The Uptime Challenge, A Framework Providing Automatic Feedback to Students Learning Value-Driven Operations

Stian Strøm Anderssen
Master’s Thesis Spring 2016
The Uptime Challenge, A Framework Providing Automatic Feedback to Students Learning Value-Driven Operations

Stian Strøm Anderssen

2nd May 2016
Abstract

Today, the world relies on complex infrastructures to host crucial services needed in a society. The need to keep a service running over time can be the key factor that keeps a person from dying. The world's economy relies on complex IT-services to run, and businesses invest largely in new technology that can be used in large-scale infrastructures.

System administrators are responsible for maintaining IT-services. When a service is up and running, the end-users can achieve maximal value from the application. The value indicates the need for the IT-service for an end-user. When an IT-service stops working it could be a costly affair for businesses to get the system up and running.

Educational IT focuses mainly on teaching students methodologies and principles of IT. Students gain knowledge into how to create and build new systems from scratch. The knowledge of evaluating a running infrastructure is something one must learn from experience. This thesis aims to build a framework that can be used by teachers to evaluate student-progression in system administration-related courses. The students will benefit from the solution by getting continuous feedback on their progress in courses. The feedback aims to give students a better insight into evaluating IT-infrastructures, and to map potential vulnerabilities that may cause future problems.

The prototype developed reflects the model designed, and aims to simplify the process of automating assessments between teacher and student. The prototype consists of an application along with a larger infrastructure. The thesis is written as part of a master-program in Network and System Administration, and focuses on the operational aspect of evaluating IT-services.
Acknowledgements

I would like to thank my supervisor Kyrre Mathias Begnum for guidance and help with the thesis. I am very grateful for all help and confidence he has given me. I would also like to thank my friend and fellow student Jostein Lavbråten for good help and support during long programming sessions.
## Contents

1 Introduction .......................... 1
   1.1 Motivation .......................... 1
      1.1.1 Value .......................... 3
   1.2 Problem Statement ............... 3
      1.2.1 Case .......................... 4
      1.2.2 Thesis structure ............... 4

2 Background .......................... 7
   2.1 Computer History .................. 7
   2.2 Deriving value from a computer system 8
      2.2.1 Business value ................. 8
      2.2.2 End-user value ................. 8
   2.3 Automation ........................ 8
   2.4 System administration automation 9
   2.5 Uptime .......................... 9
   2.6 Quality of Service ................ 10
   2.7 Measuring Value ................... 10
   2.8 Education ......................... 11
   2.9 Execution monitoring .............. 11
   2.10 Cost ............................ 12
      2.10.1 Determining IT value with cost 12
      2.10.2 Cost Calculations ............ 12
   2.11 Monitoring ....................... 13
      2.11.1 Nagios ........................ 13
      2.11.2 ELK-stack ..................... 14
   2.12 Security ......................... 14
   2.13 Scalability ....................... 15
      2.13.1 QoS variables ................ 15
   2.14 Technologies .................... 16
      2.14.1 OpenStack ...................... 16
      2.14.2 Puppet ......................... 16
      2.14.3 Containers ..................... 17
      2.14.4 Code ........................ 17
   2.15 Test topologies .................. 18
      2.15.1 Load testing ................... 18
      2.15.2 Stress testing ................ 19
      2.15.3 Continuous testing ............ 19
2.16 Educational projects ........................................ 19
  2.16.1 Value-driven operations through gameification .... 20
  2.16.2 MLN- Manage Large Networks ..................... 20
  2.16.3 Research and teaching ............................... 21
2.17 Projects .................................................. 21
  2.17.1 Reactive based framework tests ...................... 21
  2.17.2 Comparison of different performance Testing tools . 22
  2.17.4 A test audition ......................................... 23
  2.17.5 Mapping distributed application SLA to network QoS parameters ........................................ 23
  2.17.6 Performance evaluation of netfilter .................. 23
2.18 Different tools ............................................ 24
2.19 Taxonomy .................................................. 25

3 Approach ..................................................... 29
  3.1 Introduction ............................................. 29
  3.2 Approaching the problem statement ..................... 29
  3.3 Scalable framework ...................................... 29
    3.3.1 Determining scaling methodology .................. 30
    3.3.2 Education ........................................... 30
  3.4 Value-driven operations .................................. 31
    3.4.1 Teacher value ....................................... 32
  3.5 Stakeholders ............................................. 32
  3.6 Project phases ........................................... 32
  3.7 Determining design-model ................................ 35

4 Design ........................................................ 37
  4.0.1 Introduction .......................................... 37
  4.1 Model .................................................... 37
    4.1.1 Scenario ............................................ 37
  4.2 Phase I - Building the infrastructure .................. 39
    4.2.1 Project infrastructure .............................. 39
    4.2.2 Workflow .......................................... 40
  4.3 Manager ................................................... 42
    4.3.1 Overview ............................................ 42
  4.4 Workers ................................................... 42
  4.5 The interpreter .......................................... 43
    4.5.1 Workflows .......................................... 43
  4.6 The database ............................................. 44
  4.7 Summarizing the infrastructure ......................... 45
    4.7.1 Queues .............................................. 45
    4.7.2 API .................................................. 47
    4.7.3 Test case: Bookface ................................ 47
    4.7.4 Multi-Language support .............................. 47
  4.8 Automation .............................................. 48
    4.8.1 Exception handling on the workers ............... 49
4.8.2 Time-check ........................................ 49
4.8.3 Queue-monitoring ................................. 49
4.8.4 ICMP-check ..................................... 49
4.9 Job-deployment algorithms ......................... 49
4.9.1 Webuse ........................................... 50
4.9.2 Httperf .......................................... 50
4.9.3 Rewarding ...................................... 51
4.9.4 Purser .......................................... 51
4.9.5 Clerk ............................................. 51
4.9.6 Leeshore ......................................... 52
4.10 Workload-profiles ................................. 52
4.10.1 Total base number-sum and noise-sum ...... 56
4.10.2 Strength sum .................................. 57
4.10.3 Capacity ....................................... 57
4.10.4 Sample space .................................. 58
4.10.5 Total strength probability ..................... 59
4.10.6 Total strength probability sample space ... 59
4.11 Determining live-evaluation criteria ............. 60
4.12 Historical evaluation ............................... 61
5 Implementation ........................................ 63
5.1 Problem statement .................................. 63
5.2 Technologies ....................................... 63
5.2.1 Github .......................................... 63
5.2.2 Trello ........................................... 64
5.2.3 Python ......................................... 64
5.2.4 RabbitMQ ....................................... 65
5.2.5 Docker .......................................... 65
5.2.6 CouchDB ........................................ 65
5.3 Infrastructure ....................................... 65
5.3.1 Alto Cloud HiOA ............................... 66
5.3.2 OpenStack Cloud NTNU Gjøvik ............... 66
5.4 Program structure .................................. 66
5.4.1 The managers ................................... 66
5.4.2 The workers ..................................... 70
5.4.3 Database Layout ................................ 76
5.4.4 Querying documents ............................ 76
5.4.5 Temporary views ................................ 77
5.4.6 Interaction ...................................... 78
5.4.7 RabbitMQ ....................................... 79
5.4.8 Workers .......................................... 81
5.4.9 Docker images .................................. 82
5.4.10 Deployment map ................................ 84
5.4.11 Puppet .......................................... 84
5.4.12 Logging ......................................... 85
6 Analysis

6.1 Test case .................................................. 89
  6.1.1 Infrastructure configuration .......................... 90
6.2 Methodologies to evaluate value ....................... 90
6.3 Test deployment data ..................................... 92
6.4 Uptime and Balance relation ............................ 94
6.5 Workload-profiles ........................................ 97
6.6 Httpperf connection rate modelling .................... 98
6.7 Uptime evaluation ......................................... 103
  6.7.1 GroupE .................................................. 104
  6.7.2 Reward and bonus calculation groupE ................ 105
  6.7.3 GroupC .................................................... 106
6.8 Statistics for the course ................................ 109
  6.8.1 Course survey ......................................... 110
  6.8.2 Question one: "Verdispillet" med Bookface motiverte
                  meg til å bruke mer tid på kurset / The "value-game"
                  Bookface motivated me to use more time on the
                  course .................................................. 110
  6.8.3 Question two: "Verdispillet" med Bookface motiverte
                  meg til å bruke mer tid på andre aktiviteter med
                  kurset som ikke var direkte tilknyttet spillet. / The
                  "value-game" Bookface motivated me to use more
                  time on activities related to the the course that was
                  not directly connected to the game .................... 111

7 Discussion .................................................. 113

7.1 Student survey ............................................ 113
  7.1.1 Problem statement versus result .................... 113
7.2 Final result ................................................ 114
  7.2.1 Educational value .................................... 114
  7.2.2 Pedagogical value ..................................... 114
  7.2.3 Scalability .............................................. 115
7.3 Maintenance and configuration .......................... 115
  7.3.1 Managers ............................................... 115
  7.3.2 Workers .................................................. 116
  7.3.3 Interpreter API ....................................... 116
  7.3.4 CouchDB ............................................... 116
  7.3.5 Overall setup .......................................... 117
7.4 Analysis of graphs ........................................ 117
7.5 Implementation ............................................ 117
  7.5.1 Data ...................................................... 118
7.6 Approach and design ...................................... 119
7.7 Impact ....................................................... 119
  7.7.1 New thinking ........................................... 119
7.8 Future work ................................................. 120
  7.8.1 Scalability projections ............................... 120
  7.8.2 Scaling by monitoring RabbitMQ ..................... 124
  7.8.3 Evaluation and valuable feedback .................. 124
7.8.4 Campaign ........................................... 125
7.8.5 Integration against other sites .................. 125
7.8.6 Automation .......................................... 126
7.8.7 Creating new workers and managers .......... 126
7.8.8 REST .................................................. 126
7.8.9 Computer security .................................. 127
7.8.10 Other programming languages ............... 127
8 Conclusion .................................................. 129
Appendices .................................................. 135
A Managers classes ..................................... 137
B Workers ...................................................... 155
C Purser-script ............................................. 169
D Puppet Manifests and Dockerfile .................. 183
E CouchDB database script .............................. 187
F Physical infrastructure Overview .................. 191
List of Figures

