Figure 11.2: Comparison of throughput on old and new web server vs. image size

Implications for news sites

The fact that a scale-up of the processor leads to a considerable increase in throughput, is important for news web sites to keep in mind. For them it means that upgrading the hardware will help them in serving more customers per server. When we are performing a scale-up, it is important to identify the bottleneck in the system, as a scale-up is only possible if the bottleneck component either is replicated with more components or replaced by a better performing component. What this implies is that the performance of the overall system does not improve unless the performance of the bottleneck component is improved.

However, it is hard to predict the effects a scale-up of one or more components (a non-uniform scaling) will have on the throughput. This is because the throughput have different levels of dependency on the components on different image/text sizes. As we have seen the effects of the scale-up on the throughput varied between the different image sizes (a increase in throughput by 4.5 on 100 KB, but only an increase by a factor of 3.2 on image size 300 KB).

11.3 Connection between web server and database

Another interesting finding in our experiment is that the web server has a much lower throughput than the database server on identical hardware. So on a news site, the web server component would be the bottleneck. In the same way a disk or a CPU can be replicated, a web server node could be replicated, so that two web servers uses the same database. In this section we calculate how many web servers a database server can support on identical hardware.

Scenario calculations In this example we will use an imaginary news site with two 100 KB images per article, and a article text size of 10 KB. We will let this imaginary system run on the new nodes, and use the measurements data for these. The new web server can serve 957[2] 100 KB images per second, while the database can serve 2245 articles of text size 10 KB per second.

Since each article contains two images the web server can only serve 468 ($\lfloor 957/2 \rfloor$) articles per second. In addition, the web server must forward the *Get article* request to the database, which requires some resources. We can therefore estimate that each web server can serve ≈ 400 articles per second. This means that each database server can serve five web servers ($\lfloor 2245/400 \rfloor$), given that there is no non-linear effects.

Another point, is that the news sites find it more likely to experience an increase in image size than in text size (see Appendix A), and this means that each database can serve even more web servers.

Implications for news sites

For a news sites these findings are important because it implies that it possible to replicate at the web server layer. And if there are no non-linear effects when connecing more and more web servers to a database, it should be fairly easy for news site to calculate at what point they need to make changes to their database.

11.4 Capacity planning

Figure 11.3 on next page shows a near linear relationship between throughput and CPU utilization for each of the image sizes and this was expected. On image size 100 KB the graph can be interpreted of having a polynomial shape, but we feel a linear tred line fits even better. However, it should be noted that while the throughput vs. utilization is a near linear relationship, response time vs. utilization is not linear. And response time is very important for websites, as the customer should never have to wait too long.

Since the throughput vs. utilization relationship is a near linear one, it is easy for web site to predict how increased number of user will affect the web site. If we study the graph in Figure 11.3 we can imagine two example scenarios.

[2]All numbers are for a operating point of 60% on the CPU

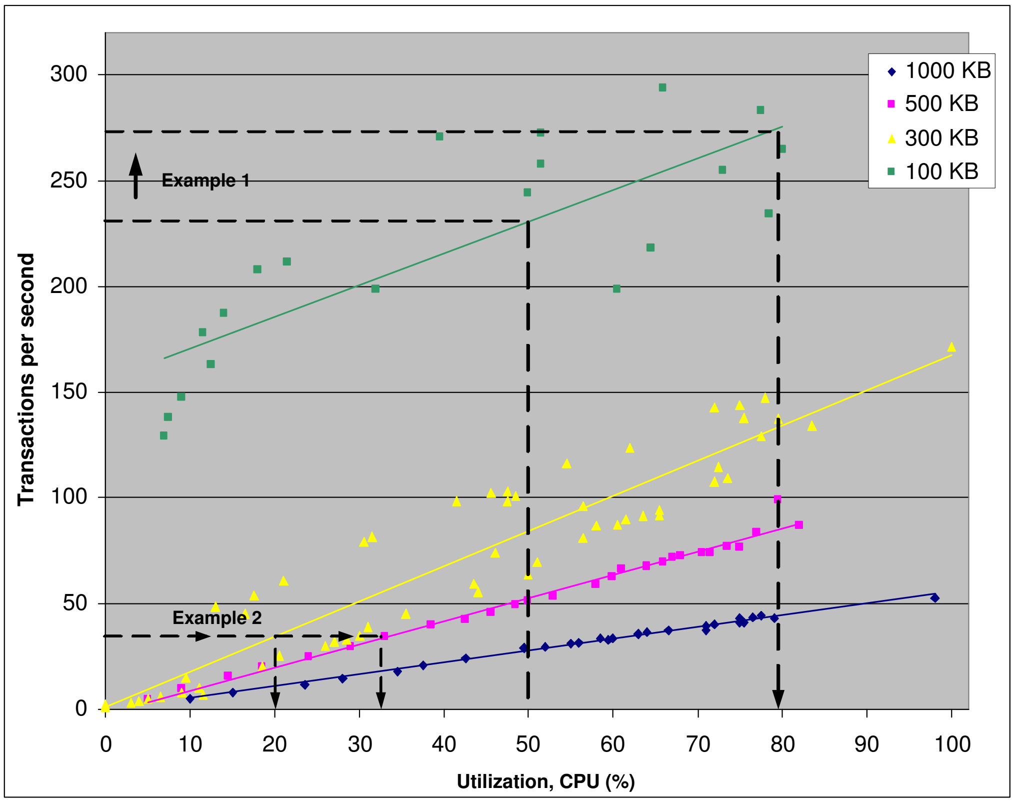


Figure 11.3: Throughput vs. utilization on the old web server

Example 1 If news sites currently use images size of 100 KB and run at 50% CPU utilization, the graph in Figure 11.3 shows that the throughput is 240 Tps. Now if this web site want to have a throughput of 270 Tps, then they can use the regression line to find the new utilization, in this case it will be around 80%.

Example 2 Now if another web site with image size 300 KB and throughput of 40 Tps want to increase the image size to 500 KB but keep the same throughput. They can then use Figure 11.3 to find how this will affect their utilization. From the figure we see they currently have a CPU utilization of 20%, but that it will raise to around 30%.

Implications for news sites

As we have seen in example 1 and 2, the linear relationship between throughput and utilization makes it easier for web sites to plan the implications of increased load and/or increased image/text size. However, it is important to remember that the response time will also increase as the throughput and the utilization increases. This will on some point be bigger than the customers can accept, and this point should be identified.

Chapter 12

Method evaluation

We will in this chapter evaluate the work and the methods used in this thesis, based on our experiences throughout the thesis.

12.1 SP method

We have used the SP method to model the components and the connections between the components in our test systems. The SP method provides a useful framework that really aids in understanding the system and the connections between the components. As with all modeling tools, finding the right granularity is hard. It is always desirable to model the system with a high granularity, but this requires deep domain knowledge about all the parts in the system. In addition, a fine model is costly to solve. On the other side, a too coarse model limits the applications of the model.

We are happy with the level of detail in our model, as we have tried to get a model which we could understand completely and be able to solve. Even though our model looks quite simple, getting a full grasp of it required a lot of domain knowledge. This is one of the problems with scalability assessments; it is a domain which requires deep knowledge in many areas (hardware, software, programming, statistics, etc.).

12.2 Resource function workbench

Making a framework for automating the collection and processing of the measurements, has together with the installation and setup of the components been the most time consuming processes during this thesis. Making the framework took about five weeks to complete, where the main activity was scripting.

Although, this was a secondary objective according to our problem definition, it was fundamental in order to be able to perform and manage the measurements. In addition, this part is highly useful as it makes it easy to recreate our measurement, and could potentially be very useful for future projects within this area.

12.3 Measuring process

Even with the resource function workbench, the measuring process has been quite demanding and time consuming. As measuring is a time consuming process we had to perform less measurements that we would have liked. We would recommend future projects in this area to use the pre project before Christmas to establish the system baseline and a the framework for the measurements; this way they will get more time to perform and interpret the measurements.

During the measurements we have encountered some challenges with choosing the right load generator, and we have actually tested out two load generators before the one we ended up using. This was just bad luck, as there was no way of knowing their shortcomings before we had tested them.

We also had some challenges specific to the new system. The database component needed two load generators and two web servers in parallel to reach the decided operating point. It took some time to configure these components to work in parallel, as we needed to merge the results of the two individual measurements.

Looking back, there is a possibility that we should have used equal response time to determine the operating point, both because response time is an important factor on the Internet, and because it makes it easier to solve the dynamic model.

However, we found it more feasible and more reliable to use the equal utilization. We focused the measurements on the CPU which was the expected bottleneck in the system, and we choose the operating point to be 60% CPU utilization. On the small file sizes this operating point required a very high number of connections, and we could possibly have chosen a lower operating point, which could have made our measurements a bit easier.

Even though the measuring process took much time, we are very satisfied with the results, especially the CPU and throughput measurements. Later tests have proved them to be highly reproducible, and thus we can conclude that these measurements are reliable and sound.

12.4 Dynamic model

To fully exploit the dynamic model, the response times and residence time on each component in the system are required. Since we were unable to measure the residence time on disk, the model lost much of its precision. In addition, when we are uncertain of the measured overall response time, the model looses some of its value.

One of the challenges with using a dynamic model in a scalability assessment is to perform a good modification analysis. We have focused on the major components in the system and found the relative size between them. A complicating factor in this process was the fact that we have non-uniform scaling, that is that the scale-up is different between the components.

We feel however, that creating a dynamic model has been very useful, and we believe it can make valuable predictions if the accuracy of the parameters are improved. This should be feasible, as our measurements can be easily recreated.

12.5 Work process

We are happy with the way the work process of this thesis has been conducted. We early understood the value of a structured approach, as the task was very comprehensive and had to be completed in a short time span. Because of this, we started the project by breaking down the work into smaller parts and setting a deadline for each part. We understood that the writing would have to be a continuous task, so we have been writing throughout the entire project.

Table 12.1 show how much time we have spent in the different phases during our work process. The table shows the allocation of the work during the 20 weeks that were available. The most interesting thing about this table is to see that a large time of this project was spendt on system installation and the RFW. This work had to be done properly, as it created the fundation of our thesis.

The measurements took about 6 weeks to complete, and this may seem like a long time, but measurements are time consuming. Each individual experiment took about 5 minutes and we have nearly 700 of them. In addition, the cluster was unstable and down for longer periods of time.

Time – Work phase
3 weeks – Installation and system configuration
5 weeks – Resource Function Workbench
6 weeks – Measurements
3 weeks – Interpretation of measurements data
3 weeks – Writing

Table 12.1: How the work was distributed during our work process

Chapter 13

Conclusion

We have in this thesis continued the work started in our project [21], i.e. to explore the practical and economic feasibility of assessing the scalability of a read-intensive large-scale Internet site. To do this we have installed the main components in a news site using open source software. This scalability exploration has been driven by the scaling scenario of increased article size.

We have managed to assess the scalability of our system in a good way, but it has been more time consuming and knowledge demanding than expected. This means that the feasibility of such a study is lesser than we expected, but if the experiences and the method of this thesis are applied, such a study should be more feasible. We have assessed the scalability of a general web architecture, and this means that our approach can be applied to all read-intensive web sites and not just the one looked at in the [21]. This general focus is one of the strengths with this thesis.

One of the objectives in our thesis was to make a resource function workbench (RFW) that is a framework which aids in the measuring and data interpretation. We feel that our RFW is one of the most important outcomes from this thesis, because it should be easy to reuse, thus saving time for future projects and making the feasibility of such a study higher.

In Chapter 11 we discuss the main findings in our thesis, and one of the most important is that the impact of increased article size on the throughput is bigger than expected. A small increase in article size, especially image size, leads to a clear decrease in the throughput. This reduction is larger on the small image sizes that on the large ones. This has wide implications for news sites, as many of them expect to increase the article size and still use the same system.

Another major finding is that it is hard to predict the effects a scale-up of one or more components (a non-uniform scaling) will have on the throughput. This is because the throughput have different levels of dependency on the components on different image/text sizes. As we have seen the effects of the scale-up on the throughput varied between the different image sizes (a increase in throughput by 4.5 on 100 KB, but only an increase by a factor of 3.2 on image size 300 KB).

In our case we have performed a non-uniform scaling, where we have increased the CPU by 2.4 and the disk by 1.1 On some image sizes and text sizes, the overall throughput was increased by a factor 10, but on others there was almost no improvement. The implications this have for web sites, is that it is hard for them to predict how system alternations will affect the overall throughput. As it is dependant on the current image and article size.

It was an open question whether or not a dynamic model of the system could be constructed and solved. We have managed to construct the dynamic model, but the predictions it makes are a bit crude. However, we feel that creating a dynamic model has been very useful, and we believe it can make valuable predictions if the accuracy of the parameters are improved. This should be feasible, as our measurements should be easy to recreate.

This thesis has been very demanding, because scalability requires a wide field of knowledge (statistics, hardware, software, programming, measurements etc). This has made this work very instructive, as we have gained knowledge in so many different aspects of computer science. Ideally, the thesis should have a larger time span, as the there are so many time consuming phases, which would have been interesting to spend more time on. As consequence of this short time span there are some further work which can be conducted in order to gain further valuable knowledge.

- Perform more measurements with different cache levels to fully explore its effects.
- Look into the problems with the disk measurements on the old disk.
- Spend some time validating the response time, so that the accuracy of the prediction made by the dynamic model can be increased.
- Investigate the effects of a scale-up on the database.
- Measure on a system which allows the same scale-up on all major components (memory, CPU and disk).
- Continue the work of the resource function workbench, to make it less domain specific.

Bibliography

[1] Almeida, V., and Bestavros, A. Characterizing reference locality in the WWW.

[2] Brataas, G. Scalability Exploration at TV2 Interaktiv. June 2004.

[3] Brataas, G., and Hughes, P. H. Exploring architectural scalability. WOSP'04: Workshop on Software and Performance, California, USA.

[4] Buret, J., and Droze, N. An overview of load test tools.

[5] Christiansen, T., and Torkington, N. *Perl Cookbook*, second edition ed. O'Reilly & Associates, Inc., August 2003.

[6] Coar, K., and Bowen, R. *Apache Cookbook*, first edition ed. O'Reilly & Associates, Inc., November 2003.

[7] Dilley, J. Web server workload characterization. Hewlett-Packard Laboratories.

[8] Hughes, P. H. Lecture notes in performance engineering, IDI, 2005.

[9] Hughes, P. H. SP Principles. STC Technology Ltd, 1988.

[10] Hughes, P. H. Considerations relating to SP model paramaterisation. IDI working paper. October 2004.

[11] Hughes, P. H., and Brataas, G. Scalability of a work-station cluster architecture for video-on-demand applications. LNCS 1786, Springer Verlag, 2000, pp. 277 - 293.

[12] Hughes, P. H., Brataas, G., Fagerli, J.-A., and Landmark, O. C. Exploring the scalability of an enterprise architecture. Working paper, IDI, NTNU.

[13] Jain, R. *The art of Computer systems performance analysis.* John Wiley & Sons Inc, April 1991.

[14] Mainkar, V. Performance implications of security protocols. Retrieved on the 20 May from the World Wide Web:
www.cse.iitb.ac.in/∼varsha/ allpapers/softwarePerformance/ssl.pdf.

[15] Menascè, D., Ribeiro, F., Almeida, V., Fonseca, R., Riedi, R., and Jr, W. M. In Search of Invariants for E-Business Workloads. Proceedings of the 2nd ACM conference on Electronic commerce, Minneapolis, Minnesota, United States.

[16] Menascè, D. A., and Almeida, V. A. F. *Capacity Planning for Web Services.* Prentice Hall PTR, 2002. Metrics, Models, and Methods.

[17] Møgelstue, K. Hvem besøker nettstedene. From TNS-Gallup, retrieved on the 18 February, available online at:
http://www.tns-gallup.no/arch/img.asp?file_id=204861&ext=.ppt, January 2005.

[18] Newham, C., and Rosenblatt, B. *Learning the bash shell*, first edition ed. O'Reilly & Associates, Inc., 1995.

[19] Pitkow, J. E. Summary of WWW Charaterizations. *Proc. World Wide Web Conf.* (January 1999).

[20] Ruud, J., and Tveiten, O. G. Master thesis website. This is the website in connection to the master thesis, available online at:
http://www.stud.ntnu.no/groups/diplomgisleruud.

[21] Ruud, J., and Tveiten, O. G. Feasibility of scalability exploration for large-scale internet sites, November 2004.

[22] Vetland, V., Hughes, P. H., and Sølvberg, A. Improved parameter capture for simulation based on composite work models of software. To appear in proceedings of the 1993 Summer Computer Simulation Conference, Boston.

[23] Walpole, R. E., Myers, R. H., and Myers, S. L. *Probability and Statics for Engineers an Scientists*, sixth edition ed. Prentice Hall International, INC., 1998.

[24] Wang, H., and Wang, C. Open Source Software Adoption: A Status Report. *IEEE SOFTWARE* (March/April 2001).

[25] Woodside, M., Vetland, V., Courtois, M., and Bayarov, S. Resource Function Capture for Performance Aspects of Software Components and Sub-system. Published by Springer-Verlag Berlin Heidelberg 2001.

Appendix A

News sites survey

In order for us to make scaling scenarios which reflects real news sites, we contacted around 20 of the largest news sites in Norway, and asked them about the future growth of their website. The question and answers can be found below.

Some of our participants have asked to remain anonomous, but we have asked a a large variety of norwegian news sites, and we have mostly received answeres from the smaller players in the marked.

A.1 Key findings

It seems like almost all the newspapers agree that the total article size will increase a lot in the years to come, not so much because of increased text size, but because of increased use of multimedia. Almost all also predicts that the users will start using the Internet more frequently.

A.2 Participants

- TV2 Interaktiv **(TV2i)**, Pål Nisja
- Teknisk Ukeblad **(Tu)**, Tore Stensvold
- Stavanger Aftenblad **(Sa)**, Kjell T. Skurnes
- Anonymous **(An)**, medium size Norwegian newspaper
- Drammens Tidende **(DT)**, Lars Lager Espevalen
- ITavisen **(It)**, Tore Neset

Question: Is the size of the articles likely to increase in the years to come?
TV2i: Have noticed that articles with more multimedia (images, video, sound) content is more popular than other articles, because of this it is very likely that they will increase the number of images and video. But it is also likely they will try to write more in-depth articles, and move away from the short writing format on the web. They do not have an idea of how much, but because video and images is much bigger that text, a many doubling is likely.
Tu: Thinks the article size will increase, both in terms of text and in terms of multimedia content, mainly images and video.
Sa: Thinks the article size, in terms of text, will decrease on the web, while the articles which are published in the printed copy of the paper will increase. The article size, will however increase, as the number of multimedia content (images, video, sound) will increase a lot.
An: The text of some of the articles will increase, while other still will be quite short. Some articles will increase a lot, because of increased use of illustration (images, sound and video). He also predicts an increase in the quality of the multimedia posted, e.g. **higher resolution on the images** and video.
Dt: A distinct increase in the use of multimedia (images, sound, video), will lead to an increase in article size, but the text will stay more or less the same.
It: Finds it hard to predict, they try to write long if they have the opportunity, and writes short if the news must reach the marked. He does not mention multimedia (images, video and sound).
Summary: As we can see, almost every one expects a massive increase in the overall article size. It is not so much an increase in the text, but the increase will come as a result of increased use of multimedia (sound, images, and video).

Table A.1: Question 1 in survey

Question: Do you think web sites will begin to charge money for some of its content (Pay-per click, Pay-per view)?
TV2i: Yes, in form of extra services like WebTV, article archive, more in-depth articles. The general news will however be available for everyone
Tu: Do not believe in this, if so it had to be very special things.
Sa: Not for news in general. Might happen for special things.
An: Not for news in general, might work for niche products
Dt: Not for news in general, might work for niche products
It: Not relevant for us.
Summary: As we see, there is a wide agreement that charging for the news content is unlikely. However it might happen to niche products, like access to the archive.

Table A.2: Question 2 in survey

Question: Will the number of articles increase (number of articles published per day, number of articles on the front page, the number of articles in the archive.)
TV2i: They publish around 250 articles per day, and do not think that this is going to increase. They will rather work more with the articles they publish
Tu: Probably more articles per day, but it all depends on how much people are able to read. The number of articles (headlines) on the front page will stay the same
Sa: The number of articles on the front page will increase a lot, each article will have a much shorter time span. The news site will become much more dynamic, with more articles published a day. They think people will only access the news site for a quick glance at the news.
An: Expects an increase in the number of articles published a day. They also predict that the full archive will be closed with a pay per view solution.
Dt: The number of articles on the front page will probably stay the same, perhaps a moderate increase. The number of articles published per day, varies from day to day. At the most they published 38 articles, much more than this is not likely.
It: They see no reason for increasing the number of articles per day, the have 10 articles per day as an upper limit.
Summary: There is no clear trend here, all though most of them agree that there is an upper limit for how many articles which is reasonable to publish per day. This limit depends on the size of the web site, the larger the site, the larger the limit. Some of them believe in an increased number o f articles per front page

Table A.3: Question 3 in survey

Question: Do you believe the number of users will increase, or have we reached a limit on users in Norway. What do you believe the limit for your site is?
Tv2i: They believe the Internet will grow into a medium available to everyone (sex and age), and that people will use it more frequently (daily). They also believe that the time spent on Internet may double within the next 3 to 5 years.
Tu: Believes that large web sites is beginning to get close to the limit, but that smaller web pages is likely to experience further grow.
Sa: Believes in a large growth in the number of users, mainly because of the increased use of technical devices and thee increased use of broadband. Also thinks that people will access the same site more frequent (several times a day)
An: Thinks that we on a national level, will see an increase in the number of users and an increase in how often they access web sites. He believes that more people will read their web site on a daily basis, mainly because local web sites will become more important and more accessible for most people
DT: Finds it hard to predict, as this is closely connected to how you measure users. **It:** Find it hard to predict. Though they had reached the limit last year, but have experienced a 25% growth during the last year.
Summary: Most believe there is an upper limit of user in Norway, and that the largest web sites are beginning to get close to it. However most of them believe that the behavior of the user is likely to change. They expect that each user will access the site much more frequent.

Table A.4: Question 4 in survey

Appendix B

Open source software adaption

To generate load on the web system we needed a load generator which satisfies the requirements for the measurements. After searching the web, we found that there exists a huge number of load generators available, both open source and commercial products. For us, only open source generators are of current interest, and all commercial products was thrown out immediately. We found several interesting load generators, and picked out 6 to look further in to.

As we not know much about any of these we needed a quick evaluation to find the best fitted generator. Based on the open source software classification in [24], we defined a set of attributes and characteristics to decide what generator that fit our requirements best. By assessing a value to each of the attributes defined, we can specify a generator capability to meet the measurements requirements. The following list describes the characteristics and the possible values we can assign.