3.1 Figure outlining the project periods ........................................ 36
4.1 Manager - worker company structure ...................................... 38
4.2 Model of the infrastructure ................................................... 41
4.3 JSON datatype conversion .................................................... 48
5.1 Code deployment flow figure .................................................. 84
6.1 Webuse deployment graph for a group .................................... 93
6.2 Uptime percentage and Leeshore graph for groupC ................. 95
6.3 Balance development for groupC ............................................ 96
6.4 Webuse deployment simulation for three groups ..................... 98
6.5 Httperf transsine.dat simulation .......................................... 99
6.6 Httperf transsine2.dat simulation ......................................... 100
6.7 Httperf lowtranssine.dat simulation ..................................... 101
6.8 Httperf sysadm.dat simulation ............................................. 102
6.9 GroupE one-percent uptime cost/reward distribution plot ........ 104
6.10 GroupC one-percent uptime cost/reward distribution plot ....... 107
6.11 Survey question one bar plot .............................................. 111
6.12 Survey question two bar plot ............................................. 112
F.1 Full physical infrastructure .................................................. 192
# List of Tables

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Taxonomy report</td>
<td>26</td>
</tr>
<tr>
<td>4.1</td>
<td>Variable descriptions</td>
<td>52</td>
</tr>
<tr>
<td>5.1</td>
<td>Detailed Alto Cloud infrastructure</td>
<td>66</td>
</tr>
<tr>
<td>5.2</td>
<td>Detailed OpenStack Cloud infrastructure</td>
<td>66</td>
</tr>
<tr>
<td>6.1</td>
<td>Reward calculation</td>
<td>105</td>
</tr>
<tr>
<td>6.2</td>
<td>Bonus variable-values</td>
<td>105</td>
</tr>
<tr>
<td>6.3</td>
<td>GroupE one-percent distribution dataset</td>
<td>106</td>
</tr>
<tr>
<td>6.4</td>
<td>Partial OK calculation</td>
<td>108</td>
</tr>
<tr>
<td>6.5</td>
<td>GroupC Balance 1% distribution dataset</td>
<td>109</td>
</tr>
<tr>
<td>6.6</td>
<td>Question one statistics</td>
<td>110</td>
</tr>
<tr>
<td>6.7</td>
<td>Question two statistics</td>
<td>111</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This section will explain the motivation and importance of doing research on Quality of Service-testing and value-driven development. A set of problems will be defined, along with a problem statement that will be the basis for the thesis.

1.1 Motivation

The IT industry is the fastest growing industry in the world. Every day new products are added to the market. 50 percent of all new investments in organizations has been in information technology.[56] The importance of doing research on testing new IT products becomes more and more clear every day. The knowledge of foreseeing potential problems before they happen is crucial in the IT industry. A problem could be a server that stops working due to capacity problems, or a server that stops working because the performance is too high. The challenge is to determine how to prevent problems from happening, and how to fix them once they occur.

Society is dependent on IT services

Today, the world is globalised through the Internet. All over the world, people can access what they want. Business, economy, politics and industry relies on the Internet and its services. In order to complete stock trades, send contracts and complete purchases, an IT service must be running in the background. In medicine, doctors rely on IT-services to protect patient data, and to help them do technically difficult surgeries. A country’s infrastructure is governed by IT services, and people rely on them to work properly. The cost of running a business is about operating a service and businesses base their income on the quality of the service they offer. The goal is always to develop a service the best possible way that can provide maximum value. If the service is working properly, the business can focus on developing the service with new functionality instead of fixing all that is wrong. The goal is to operate the service with a 100% uptime rate.
The job of maintaining a service is crucial for value creation.

Oxford Dictionaries defines values as "The regard that something is held to deserve; the importance, worth, or usefulness of something." [23] The term value is used in a lot of different settings. Value can be a number on a check, human qualities and operational value. The importance of getting valuable feedback is crucial in several ways. A surgeon must know how to use their technical tools, to be able to perform complex surgeries, a salesman must know the value of their product and a system administrator must know every aspect of the service the person operates. Looking from an operations perspective, knowledge, experience and education is crucial in the process of maintaining a service. When a system administrator knows every aspect of the service the person is maintaining, the production value will increase, the cost of running the service will decrease because the operator knows how to run the service with the least amount of energy needed.

The responsible person for operating an IT service is a system administrator.

A system administrator has the responsibility of operating a service. The goal is to operate the service and do the least amount of changes during production and achieve maximal value. The system administrator must build a framework around the service in order to create and achieve maximal value. In order to operate a service properly, a set of tools is needed to be able to detect bugs and things that may go wrong while the service is running. The framework is a set of tools that can help identifying these aspects.

Education and experience is essential for a system administrator to be able to maintain a service. If a service stops working, a system administrator must know how to quickly get the service up and running again. A lot of scenarios that may occur on a service can be prevented by using the try and fail methodology. Some aspects will require experience and education, in order to prevent. The technical requirements will change from service to service, but operations requirements may be similar in different projects. When a service goes down, knowledge and experience can be crucial to be able to get the service up and running again.

Testing the quality of a service will help a system administrator to get insight in how a service can operate under pressure. To perform such tests a test framework is needed. As complexity of services today increases, and management of large-scale services becomes harder, the need for quality of service mapping becomes more necessary. By creating a framework of tests configured specifically for the service, may help a system administrator to get insight in what may go wrong. By developing complex system tests that can be executed over a long period of time, it may be easier to see how a system interacts in a simulated production environment.
Little research

Unit testing and penetration are often performed on applications during development. When developers have finished an application, a responsibility change will happen. The application will be the system administrators responsibility. The system administrators must make sure the solution is properly tested to prevent unforeseen problems. When looking at the operations perspective, it has been done little research on developing tests that can give feedback on the configuration and environment the service is running in. Such tests ignore the functionality of the code, and have the purpose of testing the system in production. Existing research today focuses on performance testing and penetration testing of applications.

1.1.1 Value

Values can be measured in different ways. "IT generates value at two complementary levels: the core asset value (e.g., hardware and software) and the vitally important value-in-use."[3]. Value in IT is measured differently depending on the situation. IT-value for a student can be to be supervised by a teacher with evaluations to ensure progress, while IT-value for a teacher can be a student thriving to work hard on IT-projects, by participating in lectures and discussions. Value in education is often well-formed feedbacks from teacher to students that both understands and benefits from. Normally a teacher may have responsibility for several students at a time, and it may be difficult for every student to achieve the same value from a course due to limited individual teacher-student time. In some cases, it might be good for a student to get an automated feedback on student projects. Automated feedback must be simple, and easy for the student to understand if there should be any value to gain from it. A value-driven model for feedback must then be made, stating how automated feedback should be generated, and different criteria to use as base for the evaluation process.

1.2 Problem Statement

Based on the problems mentioned in the sections above, a problem statement can be defined. To be able to investigate something that has not been looked in to before, a common problem domain must be found, from the problem domain, a problem statement can be defined and create a foundation from where the work in this thesis will outgrow from.

The problem statement defined for this study is:

*Design and develop a scalable framework that provides relevant QoS data, which may give better insight into value-driven operations.*

The goal is to develop a framework that can help users configure and develop tests made specially for their IT-service. The framework should have
the ability to return relevant quality of service data to the tester, which may give an insight into how the service operates.

Several benchmarking tools have already been made today that can measure all kinds of performance on a service. This study will focus more on the value-driven aspect of testing a service. The thesis will look into different mechanisms that can measure performance, and how the results from these tests can give a valuable answer to the end user.

The main challenge with this thesis is to figure out what value in the IT industry actually is, and how it is possible to implement this into something useful. A framework will be developed focusing on these aspects. This thesis will outline the educational value one gets by testing a service. The output from the tests should give a student more knowledge about how a service operates, and suggestions that can help improve the test results. A teacher can use the results as a guide to see how the students improve their skills during a course.

1.2.1 Case

The project aims to be optimised and used in a course at NTNU Gjøvik. A prototype with ability to automate assessment of student projects will be developed. A series of tests will be implemented and used during the course to give students instant continuous feedback on the progress. The finished prototype will be optimised for usage in educational related courses, but should have the ability to be used in other scenarios.

1.2.2 Thesis structure

The thesis will consist of eight chapters outlining different periods of the project. The thesis will follow a standard structure where each chapter describes an aspect of the project. Below a detailed description of the structure is provided.

Introduction

This chapter introduces the problem-area and problem-domain that will be the base for the thesis-work. The introduction presents a problem statement that will be used as reference point when research is conducted.

Background

The background-chapter investigates existing literature on the field, and describes other research-projects relevant for the thesis. The background-chapter will provide the reader with information needed to be able to understand the problem-domain, and the work that lies ahead.

Approach

The approach-chapter will describe the research methodologies that will be used to conduct the research. The approach will describe each project
period into detail, and define areas to investigate. The approach will raise some problems that must be answered in the analysis-chapter.

**Design**

The design-chapter will describe the model that will be used for the implementation-period. The design-chapter will outline workflows for the applications developed and describe technical requirements for the prototype.

**Implementation**

The implementation-chapter will describe the implementation of the prototype and outline technologies used during development. The implementation-chapter will provide the reader with relevant code-examples and describe technical modules into detail.

**Analysis**

The analysis-chapter will investigate the problems presented in the approach. The analysis will consist of graphs and statistics gathered from using the prototype, and will be the main foundation for the discussion-chapter structure.

**Discussion**

The discussion chapter will evaluate the entire project period and discuss different aspects of the product and the prototype. The discussion will evaluate the project and look into if the final result reflects the expected result.