1. **Technical Support:** the amount of available support for the load generator. Available values:
   - – Support limited to direct and individual support from developer.
   - + Support based on community-oriented group support.
   - ++ Support from commercial entities providing comprehensive support for the load generator.
   - — No longer being developed or supported.
2. **Documentation:** the amount of available documentation from developer or other entities.
   - – No or minimal documentation.
   - + Documentation of the basic configuration and functionality, and not fully updated to the latest version.
   - ++ Complementary and thoroughly documentation and explanation of all the functionality and possibilities of the load generator.
3. **Backward compability:** the effort required by an existing system to maintain compability with the load generator. Available values:
   - – The load generator is either in its first stable release or its functionality has been modified such that systems using a previous version would require significant effort to upgrade to the current one.
   - + A moderate effort are required to upgrade to the current version.
   - ++ virtually no effort is required to upgrade to the current version.
4. **Binary Availability:** Official or unofficial binary distributions are available. Available values:
   - Yes.
   - No.
5. **Platform dependency:** If the generator requires a specific operating system. Available values:

- Unix
- Linux
- Windows
- BSD
- Multiplatform (Java-based)

6. **Software license:** The licensing format.

   - GPL - applies to all open source applications developed by the Gnu organization.
   - LGPL - covers the various libraries developed be the Gnu organization.
   - BSD - includes all derivatives of the BSD license, such as the X-Windows license "X".
   - Apache license - The Apache Software Foundation uses various licenses to distribute software and documentation, to accept regular contributions from individuals and corporations, and to accept larger grants of existing software products.

7. **Current development status:** Available values:

   - Development release: The load generator is still being actively developed and features added.
   - Stable: a stable, widely installed version of the load generator exists, with ongoing development efforts underway.
   - Discontinued: Load generator development effects have effectively stopped.

Load generator	Technical support	Documentation	Backward compability	Binary Availability	Platform dependency	Software License	Current development status	Notes
OpenSTA	+	+	++	Y	Windows	GPL	Stable	opensta.org
Hammerhead2	+	+	+	Y	Linux, Solaris, FreeBSD	GPL	Stable	hammerhead.sourceforge.net
Grinder	+	++	+	Y	Multiplat.	BSD	Stable	grinder.sourceforge.net
JMeter	+	+	-	Y	Multiplat.	Apache license, version 2.0	Stable	jakarta.apache.org/jmeter
Siege	+	+	+	Y	Linux, Solaris	GPL	Stable	joedog.org/siege
Dieseltest	-	-	-	Y	Windows	LGPL	Stable	sourceforge.net/projects/dieseltest

Table B.1: Load generator characteristics

Based on the evaluation of the load generator characteristics, shown in Table B, we decided to look further into Hammerhead2, Siege and Grinder. OpenSTA and Dieseltest required Windows platform and were excluded for that reason. JMeter was the weakest load generator from the characterization and was also excluded. Among Grinder, Hammerhead and Siege came Grinder out as the best alternative, and was our first choice.

Appendix C

RFW - File walkthrough

In this chapter all the files in the workbench will be introduced and explained. All the following files read the runconf file, so this is not listed under each file.

The main directories in our experiment are organized in the following way.

- conf
  - skeletonfiles
- scripts
  - initiate
  - start-stop-scripts
  - measurementscripts
  - measurementscripts_oldnode
  - DB
- measurement_logs

C.1 Conf files

This directory contains the configuration file for our experiment.

.current_run_path

- *Location:* $HOME directory
- *Purpose:* This is an important file, it tells the location of the runconf file. This is the only file in addition to the runconf.sh file, which needs to be altered if the location of the installation path is changed.
- *Reads:* N/A
- *Outputs:* N/A

runconf.sh

- *Location:* conf
- *Purpose:* This is the main configuration file which set all the global variables. This configuration file is sourced (read) by all the scripts, and this ensures that we only need to change the global variables in one place. If we e.g. move the MySQL server binary, we only need to change this in runconf.
- *Reads:* This configuration file, reads the *.reserved_nodes.sh* file, to get the IP adresses for the servers which we have reserved.
- *Outputs:* N/A

.reserved_nodes

- *Location:* conf
- *Purpose:* Is a file outputted by the *reserve-node.sh* script. It contains a list of reserved nodes.
- *Reads:* N/A
- *Outputs:* N/A

.reserved_nodes.sh

- *Location:* conf
- *Purpose:*Contains a list of export variables statements, on the format *export WEB-SERVER="192.0.0.0"*. This file is then read into runconf.sh.
- *Reads:* N/A
- *Outputs:* N/A

dummy.job

- *Location:* conf
- *Purpose:* This file is a dummy.job used in the *reserve-node.sh* script. This dummy job is sent to the clusters job scheduler, which reserves time at given nodes.
- *Reads:* N/A
- *Outputs:* N/A

C.1.1 Skeleton files

This directory contains the skeleton files, that is files which are used as templates and needs to be altered for each individual measurement, or in case some of the global variables are changed.

dummy.job.template

- *Location:* conf/skeletonfiles
- *Purpose:* Is used by the reserve node script, it is a dummy job which is sent to the cluster to hold then nodes.
- *Reads:* N/A
- *Outputs:* N/A

select.php.skel

- *Location:* conf/skeletonfiles
- *Purpose:* Is the skeleton file for the PHP page which is used to test the database. This skeleton file is copied into the apache directory, and the IP adress of the DB set.
- *Reads:* N/A
- *Outputs:* N/A

hammerhead.conf.skel

- *Location:* conf/skeletonfiles
- *Purpose:* This is the skeleton file for the hammerhead load generator. This file is copied to the hammerhead.conf file (see below) in the start of the experiment. When it is copied the IP adress of the target machine, logdirectory and some other parameters is set.
- *Reads:* N/A
- *Outputs:* N/A

hammerhead.conf

- *Location:* conf/skeletonfiles
- *Purpose:* This file is copied to a local configuration file for each measurement, where the right session and runtime is set.
- *Reads:* conf/hammerhead.conf.skel
- *Outputs:* N/A

http.conf.skel

- *Location:* conf/skeletonfiles
- *Purpose:* This is the skeleton file for the apache web server configuration file, this file is copied into the apache directory, and the IP adress of the web server is set.
- *Reads:* N/A
- *Outputs:* N/A

skel.scn

- *Location:* conf/skeletonfiles
- *Purpose:* This is a skeleton file for the scenario files used by Hammerhead. This file is read each time a scenario is created, we use around 2000 scenarios for each measurement.
- *Reads:* N/A
- *Outputs:* N/A

C.2 Scripts

This directory contains the script used in the workbench. All scripts read (sources) the *runconf.sh* configuration files, where all the global variables are set.

startexperiment.sh

- *Location:* scripts/
- *Purpose:* This is the main script for starting a new experiment. The sequence of the script can be seen in Figure 7.4 on page 48. What it basically does is to SSH into the reserved nodes, and starts the given service on that machine.
- *Reads:* N/A
- *Outputs:* N/A

stopexperiment.sh

- *Location:* scripts/
- *Purpose:* This is the main script for stopping an experiment. The sequence of the script can be seen in Figure 7.7 on page 51. What it basically does is to SSH into the reserved nodes, and stops the given service on that machine.
- *Reads:* N/A
- *Outputs:* N/A

measure.sh

- *Location:* scripts/
- *Purpose:* This is the main measurement script, here the experimentname and the test are made.
- *Reads:* N/A
- *Outputs:* N/A

measureoldnodes.sh

- *Location:* scripts/
- *Purpose:* This script is the same script as *measure.sh* except that is has been altered to work on old nodes.
- *Reads:* N/A
- *Outputs:* N/A

Dbmeasure.sh

- *Location:* scripts/
- *Purpose:* This script is the same script as *measure.sh* except that is has been altered to work on the database.
- *Reads:* N/A
- *Outputs:* N/A

C.2.1 Initiate scripts

This directory containts the initiate scripts which run just before or just after the experiment is commenced.

reserve-nodes.sh

- *Location:* scripts/initiate
- *Purpose:* This script has two arguments, number of nodes needed, and time needed. This scripts job is to reserve nodes in the Clustis cluster, so that you have exclusivity of the nodes in the period you run the experiment. This script is written by Geir Bostad (geirbo@idi.ntnu.no), and altered by us to fit our cause.
- *Reads:* runconf.sh, dummy.job.template
- *Outputs:* conf/.reserved_nodes in the conf dir. This file contains a list of the hostname of the reserved nodes.

transform-skel-files.sh

- *Location:* scripts/initiate
- *Purpose:* This scripts task is to copy the skel files into their given configurations files. It is done this way so all the changes can be done in runconf.sh. E.g. ipadress must be specified in all these files.
- *Reads:*
- *Outputs:*

mysql-data.sh

- *Location:* scripts/initiate
- *Purpose:* This scripts copies the mysql data to a local temporary directory on the Database node.
- *Reads:* runconf.sh
- *Outputs:* Mysql data directory in a temp directory on local node.

reserved_nodes_hack.sh

- *Location:* scripts/initiate
- *Purpose:* This is a small scripts, which translates the list of reserved nodes, from hostname to ipadress. The reason for this is that the apache server does not work if you address it with hostname. It outputs the file .reserved_nodes.sh, which list the different servers (Mysql, Web, Hammerhead2) with a given ipadress. This file is read by runconf.sh, so these variables are global.
- *Reads:* N/A
- *Outputs:* conf/.reserved_nodes.sh

C.2.2 Measurementscripts

generateload.sh

- *Location:* measurementscripts/
- *Purpose:* This is the script which initiates the load, and coordinates the measurement for an individual measurement. For further information abotu this script see section 7.2.2, and in particular figure 7.6.
- *Reads:* N/A
- *Outputs:* N/A

capturecpuusage.sh

- *Location:* measurementscripts/
- *Purpose:* This script is executed on the machine which is measured on, it runs vmstat and captures the cpu usage on the machine for a given time. It outputs time.log which is a timestamp used in the postprocessing, and vmstat.log which is a log file.
- *Reads:*
- *Outputs:* /home/jorgenru/source/hammerhead/time.log, /home/jorgenru/source/hammerhead/vmstat.log

captureiousage.sh

- *Location:* measurementscripts/
- *Purpose:* This script is similar to the capturecpuusage.sh except that is captures IO usage instead.
- *Reads:*
- *Outputs:* /home/jorgenru/source/hammerhead/iostat.log

copyimage.sh

- *Location:* measurementscripts/
- *Purpose:* This scripts copies the images of a given filesize to a local directory on the web server node.
- *Reads:*
- *Outputs:* Images for a given filesize in temporary directory.

formatmeasurement.sh

- *Location:* measurementscripts/
- *Purpose:* This script processes the log files outputtet by each individual measurement, and prints it to a human readable and parse friendly format.
- *Reads:* /home/jorgenru/source/hammerhead/*.log (log files outputtet by load generator, captureiousage and capturecpuusage.
- *Outputs:* measurement_logs/*EXPERIMENT_NAME*/FS*X*-S*Y*-RT*Z*, where *X*, *Y*, *Z* are parameters.

interpretmeasurements.sh

- *Location:* measurementscripts/
- *Purpose:* This script parses the logfiles outputtet by the formatmeasurements.sh script, into excel friendly format.
- *Reads:* measurement_logs/*EXPERIMENT_NAME*/*
- *Outputs:* measurement_logs/*EXPERIMENT_NAME*.txt

makescenario.sh

- *Location:* measurementscripts/
- *Purpose:* This scripts makes scenarios for the load generator for the web component.
- *Reads:* conf/skeletonfiles/scenario.skel
- *Outputs:* /home/jorgenru/source/hammerhead/scenario/*.scn

removeimage.sh

- *Location:* measurementscripts/
- *Purpose:* This script removes the images of a given filesize after each individual measurement, to save disk space.
- *Reads:* N/A
- *Outputs:* N/A

C.2.3 Measurementscript_oldnodes

This section contains measurement scripts which had to be slightly altered to fit on to the old node. These will not be listed here as they are only the scripts in Measurementscript see section C.2.2 with small alternations. The scripts can however be found on the website [20].

C.3 DB

This directory contains measurement scripts which had to be altered to measure on the database.

makescenariosDB.sh

- *Location:* DB/
- *Purpose:* This script makes scenarios for the load generator for the DB component.
- *Reads:* conf/skeletonfiles/scenario.skel
- *Outputs:* /home/jorgenru/source/hammerhead/scenario/*.scn

generateload.sh

- *Location:* DB/
- *Purpose:* This har the same function as the other generateload, but this is a bit alternated to work on the database.
- *Reads:* N/A
- *Outputs:* N/A

formatmeasurement.sh

- *Location:* DB/
- *Purpose:* Same purpose as the other formatmeasurement.sh, but with some tiny alternations to work with DB.
- *Reads:*
- *Outputs:* DB/.reserved_nodes.sh

C.4 start-stop scripts

These scripts does exactly what the names implies, they start and stop services, namely Apache and Mysql. These scripts must be run on the nodes where the service is run.

start-apache Starts Apache, this script is a wrap-around on the standard apachectl script.

stop-apache Starts Apache, this script is a wrap-around on the standard apachectl script.

start-mysql Starts mysql, wraparound on the standard mysqld_safe startup script.

stop-mysql Stops mysql. It is a hack on the mysql.server script, it reads the PID and kill mysqld.

Appendix D

RFW script

In this chapter we have included includes three of the scripts from the RFW. The rest of them can be found on the web site [20] in connection to this thesis.

D.1 measure.sh

This is the scripts which control an experiment. The experiment name is set, and the test are defined.

Listing D.1: Measurementscript.sh

```
#!/bin/bash
# Version: 1.4 (10-04-2005)  Author: Jorgen Ruud

# Purpose: This is the main measurement script.  Here the experimentname and the
# test to be performed are set.

# This script is runned without any arguments.

export EXPERIMENTNAME="Test-on-new-webserver"

cd `cat $HOME/.current_run_path`; source runconf.sh   #load config


#Example of a test on image size 300 with 372 sessions.
filesize="300"

for session in "372"
    do
    $MEASUREMENTSCRIPTS/generateload.sh $filesize $session $RUNTIME
    sleep $SLEEPTIME
    $MEASUREMENTSCRIPTS/formatmeasurement.sh $filesize $session $RUNTIME
    echo "I'm done. I will just sleep in 20 seconds"
    sleep 20
done

# Add more tests here
#
```

D.2 formatmeasurement.sh

This script is a wraparound for the *formatmeasurement.pl* script presented later in this chapter. This script has to do with the postprosessing of the results.

Listing D.2: Measurementscript.sh

```
#!/bin/bash
# Version: 1.2 (10-04-2005)  Author: Olav Gisle Tveiten

# Purpose: This is a wraparound script for the perl script with the same name.
# It check that the apropiate logs files has been made,
# Runs the perl script
# Deletes the logfiles.

cd `cat $HOME/.current_run_path`; source runconf.sh

#Test that logfiles has been written.
if test -e "$HAMMERHEAD_LOGDIR/hh.log"
then
echo "hhr.log exists"
else
echo "no hhr.log"
exit
fi

if test "-e $HAMMERHEAD_LOGDIR/time.log"
then
echo "time.log exists"
else
echo "no timelog"
exit
fi

if test "-e $HAMMERHEAD_LOGDIR/vmstat.log"
then
echo "vmstat.log exists"
else
echo "vmstat.log does not exist"
exit
fi

if test "-e $HAMMERHEAD_LOGDIR/iostat.log"
then
echo "iostat.log exists"
else
echo "iotat.log does not exist"
exit
fi

# Removes unessicary information.
cat $HAMMERHEAD_LOGDIR/hh.log |grep ^. |grep -v ^Read |grep -v ^START  |grep -v
    Starting |awk -F ' ' '{print $4 }' > $HAMMERHEAD_LOGDIR/request.log



#Calls the PERL script.
$MEASUREMENTSCRIPTS/formatmeasurement.pl $1 $2 $3


echo "Appending vmstat and iostat log to FS-File logfiles"

echo "IOSTAT LOG" >> $EXPERIMENT_LOGDIR/FS$1-S$2-RT$3
cat $HAMMERHEAD_LOGDIR/iostat.log >> $EXPERIMENT_LOGDIR/FS$1-S$2-RT$3
echo "VMSTAT LOG" >> $EXPERIMENT_LOGDIR/FS$1-S$2-RT$3
cat $HAMMERHEAD_LOGDIR/vmstat.log >> $EXPERIMENT_LOGDIR/FS$1-S$2-RT$3
```

```
echo "Removing LOGS"
rm $HAMMERHEAD_LOGDIR/hh.log
rm $HAMMERHEAD_LOGDIR/hhr.log
rm $HAMMERHEAD_LOGDIR/time.log
rm $HAMMERHEAD_LOGDIR/vmstat.log
rm $HAMMERHEAD_LOGDIR/request.log
rm $HAMMERHEAD_LOGDIR/iostat.log


echo "Removing images, to free disk space"

echo "Logging into $WEBSERVER"
$RLOGIN $WEBSERVER "cd $MEASUREMENTSCRIPTS; ./removeimage.sh $1"      &
sleep 5
```

D.3 formatmeasurement.sh

This is a perl script written to interpret the logs of an experiment, and create a human readable and a parseable output of these file. It also deletes the end effects.

Listing D.3: Measurementscript.sh

```
#!/usr/bin/perl -w
use POSIX;

# Version: 1.8             Author: Jorgen Ruud
#
# Purpose: This script is part of the postprosessing of the result.
# It analyses the logs and outputs, the appropirate results
#
# It will take a series of logfiles, generated by
# - Hammerhead: hh.log and hhr.log
# - Capturecpuusage: time.log, vmstat.log
# - Captureiousage: iostat.log
#
# And output a result file with the name: FS{filesize}-S{sessions}-RT{runtime}
#


# SUBFUNCTIONS

# This is our way of calculating the average operating point.
# Self written because there is some massuing of the results
# Takes an array of operating points as input, and prints out
# the results.

sub calculateaverage {
    my @resultarray;
    my $MAX=0;
    my $MIN=100000;
    my $counter=0;
    my $average=0;
    my $median=0;
    my $sum=0;
    my @list = @_;
    chomp(@list);
    my @sorted = sort @list;
    foreach (@sorted) {

   if ($_ < $MIN){$MIN=$_}
   if ($_ > $MAX){$MAX=$_}
   $counter++;
   $sum=$_+$sum;
    }
    $average=$sum/$counter;
    #Check for odd and even.
    $modulo = $counter % 2;

    if ($modulo == 0) {
        my $n = floor($counter/2);
   $median=0.5*($sorted[$n-1]+$sorted[$n+1]);
   }else{
       my $n = ($counter+1)/2;


       $median=$sorted[$n];
   }

    $resultarray[0]=$average;
    $resultarray[1]=$median;
    $resultarray[2]=$MAX;
    $resultarray[3]=$MIN;
    $resultarray[4]=$counter;

    return @resultarray;

}


#
# This sub function takes two argument.
# ARGS: TIME, INTEGER
# Returns: a new TIME.
#

sub incrementclock {
    my $time = $_[0];
    my $increment = $_[1];
    my @timesplit = split(/:/,$time);
    my $hours = $timesplit[0] + 0;
    my $minutes = $timesplit[1]+ 0;
    my $seconds = $timesplit[2] + 0;
    my $res1 = ($increment + $seconds)/60;


    if ($res1 < 1){
        $finalseconds=$seconds+$increment;
        $finalminutes=$minutes;
        $finalhours=$hours;
     } else {
         $min = floor($res1);
        $finalseconds = $seconds + $increment -($min*60);

# INNNER LOOP TEST FOR MINUTES
   $hour_res = ($min + $minutes)/60;

   if ($hour_res < 1){
         $finalminutes=$min+$minutes;
              $finalhours=$hours;

    } else {

         $hour = floor($hour_res);
         $finalminutes = $minutes + $min -($hour*60);
         $day_res = ($hour + $hours)/24;

# INNER LOOP TEST FOR HOURS
    if ($day_res < 1){
        $finalhours=$hour+$hours;
    } else {
        $day = floor($day_res);
        $finalhours = $hours + $hour -($day*24);
    }
```

```
            }
        }

        # Have to test whetever are below 10, if they are, they are
        # padded with a zero.
        if ($finalhours < 10) {$finalhours="0$finalhours";}
        if ($finalminutes < 10) {$finalminutes="0$finalminutes";}
        if ($finalseconds < 10) {$finalseconds="0$finalseconds";}

return "$finalhours:$finalminutes:$finalseconds";
};

# Subroutine. Trims a string for whitespaces.
sub trimwhitespace($)
{
        my $string = shift;
        $string =~ s/^\s+//;
        $string =~ s/\s+$//;
        return $string;
}


# Global variables.
#

# Found through commandline.
$FILESIZE = trimwhitespace($ARGV[0]);
$SESSIONS = trimwhitespace($ARGV[1]);
$RUNTIME = trimwhitespace($ARGV[2]); # The overall experimenttime

#These are changed from a skel file. And these variables are changed in runconf.sh
$TRANSIENTBURNTIME ="20"; #Wait n seconds to come in steady state.
$ENDTRANSIENT="20"; #Do not look at the n last measurements

$MEASURETIME=$RUNTIME-$ENDTRANSIENT-$TRANSIENTBURNTIME;

#These are changed from a skel file. And these variables are changed in runconf.sh
$LOGDIR = "/home/jorgenru/source/hammerhead";
$OUTDIR= "/home/jorgenru/masterthesis/measurement_logs";


#
# MAIN PROGRAM
#

# This opens the time.log, and finds the starttime of the experiment.

open (TIMELOG,"$LOGDIR/time.log") || die "could not open time.log";
while (<TIMELOG>) {
        @timelog = split(/\+/,$_);
        $STARTDATE = $timelog[1];
       }

close (TIMELOG);

# GLOBAL VARIABLES


$MAXCONNECTIONS="0";

$experimentstart=incrementclock($STARTDATE,($TRANSIENTBURNTIME+1));


# This opens the hh.log, and finds out how many request that has been
# served, during the experiment.

@counter=0;

for ($i=0; $i<$MEASURETIME; $i++){
```

```
open (HHLOG,"$LOGDIR/request.log") || die "could not open request.log";


$counter[$i]=0;

  while ($current = <HHLOG>) {
my $test = trimwhitespace($current);

if ($test eq $experimentstart){
     $counter[$i]++;
     $MAXCONNECTIONS++;

}

}

close(HHLOG);

$experimentstart = incrementclock($experimentstart,1)

}


# Average connection per second (just be justified to two decimals)
$averagereq=$MAXCONNECTIONS/$MEASURETIME;



#
# This section fetches the IDLE things. And put it in a array.


open (VMSTAT,"$LOGDIR/vmstat.log") || die "could not open vmstat.log";

$k=1;
$q=0;

while ($measuredata = <VMSTAT>) {

 #  Were inside the valid data
      if ($k > $TRANSIENTBURNTIME && $k<=($RUNTIME-$ENDTRANSIENT)){
           $operatingpointresult[$q]= trimwhitespace($measuredata);

     #print "\n" ." $operatingpointresult[$q]";
     $q++;
      }
 $k++;
}


close(VMSTAT);

@vmstatresult = calculateaverage(@operatingpointresult);

###  IOSTAT ####

open (IOSTAT,"$LOGDIR/iostat.log") || die "could not open iostat.log";

$k=1;
$q=0;

while ($iodata = <IOSTAT>) {

     my @data = split(/:/,$iodata);

 #  Were inside the valid data

     if ($k > $TRANSIENTBURNTIME && $k<=($RUNTIME-$ENDTRANSIENT)){
    $iostatresult[$q]= trimwhitespace($data[2]);
          $ioread[$q]=trimwhitespace($data[0]);
    $iowrite[$q]=trimwhitespace($data[1]);

          #print "\n" ."$operatingpointresult[$q]";
    $q++;
     }
     $k++;
}


close(VMSTAT);

@iostatoperatingpoint = calculateaverage(@iostatresult);
@iostatread = calculateaverage(@ioread);
@iostatwrite = calculateaverage(@iowrite);






#
# Lets check for failures in the experiment.
#

open (HHRLOG,"$LOGDIR/hhr.log") || die "could not open hhr.log";
while (my $fail=<HHRLOG>) {
     @hhrlog = split(/\:/,$fail);
     my $failures = $hhrlog[0];
     my $numfailures = $hhrlog[1];


     if (trimwhitespace($failures) eq "Failures"){


    if ($numfailures == 0) {
       $fail_output = "0";
       } else {
       $fail_output =  "$numfailures";
       print $fail_output;
        }
     }
}

close (HHRLOG);


#
# Print the results of this script to the file "$logfile"
#

#$dato =`date +%d-%m-%H.%m.%S`;
$logfile = "FS$FILESIZE-S$SESSIONS-RT$RUNTIME";

open(LOG, ">$OUTDIR/$logfile") || die "cannot append: $!";

print LOG "$logfile";
print LOG " \n\n ####################    VMSTAT    #######################\n";
print LOG "\n Filesize:"."$FILESIZE";
print LOG "\n Total number of connections:" ."$MAXCONNECTIONS";
print LOG "\n Average number of connection per sec:" ."$averagereq";
print LOG "\n Median:" ."$vmstatresult[1]";
print LOG "\n Number of samples:" ."$vmstatresult[4]";
print LOG "\n Max number:" ."$vmstatresult[2]";
print LOG "\n MIN number:" ."$vmstatresult[3]";
print LOG "\n Aritmetric mean:" ."$vmstatresult[0]";
print LOG "\n Failures:" ."$fail_output";
print LOG "\n \n";
print LOG "\n###############       IOSTAT DATA #######################\n";
print LOG "\n Iostat utilization [Arimetric mean]:"  . "$iostatoperatingpoint[0]";
print LOG "\n Iostat utilization [Median]:"  . "$iostatoperatingpoint[1]";
print LOG "\n Iostat utilization [MAX]:"  . "$iostatoperatingpoint[2]";
print LOG "\n Iostat utilization [MIN]:"  . "$iostatoperatingpoint[3]";
print LOG "\n ---------------------- \n";
print LOG "\n Iostat read [Arimetric mean]:"   . "$iostatread[0]";
print LOG "\n Iostat read [median]:"   . "$iostatread[1]";
print LOG "\n Iostat read [Max]:"   . "$iostatread[2]";
print LOG "\n Iostat read [Min]:"   . "$iostatread[3]";
print LOG "\n ------------------------------------------\n";
print LOG "\n Iostat write [Arimetric mean]:" . "$iostatwrite[0]";
print LOG "\n Iostat write [Median]:" . "$iostatwrite[1]";
print LOG "\n Iostat write [MAX]:" . "$iostatwrite[2]";
print LOG "\n Iostat write [MIN]:" . "$iostatwrite[3]";
print LOG "\n \n \n \n";
print LOG "------------ LOG FILES -----------------------------\n";
close(LOG);
```