**Conclusion**

The conclusion will summarize the project and shortly conclude the thesis. The conclusion will in short tell the reader about the project outcome and evaluate the process.
Chapter 2

Background

2.1 Computer History

Over the last 50 years, the world has developed from being an industrialized world, where people relied on manual labor for all kinds of tasks, to a digitized world where people can use machines to replace several man-labor years in their corporations. In the early years, computers were expensive to buy and maintain, and were only used for one specific purpose. The first computers were mainly used to solve mathematical equations. In 1944 Harvard University invested a lot of money into building a computer called MARK 1. [43] MARK 1 was designed to do complex calculations, and had the ability to store 72 numbers, each with a 23 decimal digit length. During World War II, the US Navy Bureau of Ships used MARK 1 for complex calculations, and later it was given back to Harvard. MARK 1 is a symbol for one of the first computers, and opened the door to a new digitized world. The students and US Navy both got operational value from MARK 1. They both relied on the computational service MARK 1 provided. In 1959 the PLATO project was launched, a project that aimed to use computers in education. PLATO was launched in the undergraduate system and was the first educational program for computer science.

During the 1960’s and 1970’s, computers started to become smaller, and more available to the public. Wealthy families could in the late 70’s invest in computers that could help them at home. Housewives used computers in the 70’s, to store food recipes in text-files. One of the first text editors ever developed was O26, written in 1967. The editor was known a “line editor”. A line editor had the purpose to look like a typewriter. Every time a line was written. The screen flushed the contents, and an empty line appeared. Computers would replace the necessity for typewriters in the following decades. Computers were still expensive and were not available to everyone. When the Internet came in the early 90’s, the world changed drastically. People could now communicate over long distances, and it was not longer necessary to travel across the world to do business. A new globalised world infrastructure was built with help from the Internet. It was easier to access information and people could easily share their knowledge with the rest of the world. Country infrastructures relied more on techno-
logy, when offering people valuable services that could help solve problems. In medicine, doctors could use the Internet to find information about illnesses, in economy it became easier to do business across the borders through the Internet and bank transactions became digitized. Different disciplines achieved different values from the technology they used. People could now solve a lot of their problems from home using Internet services.

2.2 Deriving value from a computer system

Value in performance analysis is a wide term and has no correct answer. As value is different from person to person, value for one system is different from other systems. As time passes by, there are new aspects to take into consideration. In the early 1970’s, a computer’s value was limited down to a small set of applications running separately. An application could be a text editor that replaced the necessity for a type writer, or a calculator that could help perform complex computations. Today there are an unlimited set of aspects that could help measuring value on a system, scalability, loadbalancing, performance regulation, uptime and bandwidth are some general aspects that are present across different services.

2.2.1 Business value

In business a service value is measured in a cost. The cost could either be negative or positive and is defined as the difference between income and expenses. If the value is negative, it means that the income is smaller than the expenses. This is one of the main reasons why several projects are being cancelled during development.

2.2.2 End-user value

For end-users, the service value can be defined as the entire user-experience when using a service. Uptime, responsivity, availability, simplicity are some of the core values. The end-user is always the most important actor in the lifetime of a service. A performance test may give some indications how the service operates, but the end-user is the one saying the final word. If an end-user finds the value of a service good, it is likely that others will.

2.3 Automation

Earlier, people relied on hard workers to perform tasks that would be automated on a later stage. Automation of manual human processes has turned the world into the industry it is today. Companies have saved millions, on replacing workers with machines. In some fields, a machine could replace several workers at once. In the IT-industry, automation is a crucial part of the infrastructure in a project. When an application is developed, it needs to go through several different processes, in order to validate the code, and to see that the application works as according
to the given specification. Instead of letting this job be done by the developers, this can be done manually by compilators and continuous integration servers. In 1997, Jon Finke wrote a paper for the USENIX LISA conference where he addressed the need for automate site configuration management. He predicted the need for automation of Unix services, focusing on database automation.[26]

**Continuous Integration**

Continuous Integration is a software development practice where members of a team integrate their work frequently.[39] Continuous integration servers have the ability to execute build jobs where an application is built and tested. Jenkins is an example. Jenkins can be configured to simulate builds of sub-modules of an application, or an entire application. When a build is executed, the output can be monitored directly in a web-GUI. If the build was a success, a green light will be added to the build, and the developer will know for sure that their code gave the desired output.

### 2.4 System administration automation

Automation is a crucial field in system administration. Automation lightens the work burden to a system administrator. Automation in system administration can be applied in almost every field; installations and configuration management are two of the most popular fields to automate. Puppet is a configuration management tool that will be described in detail in the technology section. Performance testing is also a tool that can be automated. Instead of letting a system administrator manually perform tests on a system, a configuration management tool can be set to govern the automation of the performance test process. A normal approach is to perform a set of tests every day over a longer period.

### 2.5 Uptime

With the new technology development, it became a crucial need to educate people to operate the technology. Companies invested millions into developing products and services they could earn money on, and the importance of uptime became crucial. Uptime is defined as "The period of time during which an item is in a condition to perform its intended function, whether it is in use or not."[16] In the IT industry, the importance of keeping a machine or service running is crucial. A 100 percent uptime would mean maximum potential income. A computer running a service will not have 100 percent uptime every day. A service is often under constant development, and must be patched regularly to avoid security threats, and to keep the application up to date.

Beattie et al. states in their article ‘Timing the Application of Security Patches for Optimal Uptime.’ that the a system administrators must follow a set of rules when they add new patches to services. Several aspects
must be considered before the update can be installed. Sometimes due
to security faults, it may be necessary to apply a new patch immediately,
and sometimes it must be done after work-hours, to make sure that the
least amount of users will notice the downtime of the service. Beattie et
al. has created a model that can help calculating costs and probabilities
of downtime, when a patch is applied. They conclude the survey by
recommending that people follow either a 10, or 30-day interval, to apply
patches, when a new version is released.

2.6 Quality of Service

Quality of Service is a term used in several different fields. In the IT and
telephony industry, quality of service is often referred to as QoS, and is
defined as "the totality of characteristics of a telecommunications service
that bear on its ability to satisfy stated and implied needs of the user of the
service." [30]. In this study, Quality of Service will define the value one gets
from performance-testing a web-service. QoS results are often presented
through different tests executed on a target application, such tests could
be: error rates, bandwidth use, number of HTTP requests and performance
data. Quality of service results are often used by salesmen when selling
and commercializing a product. The QoS results give insight on how the
product works during pressure, and if it’s worth buying or not.

2.7 Measuring Value

When working with QoS testing on web-based services, there are different
approaches on how value can be measured. Liu et al. suggest in their article
‘QoS computation and policing in dynamic web service selection’ to differ
QoS into deterministic and non-deterministic values. Deterministic QoS
data is QoS that is known before the service is invoked. Examples on this
can be execution price and penalty costs. Non-deterministic values are
uncertain before the service is invoked, this may include execution time,
the number of HTTP requests, and relevant performance data. There are
several ways to determine the QoS on deterministic data. Manufacturers
often use deterministic data to advertise their product, stating costs and
data that may be relevant for an end-user to know. This data is based
on internal tests from the manufacturer. Researchers use the advertised
QoS data in comparison tests where the deterministic data is being retested
externally. The bigger the difference between the advertised and actual
data, the lower QoS score. When doing tests on QoS data, Liu et al.
introduces service provider and requester. A service provider is the server
hosting a service, and the requester is the client connecting to the server.
For non-deterministic data, other measuring mechanisms are needed. Liu
et al. mentions execution monitoring as an example.
2.8 Education

Most schools today offer higher education with focus on programming, and software development. One can only get a system administration master degree four places in the world (including Norway). Therefore, most of QoS testing on systems are performed by persons, who has achieved their knowledge through experience and testing. Performance testing has been an important knowledge the last decade. Performance tuning and performance benchmarking is more and more used to perfecting running systems. A common standard for how to properly test systems has not been set. Due to different test requirements on systems, it is difficult to suggest a standard on how performance testing should be performed. In the school systems, this has not been taught to a high level degree. Performance testing is much more used in the industry, and schools do not focus on this subject. Once a system administrator starts to work in a company, the person might get assigned to QoS benchmarking, due to lack of experience on the matter, it may be hard to determine what’s important to test, and not. Therefore the challenge of producing relevant test data is present.

Some schools use performance benchmarking to help evaluate of students. Students are asked to configure a application with a goal to keep it running the best way possible. Performance tests can then be used against the application to stress-test the system, to determine the quality of the configuration. There are several tests that can be used. Some tests focuses on the uptime aspect, several requests are sent to the application while response time and availability is checked. If the site is up and running with low latency, the QoS result should be satisfactory.

Research on QoS benchmarking is still in an early phase. QoS testing on web-services has one of the main focuses. Benchmarking tools like Httperf is an example of a tool developed to test websites. In a later section, such tools will be described in detail.

2.9 Execution monitoring

Execution monitoring is a mechanism invoked by the service-requester. A client initiates a request to the server to do measurements on program execution time. [38] To perform execution monitoring, different modules must be installed on the server-interfaces participating in the service-job. These modules have the ability to measure execution time of a specific process. The numbers are then returned to the service-requester, and will be presented in the results. Liu, Ngu and Zeng state pros and cons of using execution monitoring. The benefit of using execution monitoring is the relevancy of the data gathered. The execution time will vary from environment to environment, and such tests will provide test-data gathered directly from the tested system. The check-interval is stated as a con, if the tests are performed regularly within a short time-interval, it may influence the actual performance of the web-service. If the time between checks
exceeds a certain limit, the result may present the wrong result. Liu, Ngu and Zeng propose different formulas to compute a good check-interval on a specific system, to solve this problem.

2.10 Cost

Generally cost is defined as “The effort, loss, or sacrifice necessary to achieve or obtain something.”[20]. Cost is often used in economy to describe a price on a service. Cost can also describe performance. A service needs a certain amount of energy and resources to execute successfully. This is often described as cost. Cost and time are often crucial parameters to determine QoS variables; they often act as arguments in mathematical formulas and algorithms in computer science. The cost is a wide term, and often needs a detailed description, when used in different calculations.

2.10.1 Determining IT value with cost

Cost often refers to several different phenomenon’s in the IT-industry. In system administration, cost can refer to both performance usage and the run-time price of executing a program. Begnum Kyrre and Engelstad present a model used in OpenStack to determine a price on different virtual servers. Servers with different technical specifications have different run-time costs. In their research, cost is a factor used in deciding what server to choose, when setting up a service or application. Companies often use business-cost to determine what their virtual machines are worth when running in production. Variables to determine business cost are:

- Run-time-value
- Performance-value
- Maintenance value
- Security
- SLA

When a customer needs to run a virtual machine, all these factors needs to be taken into consideration. Below a practical example is presented.

2.10.2 Cost Calculations

Star Operations and Development INC in Norway operates a business with several clients. They develop and host services for their customers over time. They have developed a car-wash booking system for a customer, and the customer wants to run the service in the company’s virtual environment. The customer signs a two year contract with the company, and a Service Level Agreement (SLA) is defined. The companies decide to give the application internal SLA level normal. SLA level normal means weekly
backup, and 10 hours restoration time.

The company’s virtual environment is running on an OpenStack Platform. All webservers are deployed from images, and chosen from a flavor. A flavor defines a set of technical specifications. Each server has a pre-defined run-time cost. The most common specifications are small ($1), medium($1.5), large($2). The car-wash system requires one front-end server and a database. The booking-application has very low technical requirements, and the company chooses two webservers with flavor low. They calculate the hourly maintenance cost to be $3. The system needs to be exposed for the Internet on port 443 for HTTPS access. Costs for maintaining the firewall security is set to $1 per hour. Based on the given costs the company calculates a total cost of $6 per hour for running the booking system for the car wash.

2.11 Monitoring

Monitoring plays a crucial part in QoS testing. When tests are executed, the output can be monitored live through logs and graphs. When tests are executed manually, the output can be monitored directly from the command line using logs. When using an automated test framework, it is easy to implement tools that let the user monitor several logs live.

2.11.1 Nagios

Nagios is an open source system-monitoring tool that makes it possible to monitor services and performance on servers. A Nagios monitoring server is configured with a set of checks that connects to a client, which performs the checks locally on the monitored servers. Each Nagios check is an executable script. A check contains four states to determine the health of the check:

- Return code 0 - OK
- Return code 1 - WARNING
- Return code 2 - CRITICAL
- Return code 3 - UNKNOWN

Based on the return codes, Nagios can be configured to raise an alarm. When the alarm is raised, an event trigger can be configured to fix the problem.[45]

Nagios differs between active and passive checks. “The major difference between active and passive checks is that active checks are initiated and performed by Nagios, while passive checks are performed by external applications.”[44] Active checks are often executed periodically by the Nagios server, while passive checks are executed when triggered by an application or event.
In performance testing, Nagios can both use active and passive checks to monitor tests. Nicholson states the challenge of performing many performance checks in Nagios at the same time. He has configured Nagios using a plugin called Check_MK that allows him to monitor 7566 services at the same time. [46]. Check_MK can be configured to monitor several components at the same time (transactions, network-traffic, CPU-usage, memory-usage, etc.) without taking too much resources from the clients. The QoS output can therefore be monitored live by using Check_MK and Nagios.

2.11.2 ELK-stack

ELK-stack is an abbreviation for ElasticSearch, Logstash and Kibana. ELK-stack is an application with the ability to serve as a remote logging-server. ElasticSearch serves as a non structured database, storing the logs in JSON format, Logstash receives the messages and writes them into the ElasticSearch database, while Kibana presents the log entries from ElasticSearch in the browser. [32] ELK-stack has the ability to process thousands of log entries every second. Logstash uses regular expressions to parse log-entries, and has the ability to split messages, and add specific parts of messages to database attributes. Such a mechanism makes it possible to logically interpret messages. If the ELK-server is configured to receive network access log from a client, data from the log-entries can be converted to integers, and presented in graphs or tables. When executing performance-tests. ELK-stack can be used to monitor the test output live while the tests are executing. When the tests have finished executing, the log data will be stored in the database, and can be used later for deeper analysis.

2.12 Security

IT-security has been a pioneer in QoS testing the last decade. As technology evolves from day to day, the security aspect needs to follow. When running a distributed web-service exposed for customers, the importance of security becomes crucial. Avritzer states the importance of letting a system use available system performance, when testing a security framework. A service should be secure, but the mechanisms that provide security for the application should not allocate resources needed for the application to run optimally. Security has a high price in the commercialized world. People pay whatever it needs to get their systems secure. Security is a crucial part of an infrastructure, and developers are often legally obligated, to make sure that their software is properly tested for bugs. When choosing a security solution that is not open-source, it is easy for the developer to presume that the distributor properly tests the security aspect. "Some companies have been reluctant to buy open-source software because it is not the privately developed and supported software they are used to buying."[35]. When
people choose open-source security solutions as a security provider, the
code has often been changed and tested several times, to make sure it
provides the desired safety for the application. There is a fear that attackers
will more easily find weaknesses and holes in the system, if the security is
based on an open source solution. Enterprise companies often share their
knowledge with competitors, when it comes to determining known attacks,
and viruses. This information is then kept a secret from the open-source
community. The challenge of keeping an updated database of known
attacks will therefore be difficult, when choosing open-source free-ware.

2.13 Scalability

A lot of services running today are running in a scalable environment.
Scalability is a term used in several different fields, in IT and networking.
Scalability is often referred to as: "The ability of a system to accommodate
an increasing number of elements or objects, to process growing volumes
of work gracefully, and/or to be susceptible to enlargement." [13]. Due to
popularity, it is necessary to create mechanisms that can deploy more vir-
tual servers when the network traffic is high. By using a loadbalancer, a
service can run separately on different webservers, while a loadbalancer
makes sure that the servers are equally loaded with traffic.

When running a scalable environment, performance testing can be a chal-
lenge. Other performance-aspects come into picture. Scalability opens the
possibility of regulating power and energy usage. During the night when
the activity is low, a service could run on one server, but during the day
with high activity, 5 servers can be used. When testing a scalable environ-
ment, new QoS variables comes into the picture, that might influence the
final result. The maximum performance of a service is now unlimited, due
to scalable technical specification. The goal is not only to have a 100 percent
uptime, but to be able to save power, and maximize the energy efficiency
while having 100 percent uptime on the service.

2.13.1 QoS variables

Alrifai, Risse and Nejdl state that the scalability of QoS-based composition
systems is affected by the time complexity of the applied algorithm. They
state a problem in QoS testing today when it comes to determining good
common QoS variables to be tested across different services. The authors
tried to find an algorithm that could help determine what service might
be the best choice, based on QoS variables. In a scalable environment, this
challenge becomes even more present when it comes to determining which
variables to be the most valuable in the system evaluation.
2.14 Technologies

There are several approaches on QoS-testing. Researchers and testers use all kinds of technology to execute performance-tests. In this section, some of them are explained into more detail.

Virtualization

Today, most servers run in a virtual environment. A virtual machine is defined as "a software computer that, like a physical computer, runs an operating system and applications."[57]. A virtual machine consists of configuration files, running in an emulated environment. On top of a physical server, several virtual machines can be deployed. Each virtual machine has virtual hardware, and it can allocate available hardware resources. Virtual machines often share their performance resources internally, which means that a virtual machine can temporarily allocate virtual resources that a different machine was using earlier.

2.14.1 OpenStack

OpenStack is an open source virtualization platform for running virtual machines. OpenStack was launched in 2010 as a joint project with NASA and Rackspace. Today OpenStack has developed into being one of the most open-Source virtualization platforms. OpenStack has very few limits in how it can be configured. Several big companies base their infrastructure on an OpenStack solution, and different companies have contributed to the OpenStack project, which is a complete non-profit organization. Red Hat is an example on such a company, they base their infrastructure mainly on OpenStack and OpenShift solutions.[48]

2.14.2 Puppet

Puppet is a configuration management tool released in 2005 by Luke Kanies.[49] Puppet is used all over the IT industry, and can be used to govern application and services running on different servers. Puppets architecture is based on the master-agent principle. A Puppet Master is configured an set up with a set of Puppet installation manifests that will be applied to the Puppet agents. A Puppet manifests contains instructions on how software should be installed and configured. Puppet will maintain the service continuously over time, and apply new patches when a new version of the software is released. If desired, this function can also be turned of if there is a necessity to run an application at a certain version. Puppet is commonly used to govern Cloud applications in a large scale. [53]. A cloud consists of several modules, and these modules can be governed directly fro Puppet. When a module needs to be reconfigured, this will be done directly on the Puppet master, and pushed out to the modules affected by this change. Puppet operates on the principle of centralized configuration management.
A change only needs to be added once on the Puppet master, before this change can be executed on all agents. This saves a lot of time for the system administrators, and they can focus on other important tasks.[49]

2.14.3 Containers

A container is a technology based on the virtualization principle. A container is still very different from a virtual machine. "Containers virtualize at the operating system level, whereas hypervisor-based solutions virtualize at the hardware level."[42] A container can be executed directly inside a virtual machine, and contains only a given set of application configured in a start-up script. When a developer creates an application, a demo version of the application can be shipped inside a container. The container can then be sent to the customer, and executed directly, without any need to reconfigure the hardware to get the application to work.

When doing performance testing, a container can be used to perform a single operation. Several containers can perform different tasks and then reused for another when it has finished the task.

Docker

"Docker is an open source project that builds on many long familiar technologies from operating systems research: LXC containers, virtualization of the OS, and a hash-based or git-like versioning and differencing system, among others"[12]. Docker is a rapidly growing container technology, and is used in different fields in the IT industry. The research performed on containers is a new way of thinking and is still under testing. Until recently, most of the Docker documentation are written and developed in the open-source community. People develop and push their applications inside a Docker, and upload them directly to Github, where the containers can be downloaded for testing and later use.[24]

2.14.4 Code

When developing performance-tests, the challenge of choosing the right approach can be challenging. One dilemma several developers meet: Is whether to develop own performance-tests specifically developed for one solution, or to use a more general framework that already contains predefined tests. When developing an own test framework, several factors must be taken into consideration. Some programming languages require a lot of performance in order to compile and execute code. These performance requirements may affect the actual performance of the software to be tested.