Appendix E

All measurements

This section gives a more detailed overview of the measurements for each file size (image/-text) on all the system we have measured.

For many of the measurements is the regression line found decreasing, which is odd, but not a problem. The reason is that the graph is shown over a small interval in the x- and y-direction and that we have a relative small set of samples. It is important to notice that the scale on the axis that varies a lot between the file sizes.

E.1 Old web server

This section contains the information for the measurements on each file size on the old web server. For each we list Tps, which is transaction per second at utilization 60% found by regression.

File size 10 KB

- **Tps:** ≈ 296
- **Confidence interval:** ± 24.5
- **KB read on disk:** N/A
- **# Samples:** 9

Comment: Since we needed a huge number of connections on small file sizes, the measurements on file size 10 KB was difficult. For that reason have we 9 test runs. Figure E.1 shows the test runs as dots, and the red line shows the regression line. The regression line is decreasing, which is because the graph only shows a small area and the number of samples is relative few.

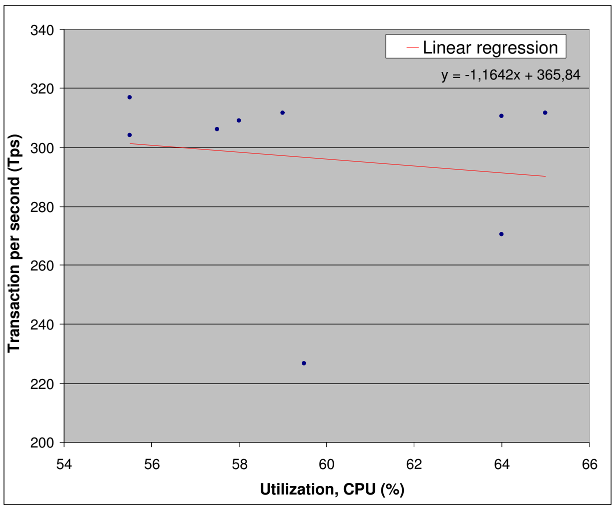


Figure E.1: Old web server, 10 KB

File size 50 KB

- **Tps:** ≈ 182
- **Confidence interval:** ±28.0
- **KB read on disk:** N/A
- **#Samples:** 9

Comment: File size 50 KB is also hard to measure because of the high number of connection, and that is the reason why we only have 9 samples. It is notable that there is a large confidence interval, which one explanation for the relative low Tps compared to file size 100 KB.

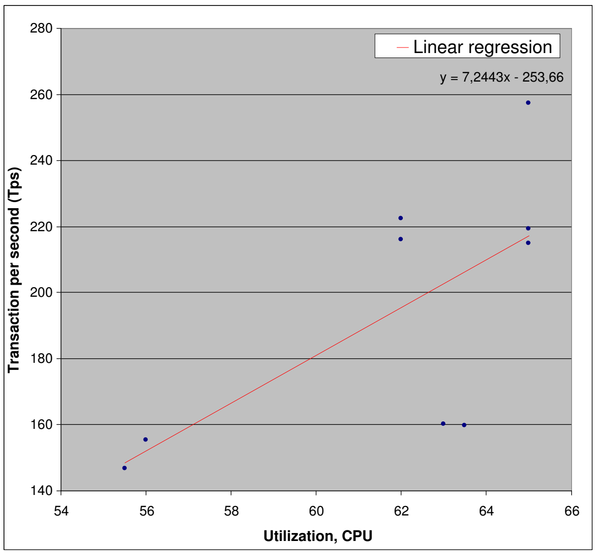


Figure E.2: Old web server, 50 KB

File size 100 KB

- **Tps:** ≈ 214
- **Confidence interval:** ±4.6
- **KB read on disk:** N/A
- **#Samples:** 10

Comment: The samples are relative concentrated around 215 Tps, but we got one sample that is very low. This outlier makes the regression line decreasing. It is notable that the Tps is higher than for 50 KB, and the confidence interval is a lot smaller, which makes this measurements very reliable.

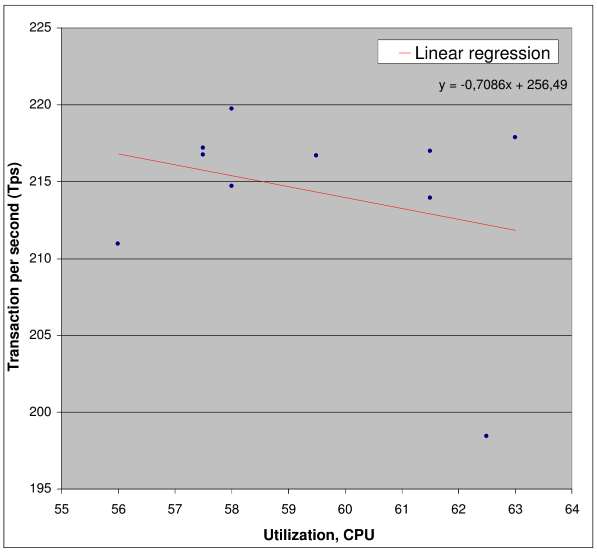


Figure E.3: Old web server, 100 KB

File size 200 KB

- **Tps:** ≈ 129
- **Confidence interval:** ±9.7
- **KB read on disk:** N/A
- **#Samples:** 10

Comment: N/A

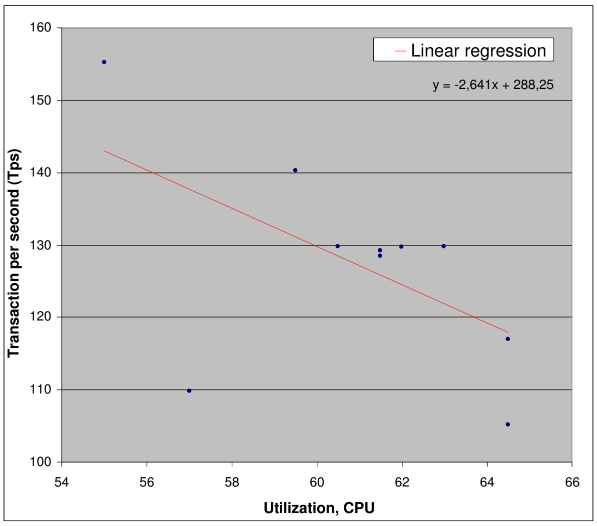


Figure E.4: Old web server, 200 KB

File size 300 KB

- **Tps:** ≈ 108
- **Confidence interval:** ±4.6
- **KB read on disk:** N/A
- **#Samples:** 10

Comment: This samples is not very good as it is very scattered.

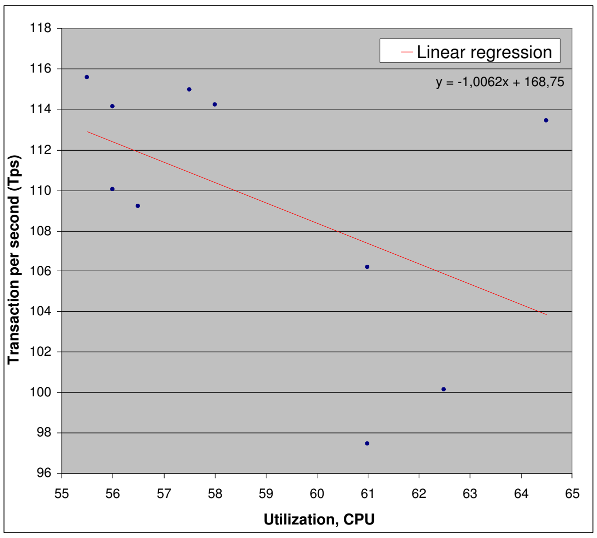


Figure E.5: Old web server, 300 KB

File size 400 KB

- **Tps:** ≈ 78
- **Confidence interval:** ±1.3
- **KB read on disk:** N/A
- **#Samples:** 10

Comment: 9 of the samples for 400 KB is pretty concentrated, with one outlier that increase the confidence interval.