Benchmarks game

Benchmarks game is a site with ability to performance benchmark different programming languages. A set of similar operations are executed on
a computer, the only difference between the different operations is the programming language used to write the given function. Several mechanisms is measured and taken into consideration:

- Execution time
- Memory usage
- Code length
- CPU usage

A set of different functions can be executed. When comparing Python 3 code to similar Java code, tests showed that Python has up to 47 times longer execution time than Java, but use less than a third of the available memory. Test results also showed that the code length in Python can be a third of what’s it would be in Java.

An experiment of comparing Python and C (using the GNU compiler collection- GCC) was performed. The function tested was a binary-tree algorithm. The results showed that Python used 39 times more memory than C, had 5 times more memory usage, but the code length was 50 percent shorter than the C code. [29]

To determine the programming language to be used will require some thinking. When developing big applications, it could be a smart thing to use Java, due to multi-platform support and big libraries, when developing quick, performance saving algorithms it could be smart to use C, and when developing scripts, Python is clever to use because of the code length. [29]

### 2.15 Test topologies

Testing QoS on services can be done in many ways. This section will cover some of the most used test methodologies.

#### 2.15.1 Load testing

"Load testing lets you measure your site’s QoS performance based on actual customer behavior." [41] A load test is a way to determine the capacity a system has to tolerate load within a short time interval. A load test could be a connection test, where a big amount of TCP connections are generated towards a target system. The goal is to measure the response time. Other load tests could be transaction tests, where sets of transactions are executed over time. An example could be user creations; the tester could be interested measure the transaction time of writing user changes to the database.
2.15.2 Stress testing

Stress testing can be defined as: "To automate, based on the system task architecture, the derivation of test cases that maximize the chances of critical deadline misses within the system." [15]. Stress tests are often similar to load tests. The difference is purpose of the test. Stress tests wants to explore how high activity a system can tolerate before it breaks. Sets of tests are executed against a target system, while checks are performed. The goal is to determine response time, availability, and transaction time. The same stress tests are normally performed continuously over a time-interval. A stress test could look very similar to a denial of service attack. A HTTP connection generator could generate HTTP-connections with an increasing rate against a target-system over time. While the generator is running, a set of checks is performed against the system, checking the uptime. If the site is up, the connection rate will increase. This will go on until the site breaks, or the connections are reset. The end result, will give a clear indication how much traffic the application can tolerate. When developing stress tests, there are many factors that need to be taken into consideration.

Scalability is one of the main challenges when executing stress tests. Several applications have the ability to scale the technical resources available. This would mean better performance. When configuring stress tests for a scalable system, other approaches must be made. Theoretically a system can scale as long as there is physical hardware resources available, when performing stress tests in a virtual environment, tests needs to foresee this.

2.15.3 Continuous testing

When configuring performance tests, the most normal approach is to let the same test run continuously over time. This will produce data, which can be used to analyse behaviour patterns over time. Continuous performance tests can also contain different tests that are executed repeatedly. One might be interested to test how the system can behave, while under heavy traffic. Two tests can be executed paralleled, to check how different services interact and depend on each other.

2.16 Educational projects

On higher education levels, education in system administration is still in an early phase. Earlier a teacher would face performance challenges when it came to granting students access to servers to do assignments on, physical hardware was a necessity to achieve the required performance. With the arrival of virtual machines in the early 2000, several new doors were opened in the system administration field. [11] Virtual machines opened the possibility for students to configure their own networks, create their own scripts, and deploy designated virtual machines for different tasks. New technology made it possible to administer several
virtual machines inside a physical server, with help from different virtual platforms. VMWare, Xen, OpenStack are examples on such platforms.

2.16.1 Value-driven operations through gameification

Along with this thesis, Kyrre Begnum has wrote a paper about the educational aspects of teaching system administration. He talks about different aspects of educating students in system administration. He introduces a game-based approach where students participate in a challenge. The students are divided into groups of two. Each group will represent a fictive company where their task is to host and operate a web-site. The application developed in this thesis will act as the activity generator for the student-sites. All groups must download the same web-site software and make sure it is properly set up. The students are not allowed to make modification to the site-code, in order to make it as close to the corporate world as possible. In a company, system administrators are not allowed to modify the code in an application. They are responsible to get the software up and running using the tools developed. During the contest, the students will constantly be awarded in fake money if their site is working properly. The reward will be added to the group’s balance to indicate how the group is performing in the contest.

The paper focuses on the teacher aspect of the project period. The paper aims to investigate the need for such a system, and the process of learning value-driven operations. [9] The paper will be published when the project-period for this is over.

2.16.2 MLN- Manage Large Networks

MLN (Manage Large Networks) is a virtual machine administration tool designed to build and run virtual machine networks based on Xen, VMware Server and User-Mode Linux. The original project was named My Linux Network. The goal with MLN is to make is easy to administer deployment of virtual machines on different platforms. MLN uses scripts as a configuration tool to deploy machines. A MLN script can have a global configuration that will apply to all the virtual machines in the script, and a specific configuration per machine. MLN has later been configured to work against the OpenStack platform, which is currently used as a virtual environment platform at Oslo and Akershus University College Of Applied Sciences.

In education, MLN has been used as a tool to simplify the need to learn complex methodologies to deploy virtual machines. MLN use projects to define a logical group of several virtual machines that can be deployed simultaneously, each project can be started and stopped directly from the command line. Students at HiOA use MLN as part of their education program. MLN supports plugins, and can easily be configured to work against different virtual platforms. [10]
2.16.3 Research and teaching

Virtualization in education is becoming more and more important. Virtualization open up the possibility to allocate physical resources to several virtual machines. As part of their education program, students learn how to configure and set up virtual machines using different mechanisms. Research on virtual machines has come very far when it comes to configuration and deploy-ability with focus on development projects. Students configure virtual machines to run servers where code can be compiled and executed safely. When working in virtual labs, students stands more freely to test different commands without risking to destroy physical components. Inside the virtual machine, the student has root privileges, and is allowed to install and configure whatever the students wants to.

QoS in education has little available research. Teachers focuses on configuration and installation, but not on performance optimization and evaluation. If students wants to learn about QoS. They must look it up themselves, and do their own research.

2.17 Projects

This section will give examples on existing QoS projects, where value-driven operations is in focus. IT security is one of few fields where research on this matter has been performed.

2.17.1 Reactive based framework tests

Researchers Yogita Rasal and Sangeeta Nagpure participated in 2015 in a research project where the goal was to analyse and foresee QoS based on HTTP access logs. The researchers executed tests on a construction web site. Every time a user accessed the site, The following data was gathered:

- User ID
- Request type
- Request time
- Execution time

The data was sent to an access log where they used a log analyzing tool called Web Log Explorer for analysis. The goal with the data gathering was to be able to see patterns in how a user interacted on the website. Based on the given data they built an automated test case generation with a goal to figure out why a site could be slow to access over a network. The results "can help to generate high quality test plans and reduce the cost by minimizing manual work." [51]

The research project is not yet finished, the next phase will be to implement a test case model with the data they gather, using complex evolutionary algorithms.[51]
2.17.2 Comparison of different performance Testing tools

Research scholar Ravi Kumar and professor A.J Singh published in January 2015 a report where they compare QoS tools. They executed the same tests on the same computer on the same site. One of the goals with the tests was to determine if free-ware versions of performance testing tools can give the same quality feedback as the tools one have to buy a license for to use. The web service they performed the test against was a conversion tool that converted an input of Celsius degrees to Fahrenheit. During the test, the following performance benchmark tools were used:

- Apache JMeter- Apache License based
- Soapui pro - GNU/LGPL 2.1 licence
- Wcfstorm - BSD license
- Wizdl GPLv2 license
- SOA cleaner - free-ware
- SOAPSonar Personal - free-ware

The tests were performed in the exact same environment, using the same input data. The first test was to convert 100 degrees Celsius to Fahrenheit, the second test was to test the response time on invalid input (Alphanumeric letters, instead of integers). The tests showed that SOAPSonar Personal has the best response time on both tests with 391.68 ms on the first test and 394.5 ms on the second. [34]

2.17.3 Web Applications Testing-An Analysis of QOS factors in Testing Methodologies and Strategies

Research scholar S. Kannan, and Head of department at G.K.M College of engineering, Vandular, Cheenai India has written a paper where they analyse different strategies in web application testing. They list site availability and response time to be crucial QoS factors, when testing a website. The authors mentions bulk testing as a methodology. "Bulk testing lets you ad measurement your site QoS achievement based on absolute chump behavior."[33]. The goal is to simulate the browser behavior, a series of continuous requests is sent to the website, the sender waits for the site to send an acknowledgment of the request, and once received a new requests is sent. After the tests are executed, the results can be used in further analysis to analyse patterns and costs in the use of the application. This test mechanism is currently used to analyse how a site will tolerate network ads. The tests concludes with the difficulty of using bulk testing on web sites in the future, due to many third party products that are used on sites. CDns, ad-networks, bounded administration and duplication, alive media features and wireless access. [33]
2.17.4 A test audition

Kyrre Begnum, Gaute Borgenholt, and Paal Engelstad wrote a paper in 2013 where they simulated a test case as an Audition. "The sysadmin resembles the director, who manages the manuscript, but needs to find the right actor for each role. [14]. The workflow was divided into several parts, as according to a screen play workflow:

- The manuscript- A model containing the necessary actors (Database, servers, monitoring hosts)
- The casting call- Identify key hardware which would follow the role [8]
- The audition- Optimizing the correct software and hardware, destined to be used in the test
- Cast selection and contract negotiations- A choice will be made, based on the cheapest server that performed within the desired technical requirements.