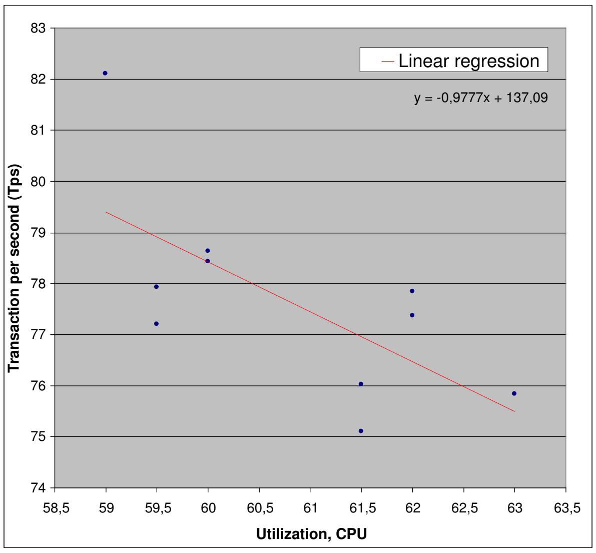


Figure E.6: Old web server, 400 KB

File size 500 KB

- **Tps:** ≈ 62
- **Confidence interval:** ±0.6
- **KB read on disk:** N/A
- **#Samples:** 10

Comment: This samples is relative concentrated, and the confidence interval is only about 1% of the Tps, which is very good.

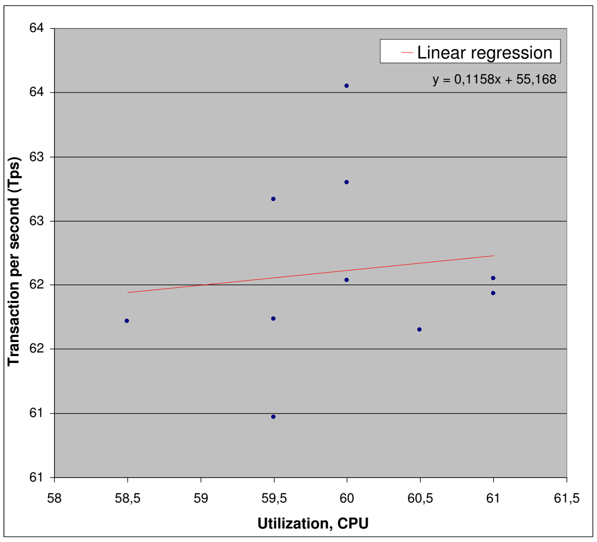


Figure E.7: Old web server, 500 KB

File size 600 KB

- **Tps:** ≈ 56
- **Confidence interval:** ±0.4
- **KB read on disk:** N/A
- **#Samples:** 10

Comment: The measurements on 600 KB are a good collection of samples. The graphs scale makes the measurements looks scattered.

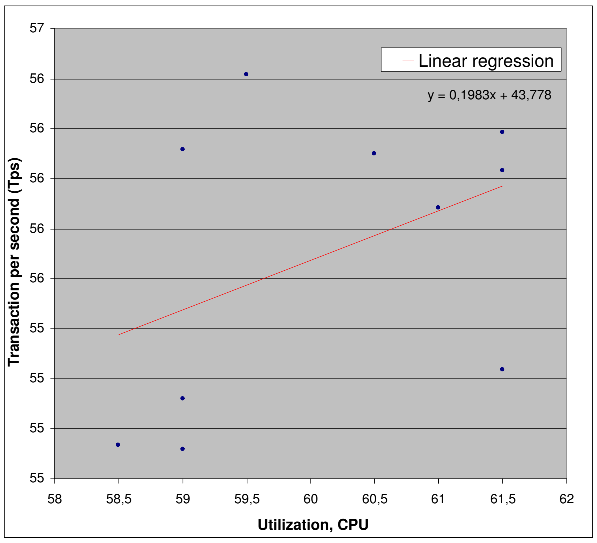


Figure E.8: Old web server, 600 KB

File size 700 KB

- **Tps:** ≈ 49
- **Confidence interval:** ±1.4
- **KB read on disk:** N/A
- **#Samples:** 10

Comment: The samples is relative concentrated, except from one that makes the confidence interval larger.

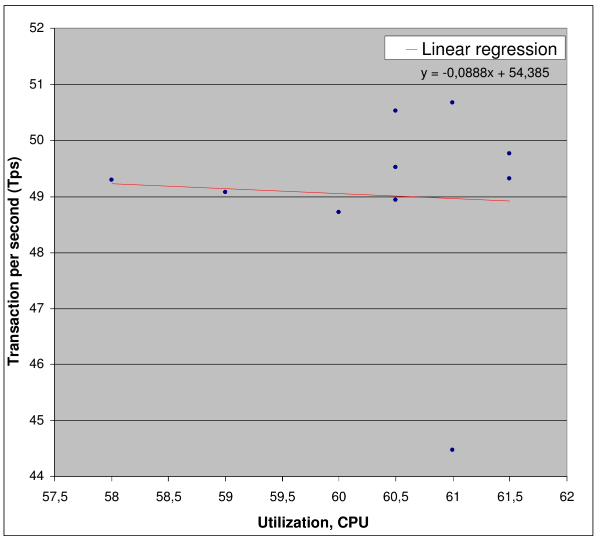


Figure E.9: Old web server, 700 KB

File size 800 KB

- **Tps:** ≈ 41
- **Confidence interval:** ±0.9
- **KB read on disk:** N/A
- **#Samples:** 10

Comment: N/A

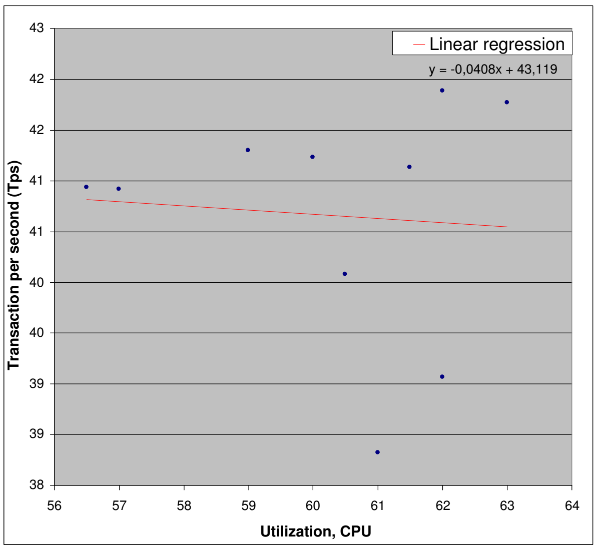


Figure E.10: Old web server, 800 KB

File size 900 KB

- **Tps:** ≈ 35
- **Confidence interval:** ±0.5
- **KB read on disk:** N/A
- **#Samples:** 10

Comment: N/A

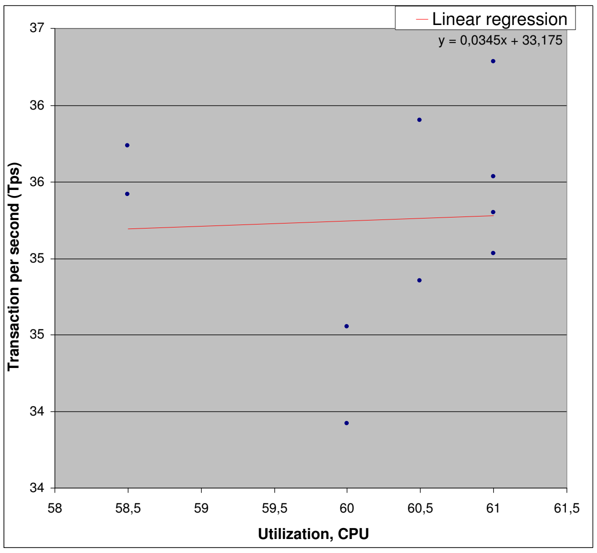


Figure E.11: Old web server, 900 KB

File size 1000 KB

- **Tps:** ≈ 32
- **Confidence interval:** ±0.5
- **KB read on disk:** N/A
- **#Samples:** 10

Comment: N/A

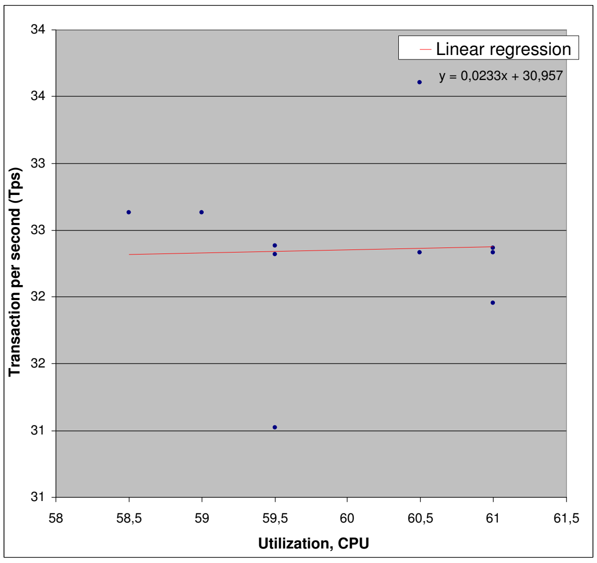


Figure E.12: Old web server, 1000 KB

E.2 New web server

The measurements on the new web server were done by requesting images on the web server.

File size 10 KB

- **Tps:** ≈ 2285
- **Confidence interval:** N/A
- **KB read on disk:** N/A
- **# Samples:** 2

Comment: The measurements on 10 KB were very hard to do. There were needed a large number of connections, and the load generator was not able to generate a stable load to the system. That is the reason for only having 2 samples, which makes the measurements on 10 KB very uncertain. With only 2 samples is there not possible to calculate a confidence interval.

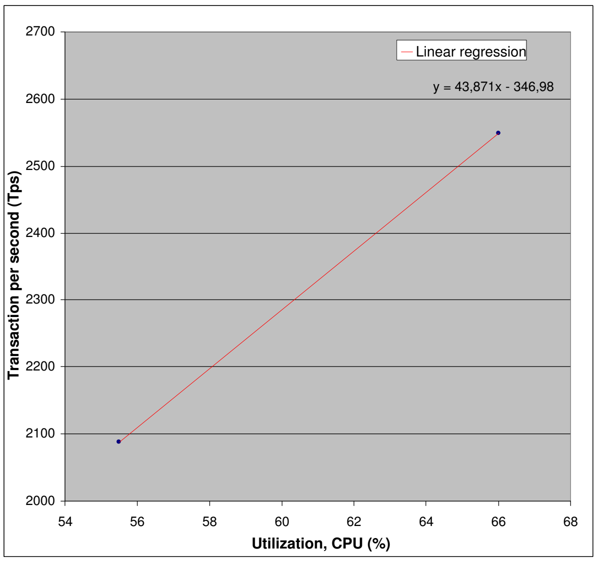


Figure E.13: New web server, 10 KB

File size 50 KB

- **Tps:** ≈ 1600
- **Confidence interval:** ±105.0
- **KB read on disk:** N/A
- **#Samples:** 8

Comment: It was also hard to generate a stable load for 50 KB, and therefore only 8 samples in this measurement. Of the graph the samples looks very scattered, but the confidence interval is lower than 6.6% of the Tps, which makes the measurement relative reliable.

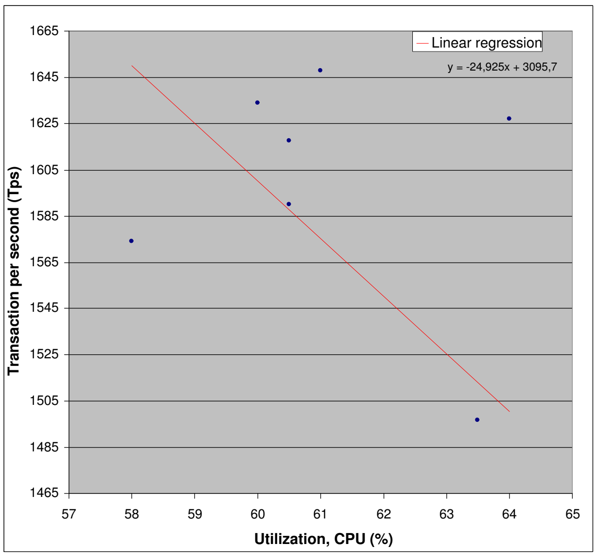


Figure E.14: New web server, 50 KB

File size 100 KB

- **Tps:** ≈ 957
- **Confidence interval:** ±28.0
- **KB read on disk:** N/A
- **#Samples:** 10

Comment: 100 KB also needed a high number of connections, but now was the load generator able to handle it, and generate a stable load to the web server.

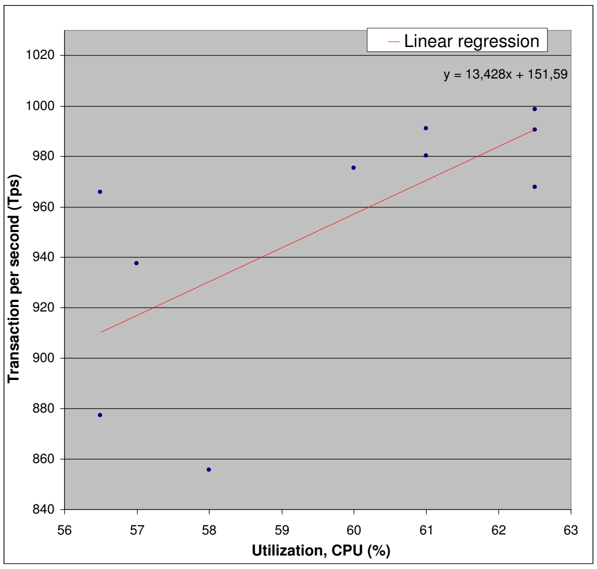


Figure E.15: New web server, 100 KB

File size 200 KB

- **Tps:** ≈ 494
- **Confidence interval:** ±13.3
- **KB read on disk:** N/A
- **#Samples:** 10

Comment: N/A

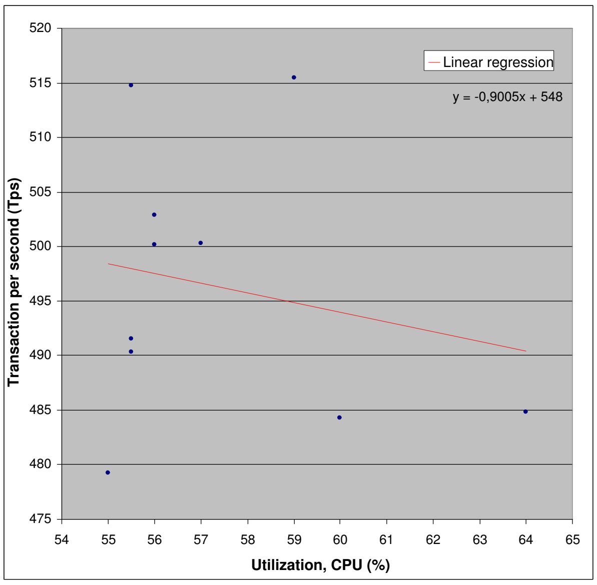


Figure E.16: New web server, 200 KB

File size 300 KB

- **Tps:** ≈ 342
- **Confidence interval:** ±11.5
- **KB read on disk:** N/A
- **#Samples:** 10

Comment: N/A

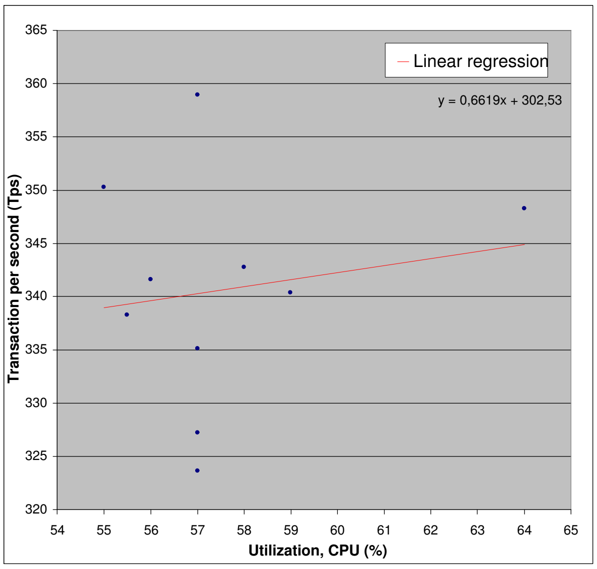


Figure E.17: New web server, 300 KB

File size 400 KB

- **Tps:** ≈ 261
- **Confidence interval:** ±26.2
- **KB read on disk:** N/A
- **#Samples:** 10

Comment: These samples have a large confidence interval, because of 2 samples that is outliers. The rest of the samples are very nice concentrated.

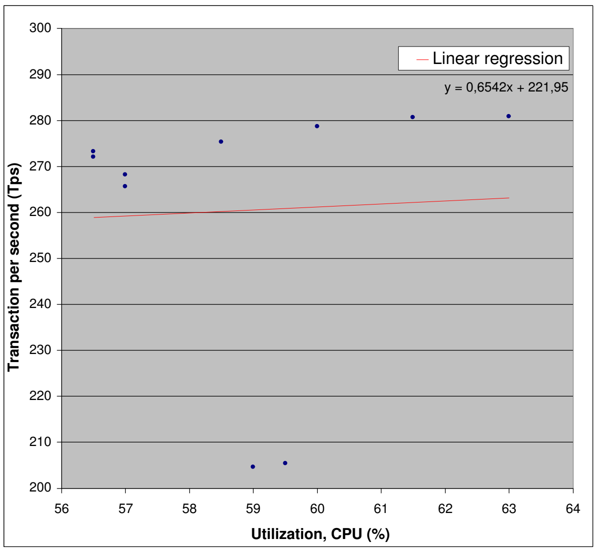


Figure E.18: New web server, 400 KB

File size 500 KB

- **Tps:** ≈ 155
- **Confidence interval:** ±7.4
- **KB read on disk:** $16137KB$
- **#Samples:** 10

Comment: N/A

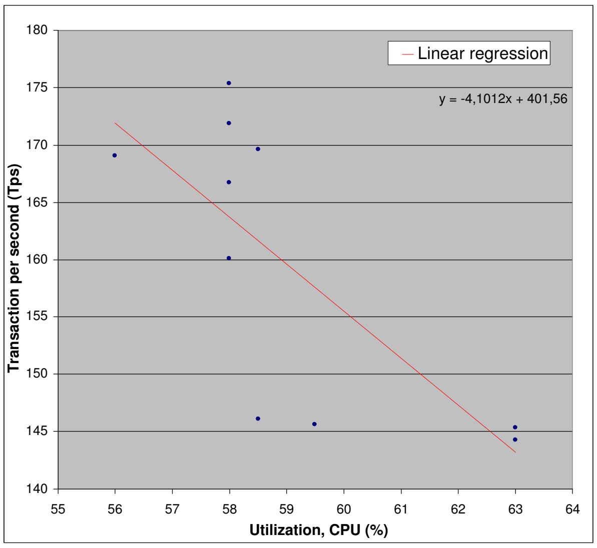


Figure E.19: New web server, 500 KB

File size 600 KB

- **Tps:** ≈ 85
- **Confidence interval:** ±1.0
- **KB read on disk:** $29083KB$
- **#Samples:** 10

Comment: N/A

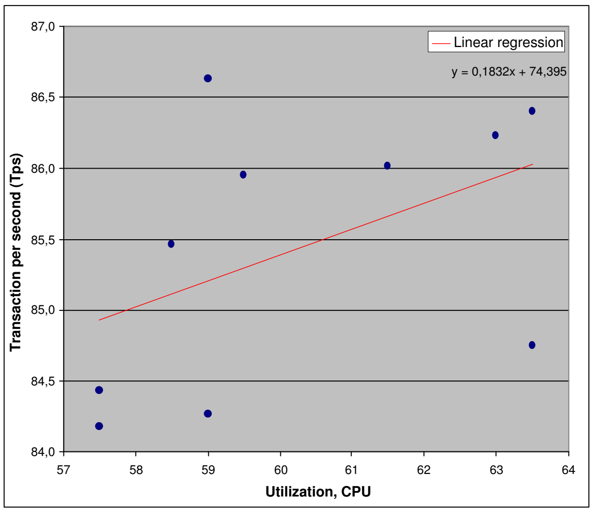


Figure E.20: New web server, 600 KB

File size 700 KB

- **Tps:** ≈ 58
- **Confidence interval:** ±0.5
- **KB read on disk:** $37939 KB$
- **#Samples:** 10

Comment: N/A

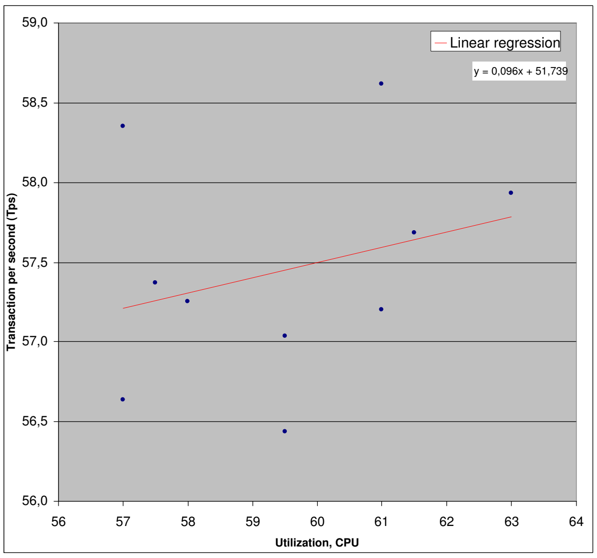


Figure E.21: New web server, 700 KB

File size 800 KB

- **Tps:** ≈ 40
- **Confidence interval:** ±0.3
- **KB read on disk:** $41217 KB$
- **#Samples:** 10

Comment: N/A

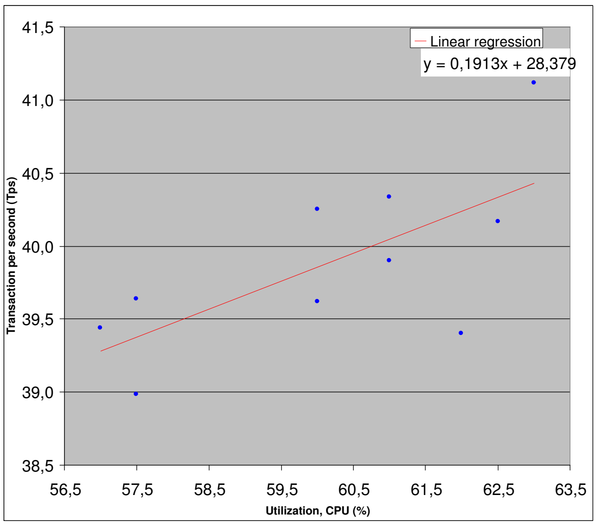


Figure E.22: New web server, 800 KB

File size 900 KB

- **Tps:** ≈ 33
- **Confidence interval:** ±0.3
- **KB read on disk:** $41471KB$
- **#Samples:** 10