The tool was optimized for Amazon Cloud services, but can be applied to other virtual environments like OpenStack, VMWare, Xen or KVM. During the test, Puppet was used to configure the desired web servers with the required software and configuration. The tests concluded with saying that "We do not see Audition and scaling as mutually exclusive. In fact, just because one can scale, does not mean one should not pay attention to the performance of web servers and to minimize the cost for the current constellation of virtual machines."[8]

2.17.5 Mapping distributed application SLA to network QoS parameters

Liu, Ray and Jha performed in 2003 an analysis on QoS parameters in network traffic based on SLA. They wanted to map the network traffic performance on the application level. They took the entire network into consideration, and looked at routers throughput/utilization, link bandwidth and response time to determine good formulas to calculate QoS results for the network. The test did not take packet loss and availability into consideration. They focused on mapping the response time on single component level an application level, to get a complete picture on response time.[37]. By using statistical methodologies, the researchers presented different models that could be followed for mapping networking into SLA.

2.17.6 Performance evaluation of netfilter

Niemann, Pfingst and Göbel are a group of researchers that have performed an analysis on the netfilter functionality for GNU/Linux kernels. They wanted to evaluate and analyse the performance of running a netfilter firewall. The hardware and apparatus used in the test followed the
guidelines in RFC 3511, that described methodologies for performance benchmarking of firewalls.

The test composed of a three series test where they wanted to check the following.

- Plain forwarding - To determine the forwarding capabilities of netfilter

- Simple up- and download rules - Check the IP addresses and the protocols used in the test.

- Simple up- and download rules as well as QoS marks - Does the same of the previous, but also inserts one netfilter rule per client thread, with the responsibility of tagging incoming and outgoing packets with a QoS mark

The experiment concluded that "The throughput loss does not depend on those parameters. The throughput loss is also quite insignificant and rises roughly linear with the number of rules." The test used iptables to write rules, but states that the successor nftables would increase the performance gain drastically.

2.18 Different tools

There are several existing solutions that makes it possible to quality-check IT services. Some tools are licensed based, and users have to pay to use the software, some of the tools are open-source free-ware. Below are a few of the most famous mentioned.

Httperf

According to Sung-Jae Jung, Yu-Mi Bae and Wooyoung Soh et al., Httperf is a tool for measuring web server performance. Httperf is a framework written and developed by David Mosberger, and was released in 1998. Httperf makes it possible to perform continuous tests on a website. To initiate a test, a rate must be specified. The rate indicates how many requests that should be initiated per second to the target webserver. A rate of 10 requests per second can be more than enough to verify how the server operates under pressure. A total number of connections must also be set for Httperf to work. Httperf will execute requests until it reaches the value of num-conn. (Number of connections) Httperf operates by sending HTTP GET requests to the target server. If the server doesn’t reply to request, a number of retries can be defined. A default value is 3.

Httperf is a used by IT operators all over the world when someone wants to test the web service. Httperf use a continuous test methodology, the same tests are performed a given number of times. The statistics from the continuous test can be used to determine how the server respond to HTTP traffic, and how many connections it can tolerate.
2.19 Taxonomy

This section will contain a taxonomy overview covering some of the projects mentioned in this chapter. A taxonomy report is a table summarizing and categorising different projects. The taxonomy report is meant to be used as a help for the reader to get an overview of each paper presented. The taxonomy report will contain the following columns:

Level - The level of research performed in the article. Either bachelor, master or Ph.D. level

Subject - The main subjects that the paper covers, and what areas this subjects has been applied to (Security, Monitoring, SLA, Virtualization)

Platform - The virtual platform these projects has been tested on. A virtual platform could be an operating system, or a virtual platform software (VMWare, OpenStack, Zen)

Deployment method - How the project is deployed.

Learning Outcome - A reference list of key elements that is relevant for the research.
<table>
<thead>
<tr>
<th>Title</th>
<th>Level</th>
<th>Subject</th>
<th>Platform</th>
<th>Deployment method</th>
<th>Learning outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping distributed application SLA to network QoS parameters</td>
<td>Master</td>
<td>SLA, SLM, QoS</td>
<td>Unix, Cobra</td>
<td>Mathematical approach</td>
<td>How to map network performance into SLA.</td>
</tr>
<tr>
<td>Performance evaluation of netfilter</td>
<td>Master</td>
<td>Security, Linux kernel</td>
<td>Linux</td>
<td>Statistical Approach</td>
<td>Performance testing the Linux netfilter functionality</td>
</tr>
<tr>
<td>A test audition</td>
<td>Master</td>
<td>Hardware selection, Cost</td>
<td>Amazon Cloud Services, Linux</td>
<td>Cost calculative</td>
<td>Defining a mechanism to determine the right virtual environment to run a service on, based on an audition.</td>
</tr>
</tbody>
</table>
Apache HTTP server benchmarking tool

The benchmarking tool comes with all Apache installations, the tool has the ability to test the maximum number of HTTP requests a server running with Apache can tolerate. The tool sends continuous requests and bases the result on how the server interacts to the requests. The has a lot of the same functionality as Httpperf, but lacks the functionality of defining a connection rate per second. The rate is automatically increased in order to check how many requests the server can handle.[4]

SOAPSonar Personal

SOAPSonar is a software diagnostics tool, and is used in SOAP, XML and REST based Web-services. The software is owned and maintained by Cross-check Networks. The software is open-source and can be used to perform advanced QoS-tests on systems. The software supports graphical testing, and can be used to create advanced test logs that can be used for stats and graphs.
Chapter 3

Approach

3.1 Introduction

This chapter will go into detail on the motives for the project, and discuss different ways to approach the problem statement described in the introduction chapter. The chapter aims to discuss different research-methodologies and how they can be used in the project.

Research today relies on support and funding from organizations. Research is expensive, and requires big investments from investors willing to support researchers in approaching a problem. In order to get investors to invest in the research-project, an approach must be made. An approach is defined as "A way of dealing with a situation or problem." [18]. Along with a presentation of the problem, an approach can present a way to investigate and conduct research on the matter. The approach is often written and presented long before the project begins.

3.2 Approaching the problem statement

The approach for this project aims to explore the problem statement presented in the introduction chapter: "Design and develop a scalable framework that provides relevant QoS data which may give better insight into value-driven operations."

To be able to analyse and work well with the problem statement, a list of adjectives is drawn from the statement. Each adjective describes a sub-problem that must be answered to be able to approach the main problem. The sections below will go into detail on each of the adjectives, and describe how to approach the problem-areas they define.

3.3 Scalable framework

A scalable solution means a solution with ability to be changed in size or scale. [22]. Automation and scaling are key-terms in focus in the IT-industry today. To be able to create solutions with ability to scale up,
and downwards based on performance-need is a must today. Companies can reduce their costs significantly by running scalable infrastructures with ability to scale down performance based on activity. There are several approaches to follow when implementing scalability in an infrastructure. The challenge one faces, is determining the bottlenecks that may cause the scaling to fail. Determining what QoS parameters to base the scaling on is critical. Several metrics can be chosen. When running a webserver, latency and incoming connections can be used as metric to determine when to scale. When running a database-service: Performance, load and memory can be used as QoS criteria for scaling.

3.3.1 Determining scaling methodology

Scaling is often referred to as horizontal and vertical. Horizontal scaling means adding/removing instances based on performance-needs. [55] When scaling vertically, more virtual resources are added to existing instances. A server running in a virtual environment could be configured with an extra CPU when running processes that requires high load.

When using containers, scaling can more easily be achieved by deploying/removing instances instantly. A container has the ability to spawn in seconds. More details about containers can be found in the background-chapter.

Relevant Quality of Service data

Determining relevant QoS data in an evaluation process is critical for giving a valuable feedback to the developers. During project development, education is the main focus. The framework that needs to be developed must have the ability to be used in courses and to help the teacher evaluate the work students deliver. QoS data for a student-project varies from project to project. QoS check-parameters must be defined manually by a teacher to make it possible for a system to perform automatic evaluation of student projects. (More details on QoS can be found in the background chapter).

3.3.2 Education

Educational assessment projects aims to evaluate a students progression in a course. Assignments aim to give a group a chance to exercise their skills by completing a set of challenges. In IT-related educational projects, QoS focuses on the technical complexity in assignments students deliver. Technical complexity can be measured using a set of defined QoS parameters. A teacher should also have the option to manually override automatic evaluation data if an error has occurred. Students should also be able to access evaluation data for their project in order to see progression over time and get feedback.
**Student responsibility**

Students are responsible for their own learning. This means each student must make sure they are motivated and prepared to learn new knowledge continuously. Students are obligated to read hundreds of pages per semester to get good feedback in courses. System administration studies are a little different from other educations. System administration requires practical training along with theoretical knowledge to gain personal skills in the field. By using automated evaluation systems in system administration related courses, the students are encouraged to try out technologies themselves. Students are granted access to virtual environments where they can work as their own administrators. With the administrator-privilege comes a responsibility to make sure that services work. The student must take responsibility for maintaining their virtual environment, and they must create mechanisms to implement functionality to monitor their services to get alerted if something unexpected happens. Evaluation-systems can help the students evaluate technical aspects of their solutions. Teachers can then spend more of their time helping the students achieve their goals, and to supervise projects.

### 3.4 Value-driven operations

Value is a wide term, and could be measured differently for different entities in a project. (More information on value can be found in the background chapter) Internal value for an IT-company could be automatic feedback on how a service is running. A value can both be negative and positive. A negative value could be a team-member contributing with low-skilled work to the team. The total teamwork value will decrease because the team-member doesn’t contribute as much as expected. A positive value could be a team delivering a product that works better than expected. When the expected value is achieved, all positive aspects after this point will create extra value that may strengthen a customer-relationship.

**Value-driven educational operations**

The research project is conducted at Oslo University College of Applied Sciences and the University of Oslo. The application developed during the development phase, shall be used as evaluation tool in a course at NTNU Gjøvik. To achieve value-driven evaluation of the progression of group-projects, a complex evaluation model must be defined to outline what needs to be evaluated.

The importance of letting the students keeping track of the own progression should be in focus. A student-group should be able to access live-data giving a form for feedback that the students can use directly to improve their solutions.
3.4.1 Teacher value

An automated evaluation process should help a teacher giving immediate feedback to students. A teacher should be able to define a set of evaluation tests, and set specific evaluation criteria. The feedback should improve the productivity for a teacher in course, allowing the teacher focus on other important aspects of a course.

3.5 Stakeholders

Before a design can be implemented, one needs to define the stakeholders in the project. A stakeholder can be defined "those groups without whose support the organization would cease to exist." [28] A stakeholder could be a manager planning the project, or a project-tester. The stakeholder will impact the projects progression, and is responsible for making the project successful. During the project period, a set of stakeholders will be defined that will play a big part of the project-implementation. The stakeholders will be key-persons for ensuring progression. Important roles that must be filled during the project development are the following:

- Project tester - Students
- Project lead - Teacher
- Project developer

The teacher must participate in the development process to supervise, and to make sure that the prototype is optimized for the course. The students will act as test cases, and may contribute with valuable feedback on improvements and functionality.

3.6 Project phases

The project period will consist of several phases. Each phase will be outline the outcome from the previous phase, and follow a strict plan. To ensure progression, a detailed plan will be made. This section will contain information about each period, and what importance it plays for the project. Each project period will be different sprints.

Sprint 1 - Background research and information gathering

When the problem statement is defined, a plan for information gathering must be defined. The problem statement must be put on the agenda, and evaluated against similar approaches and projects. Relevant literature must be found and used to evaluate what is relevant to use for the project plan.
Sprint 2 - Project planning

The project implementation period must be put under the scope, and an analysis must be made outlining what factors to consider when planning and designing the project. Stakeholders must be included in this phase to make sure that the project plan is applicable with the end-user needs. A risk analysis must be made outlining policies and concerns that may change the project outcome, if something unexpected happens.

Sprint 3- Design period

A project design must be made containing models for design and workflows. Use-cases must be defined and decisions on technologies and development-methodologies (SCRUM, Kanban) must be made. A proper development-plan should be made, giving stakeholders the opportunity to monitor the project implementation during the development phase. The design-plan should outline the following:

- Documentation specifications
- Revision control
- Inventory specifications
- Software development methodology specifications to keep track of the development process.
- Technical specifications

Sprint 4 - Development period

The development period will focus on implementing a prototype as according to the plan decided in the design period. Revision control and issue tracking will keep track of development progression, and should help the teacher supervise the developer to keep on track when it comes to implementing a proper working solution as according to plan. The development-period will consist of several small sprints with several iterations, where each iteration focuses on solving a specific task.

The use of revision control will make it possible to keep track of change-history during development. If something needs to be to a previous version, revision control can make this possible by reverting the content to an earlier version.

Sprint 5 - Deployment period

The deployment period will focus on building the production infrastructure, and to deploy the code developed during the development phase. If a new change has been made to the code, the module must be redeployed. The deployment period will focus on keeping the system running, and focus on improvements that can be made to the application and the infrastructure.
Sprint 6 - Analysis period

The analysis period will be at the end of the project period. This sprint will focus on analysing gathered data during the project. The analysis will help conclude the project and will take the problem statement into focus.

Choosing the data to analyse

The goal with the analysis-period is to use the data gathered to help answer the problem statement presented in the introduction-chapter. The adjectives used in the problem statement were described into detail in the beginning of the Approach-chapter. Scaling and value-driven will be key terms when the analysis is conducted. Before the analysis can be started, a plan must be made for how the data should be conducted. The prototype aims to be used in a course at NTNU Gjøvik starting January 2016. The research will consist of real student-data, that can be used directly in different statistical analyses. The different analyses will be outlined in the next subsections.

Uptime Challenge

The students will participate in a challenge called the uptime challenge. The uptime challenge is a challenge where each student-group participates in a competition where the goal is to administer and operate a web site, and governing the infrastructure running the site. A set of tests will execute against the student sites, and the students will receive reward/punishment costs based on the test-results over time. During the uptime challenge, tests will execute continuously against student-sites and produce new reports and rewards over time.

Evaluating student effort

An analysis looking evaluating the student effort must be conducted. The analysis should investigate if students pay attention to their infrastructures, and look at their progression. The analysis should be conducted by using gathered reports on different groups. The analysis should comparative, and should look at commonalities between reports from different tests for the same group. The commonalities can be used to see the bigger picture of the student’s effort by using different test-results from the same time-periods focusing on different aspects.

Uptime evaluation

Several analyses must be conducted focusing on the uptime for each group. By looking at the uptime-percentage for each group, it becomes easier to evaluate student-progression. The uptime-percentage will tell the tester the amount of time the student-site was up and running. A site is considered as up if it responds to a request. The uptime-percentage can also contribute to an evaluation of the student-effort in the course.
The uptime-percentage could give the students a better overview of the stability of their site. This could be compared to a company hosting a web site. If the site stops working, the company would lose money. One of the goals with the course is to learn students about the operational perspectives on operating an infrastructure. Two of the main principles when hosting web site is to keep the site up and running at all times, and at the same time consume the minimum required amount of performance.

**Scaling analysis**

The project aims to investigate different technologies, used to automate scaling in an infrastructure. An analysis must be made outlining one specific scaling-methodology. This analysis should present a set of graphs and numbers indicating how the scaling can be achieved. The aim with the analysis is to evaluate how the number of tests deployed per user can be adjusted in context with the number of running workers.

**Student survey**

At the end of the project, a survey will be conducted for the students. The survey should evaluate the prototype used in the course, and if the use of the prototype in the uptime challenge motivates the students to work harder. The statistics from the survey will be presented in the Analysis chapter, and discussed in the Discussion chapter.

### 3.7 Determining design-model

The design-model aims to present how a prototype could be made. The design-model presents different use-cases and workflows that must be implemented for the system to work properly.

**Approach outline**

The project-phase section outlines the different sprints, and the main focus for each sprint. A more general model has been designed in order to outline different aspects of the project period. Each period is named after the chapters in the thesis. The aim is to outline how each period relates to the next. Each period can be considered as a project dependency. Some of the dependencies rely on the previous dependency to be finished before it can start. The rest of the dependencies form each other over time.
Figure 3.1: A figure outlining each period of the project. The diamonds represent objects that must be created and used as reference for the research. The circle represents the problem statement which is used through the entire research period.
Chapter 4

Design

4.0.1 Introduction

This chapter will cover the design section of the application that will be made. The problem statement states that the goal with the project is to create a framework that can be used to evaluate running IT-services. The aim is to develop a prototype that can show how the evaluation-methodology can be put into practice. The implementation period will be divided into different phases where the aim is to develop and investigate different technologies relevant for the project.

4.1 Model

When developing large-scale infrastructures with the ability to be administered by several users at the same time, comes the need for a detailed model. The model gives an overview of the different modules and instances that all play a big part in the system as a whole. The model is designed from a microservice point of view. A microservice is defined as “a particular way of designing software applications as suites of independently deployable services.” [27]. It is a requirement that each module runs on a separate instance. Each instance in the model doesn’t have to rely on the other instances to be up and running. If one instance stops working, the other instances will still continue to work as normal.

4.1.1 Scenario

The project aims to develop a large-scale application with ability to deploy different set of tests to different instances. Early, it was chosen to follow a manager-worker model when designing the system.

Manager-worker model

A company consists of departments; each department consists of a set of workers; workers have different managers. The managers are responsible for assigning tasks to the workers. If the workers fail to complete the
assigned tasks in a satisfactory way, the Manager is held responsible. This model suits the system in a good way.

Figure 4.1: A figure showing a typical company-structure where a manager governs a set of workers
4.2 Phase I - Building the infrastructure

A cloud infrastructure is defined as “The collection of hardware and software that enables the five essential characteristics of cloud computing.” [40]. The five characteristics of an infrastructure describe a set of conditions an infrastructure must fulfill to be qualified as a cloud infrastructure. The set of conditions include: On-demand self-service, broad network access, resource pooling, rapid elasticity and measured service.[40] An IT-infrastructure is the basic structure that a system can run on top on. When building an infrastructure, there are several aspects that need to be taken into consideration. Enterprise organizations may set economy as their top priority when making decisions on building an infrastructure. Researchers may use performance as their top priority to be able to conduct massive data simulations.

Building an infrastructure can be a long process. A project infrastructure may be redefined during the entire implementation phase. When coding big applications, the need for more project-servers may be necessary. Estimating the cost of running an infrastructure can be difficult when several critical factors may change during the research phase.

4.2.1 Project infrastructure

One of the first decisions made in the implementation phase is mapping the infrastructure. The project will consist of two different layers: A static communication layer, and a dynamic deployment layer. The static communication must be configured before the dynamic deployment layer can be set up.

Static communication layer

The static communication layer is the infrastructure layer where data is being written, and fetched from. This is the structure that will persist through deployments of different dynamic infrastructures. The layer will consist of the following:

- A database server running CouchDB
- An interpreter server with responsibility of making calls to and from the database server.
- A communication-API distributed from the interpreter making it possible for the dynamic infrastructure to communicate in to the static infrastructure

Dynamic deployment infrastructure

This infrastructure is responsible for the test-deployment. When a user wants to execute a test, a new deployment infrastructure is needed. Each deployment infrastructure is responsible for its own tests. The infrastructure consists of two parts:
• Manager- Test deployment server  
• Workers- Instances that executes test-jobs from the job-queue.

The dynamic deployment infrastructure will communicate in to the static infrastructure. The Manager will be the server responsible for initiating the communication. When a test is performed, the results are sent back to the Manager, which is then sent through the API to the database for saving.

4.2.2 Workflow

The frameworks consist of several different modules that follows different workflows. To make the different modules interact and communicate, good communication flows must be defined. Sometimes, functions must be called in a given order, while other functions are called when events happen. Therefore, two different mechanisms for communication are defined:

A communication-API distributed on the Interpreter. - Makes it possible for the Manager to communicate with the database upon request.