Comment: N/A

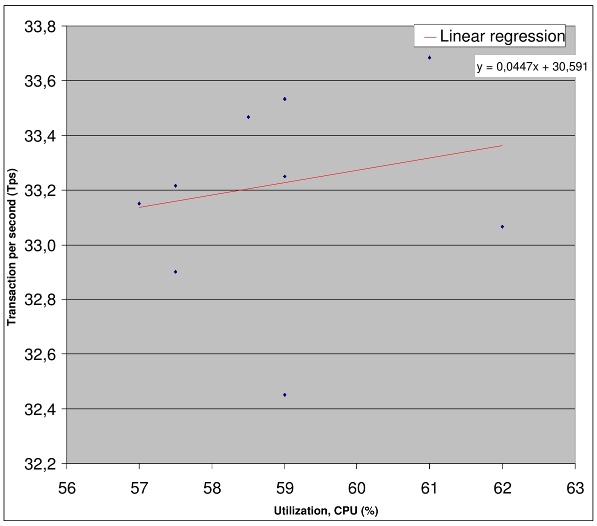


Figure E.23: New web server, 900 KB

File size 1000 KB

- **Tps:** ≈ 28
- **Confidence interval:** ±0.3
- **KB read on disk:** $50163KB$
- **#Samples:** 10

Comment: N/A

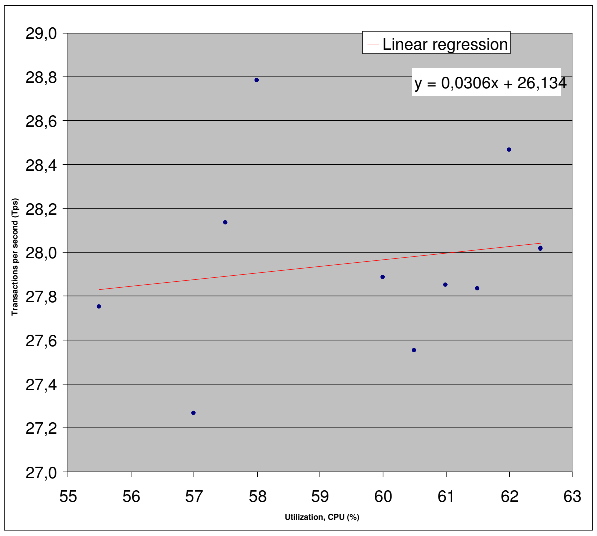


Figure E.24: New web server, 1000 KB

E.3 Database

The database measurements required a very high number of connections, and two load generators are used to generate load to the database. These measurements for the DB are generaly reliable and reproducible. All of the measurements have a confidence interval that is less than 10% of Tps, which makes the measurements reliable.

File size 10 KB

- **Tps:** ≈ 2445
- **Confidence interval:** ±239.2
- **KB read on disk:** N/A
- **# Samples:** 9

Comment: N/A

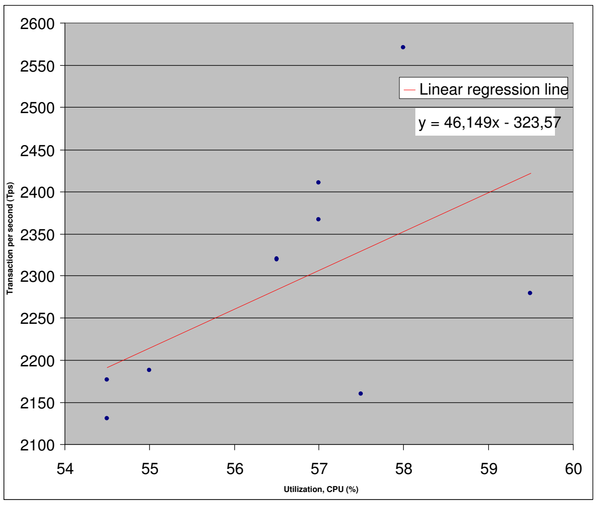


Figure E.25: Database server, 10 KB

File size 50 KB

- **Tps:** ≈ 2379
- **Confidence interval:** ±205.4
- **KB read on disk:** N/A
- **#Samples:** 10

Comment: N/A

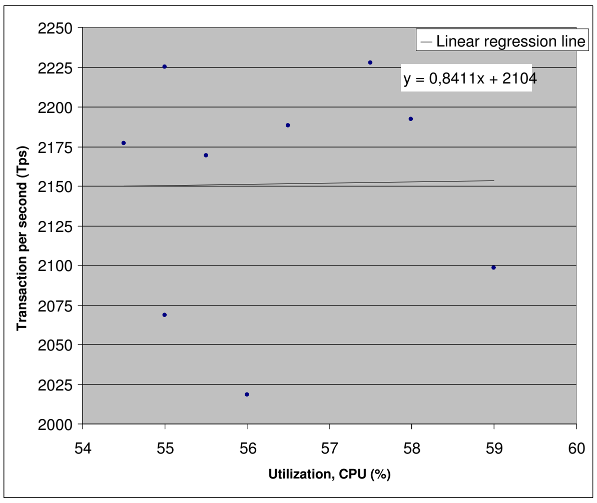


Figure E.26: Database server, 50 KB

File size 100 KB

- **Tps:** ≈ 2236
- **Confidence interval:** ±85.2
- **KB read on disk:** N/A
- **#Samples:** 10

Comment: The samples on this file size are the best on the database server. The confidence interval is small and the concentration is good, except from one outlier.

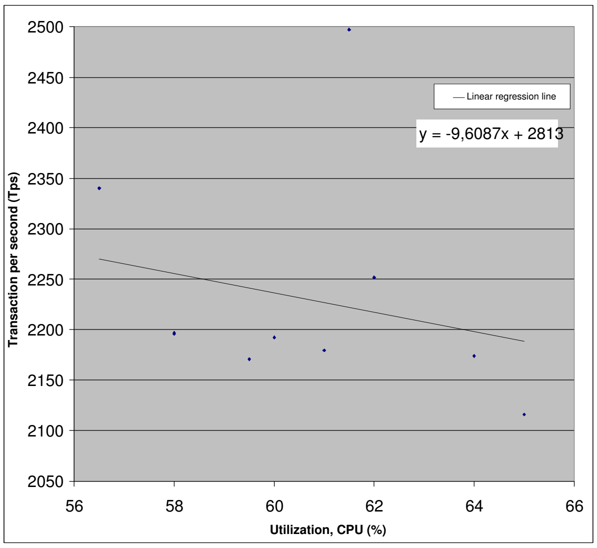


Figure E.27: Database server, 100 KB

File size 200 KB

- **Tps:** ≈ 1853
- **Confidence interval:** ±113.5
- **KB read on disk:** N/A
- **#Samples:** 9

Comment: N/A

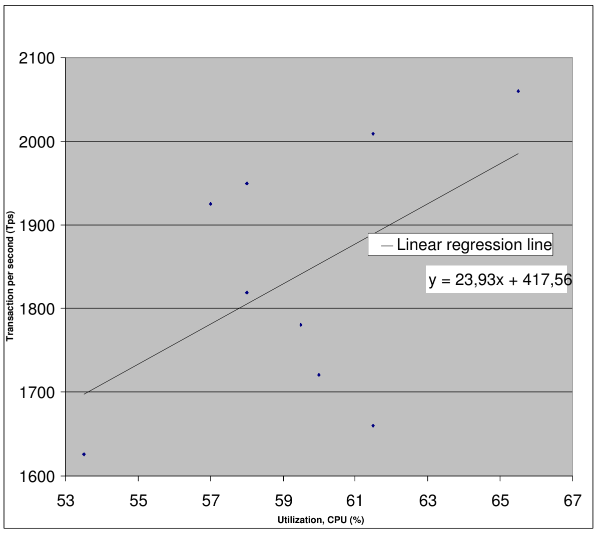


Figure E.28: Database server, 200 KB

File size 300 KB

- **Tps:** ≈ 1493
- **Confidence interval:** ± 112.6
- **KB read on disk:** N/A
- **#Samples:** 8

Comment: N/A

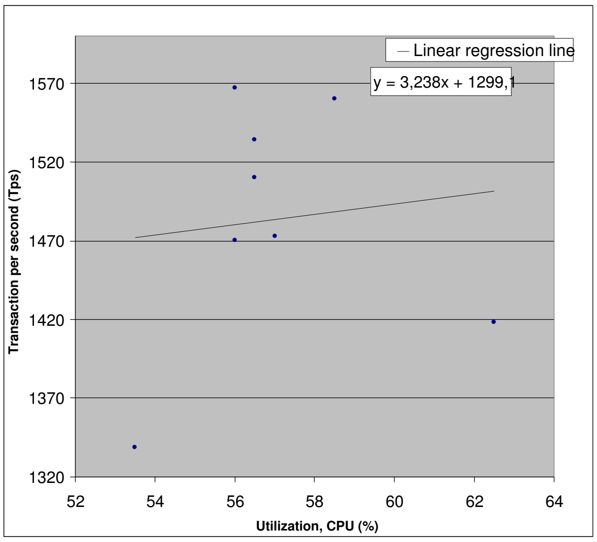


Figure E.29: Database server, 300 KB

Appendix F

Original problem definition

Some minor changes has been done on the original problem definition, and the original one is found below.

> *" In the Ruud & Tveiten project[21] the overall objective was to study the practical and economic feasibility of exploring the scalability of a large-scale read-intensive Internet site for TV2 Interaktiv. Using the SP terminology, a limited number of classes of complexity functions were identified, serving to structure the scalability assessment method. However, the resulting method was not validated with actual measurements.*
>
> *The main objective of this thesis is to continue the work started in [21]: i.e. to explore the practical and economical feasibility of assessing the scalability of a read-intensive large-scale internet site, based on the test plan described in [21]. Since the TV2 Nettavisen infrastructure was hard to measure, the essence of the infrastructure has to be reproduced in a controlled environment. For practical reasons Open Source Software will be used together with the Clustis2 computational cluster of PCs. The similarities and discrepancies between TV2i's infrastructure and the selected test baseline should be described.*
>
> *The static properties of the test baseline will be modeled using the SP method. For scaling exploration the granularity of the SP model should be finer in areas with non-linearity (drill-down). Scaling scenarios will drive the scalability exploration and should be described in detail. In practice, strict and uniform scaling is rare. Some scaling scenarios should therefore consider differential and non-uniform scaling.*
>
> *For each class of measurements a detailed test plan should be both described and validated by measurements at the level of detail which makes it possible for others to revalidate the actual measurements performed. It is still an open question if it is possible to parameterize a complete scalability model, or if only some examples of each complexity function class should be measured. If a complete set of measurements is performed, the resulting static model should be validated in combination with a dynamic model.*
>
> *To measure the components needed in the SP model will in the general case require a Resource Function Workbench [25], a tool which aids in the actual measurement and makes it easier to reuse the complexity functions. To develop*

this is a useful result in its own right. To describe and develop a suitable Resource Function Workbench is in this diploma a secondary objective."