A queue - Makes it possible for the workers to communicate with the Managers. When a job is executed on a worker. It is fetched from the work queue; executed and sent back to the Manager on a report queue. Using queues makes it possible to store jobs over time. The Manager will process elements from the report queue in a scheduled order, while managing posting new jobs to the work queue.

During runtime different modules must initiate communication on different timeslots. To ensure robustness and stability of the entire application, queues can save jobs from the last program execution. If the Manager is turned off temporary during runtime, it can automatically continue its process of posting/fetching jobs to/from the work queue.

General workflow

The framework follows a general workflow. This section will describe different states the application may enter during runtime:

A manager is spawned. Each manager has its own local configuration, and performs a function call to the interpreter to fetch configuration from the database

The Manager initiates the users and test-subjects from the configuration.

The Manager puts the tests on the job-queue. The Manager is now in idle mode.

The workers fetch jobs from the queue, and executes it against the target system. When a job is executed, a report is written and sent back to the Manager on a different report queue.
The Manager fetches a job from the report queue, and checks the results to validate the job. The report is sent back to the Interpreter through the API, for later to be saved in the database.

The Manager returns to posting jobs on the work queue. The process is repeated.

Figure 4.2: The total infrastructure, showing how different modules communicate with each other.
4.3 Manager

This section contains more information about the Manager-server. The Manager will serve as the responsible instance for fetching and posting jobs to and from the job-queue.

4.3.1 Overview

The manager should have the ability to run on several platforms. A manager could interact as a virtual machine or an isolated container. The manager is meant to run over longer periods without any manual interference. The user should have the ability to log in to the Manager during runtime, and monitor the process. The manager has several responsibilities to govern. Some of the responsibility-areas include:

- Communicate with the Interpreter
- Post and fetch jobs to the work queue
- Govern one specific test-case, and make sure that tests are deployed automatically over time.

A manager governs the workers, and has to make sure that the workers actually perform the jobs assigned from the manager. If a test has not executed properly, the Manager should have the ability to spawn a new worker to take over the job for the old worker. The Manager could govern several workers at the same time. The Manager does not need to know anything about the state of the workers. The only thing the Manager need to worry about, is to fetch results from the workers.

Cardinality

The static infrastructure could have several managers interacting at the same time. Each manager is responsible for one set of tests, and will deploy that test continuously. Each manager could also govern several workers.

4.4 Workers

A worker is spawned with one responsibility: To fetch and execute jobs from the work queue. When a test is fetched from the queue. The workers execute the job against a target system. When the job is finished. The Worker generates a report with the results and sends it back to the Manager. When the report is sent. The Worker goes back to idle mode where it listens on the work queue.

A worker is meant to run as a closed instance. The main goal is to let the Worker execute the tests automatically. If a worker enters an error state, the instance will be deleted and a new one spawned instantly.
4.5 The interpreter

The main responsibility for the Interpreter is to receive function calls from the different managers. Every time a function call is made to the interpreter, an underlying mechanism must make sure the call is forwarded through the right function to the database.

The Interpreter will serve as a translator for several different managers deploying different tests. The Interpreter will act as a communicator with the database and will only perform tasks when asked. The Interpreter will distribute an API the Managers can connect to, using a set of distributed functions. The goal with using an interpreter between the database and the Managers is to make sure that all the function calls to the database are made through a third party. This will again increase the computer security of the infrastructure and will also help preventing data to be modified from several sources at the same time.

The interpreter itself will be developed using Python, and will be using Python libraries such as Pyro4 to distribute its API.

4.5.1 Workflows

The interpreter will follow different workflows when a function call is made. Below some examples are mentioned:

A manager requesting a user

The Manager sends a function call to the Interpreter containing user information.

The Interpreter receives the function call from the Manager and forwards it to the database.

The database then returns the user data if they user already exists in the database.

If the user doesn't exist in the database, a function call is made from the interpreter containing the user-metadata, so that the user can be created.

A manager requesting configuration

The Manager sends a function call to the Interpreter containing a request for new configuration.

The function call is received on the Interpreter, and is forwarded to the database.

The configuration data is sent back to the Interpreter and returned to the Manager.
The Manager wants to update data on a field in the database
A manager sends a function called to the Interpreter containing a data field with new values.
The data is then forwarded to the database with a task to update a specific field.
The database then queries the field and updates the meta-information.
A return call is made to the Manager saying if the update request was successful or not.

The Manager wants to create a report
When a job is finished and is sent from the Worker to the Manager, the Manager will send a report through the Interpreter-API to store the report persistently in the database.
The Interpreter will receive the data from the Manager, and will forward it to the database.
The report is then created in the database as a separate document.
If a document was successfully created, the Interpreter will send a message to the Manager saying that the report was written.

4.6 The database
The database will be the most important instance in the infrastructure. The database will store all data written during runtime. CouchDB will be used as the database technology tool. CouchDB offers a web-configuration interface that makes it possible to access stored data and change it in the web-browser. This opens the opportunity to update stored data long after it has been created.
All communication to the database will go through the interpreter. The database itself will have a deny-policy in the firewall for all ports except the ports needed for the interpreter to communicate. The database will not perform any tasks unless the interpreter requests it.
The database should also support the opportunity of connecting a third-party dashboard to it. CouchDB is HTTP-based, and supports queries directly through simple HTTP-calls. Queries can also be made using JavaScript. For queries handling big data, complex query-functions can be made and stored persistently for later usage.
The database will consist of three different tables:

• One table containing user-account information. A user account could either be a group or a single user.

• One table containing reports. A report will consist specific information of a job performed by a worker.
• One table containing the configuration. Every configuration-entry has a teacher as a primary key. This means that every teacher could have their own configuration in the database.

4.7 Summarizing the infrastructure

The total infrastructure will consist of several crucial modules. The importance of making every module able to handle downtime from another module becomes crucial. Every module will be implemented with their own exception handling. This exception handling will take care of unexpected problems that may arise during runtime. A manager should be able to operate posting jobs to the workers without needing the interpreter to be up. If the interpreter is down, the report queue will only increase in size until the interpreter is up. If a manager hasn’t received any reports from a worker in a long time, that specific worker must be torn down and re-deployed. Separate mechanisms will monitor the system during runtime. The user should be able to perform automated monitoring test on the different modules. Automated event-handlers can be configured if a check returns a critical state on a module. The system should be capable of communicating with different managers programmed in different programming languages. The infrastructure will use queues as a communication tool between the workers and the Managers. Every manager will govern it’s own workers. Every worker will be responsible of performing jobs requested by the Manager. The workers doesn’t need to know anything about the state of the Manager, they will only communicate using the queue.

The managers will communicate to the static infrastructure using the API on the interpreter. The interpreter will only act as a communication tunnel when needed. If a situation may arise where the API is not working, The manager should still be able to posts new jobs to the workers, without forwarding reports to the interpreter.

4.7.1 Queues

The main communication-form between modules will be queues. When using queues, one makes it possible to post tasks on a waitlist that will be executed whenever the instance has got a free time-slot. Queues have the ability to store data persistently over time. A queue is defined as "a sequence of messages or jobs held in temporary storage awaiting transmission or processing" [36] Queues open up communication between several instances at the same time. When one instance wants a specific job performed, this job can be advertised through a queue. When the job is posted on the queue, another instance can fetch the element; perform the job and return the results on a different queue.

One of the most popular queue-technologies used today is RabbitMQ. RabbitMQ uses AMQP, which is defined as "a messaging protocol that enables conforming client applications to communicate with conforming
messaging middleware brokers."[50]. RabbitMQ will be used as the message-broker between the Managers and the workers. RabbitMQ has the ability to govern several different queues at the same time. Different instances can query different queues, and fetch data immediately after an element has been posted to the specific queue.

The main goal is to develop two main set of queues: report queues and work queues. When an element is posted to the queue, the content will be of the data-type String. In order for the receiving instance to parse the data properly, type conversion is needed. Every instance operating with queues needs to have such mechanisms running upon arrival of a new element from the queue. The next sections will describe the different queue-types into detail.

**Report queue**

The report queue is meant to contain reports issued from the workers. When the Worker has executed a job, the results are posted on a report queue, which in turn is controlled by the Manager. The Manager labels the report with the group-owner, timestamp and an ID. Once the Manager is finished processing the report, it is forwarded to the Interpreter to be saved in the database. During runtime, only the Manager is allowed to make changes to content in the report queue. When a worker has finished a job, it is up to the Manager to process the report before it is sent into the static infrastructure. Since every manager only governs one set of tests, one corresponding report queue will exist for each manager.

**Work queue**

The work queue will contain information about an executable path that the worker can use to execute a test. The Manager will operate after a time-schedule that will govern what state the Manager operates in. The manager will differ between two states: job posting and report-validation. When the Manager is in job-posting state, it will post a given number of jobs to every user configured in the database. These jobs are then posted to a work queue that the workers will listen on. When a worker enters listening mode, it has the ability to fetch newly arrived jobs on the queue. When the job is fetched, it is executed against the target system. The result is then written in a report and is sent back to the Manager on the report queue.

**Connection to the queues**

In order to be able to fetch or post content to a queue, a set of functions must be implemented. The queue-mechanism will run on a separate instance. The instance trying to post an element must authenticate the queue-server before the element can be posted. The same mechanism must also be implemented for the instance fetching the element. The instance operating on one side of the queue does not need to know anything about instances
operating on the other side of the queue. The only thing they need to relate to, is the queue. The RabbitMQ-broker uses the TCP protocol for incoming connections. Every time a new connection is made, a three-way handshake is completed. This connection will be reset every time an instance reconnects to fetch a new item.

4.7.2 API

When the Manager needs to post or fetch elements to and from the database, it has to go through the Interpreter. The Interpreter distributes and API that allows the Managers to connect to the static infrastructure. The API will be implemented with a set of standard functions. These functions will perform HTTP-GET and HTTP-POST calls to and from the database. Some of the key-mechanisms are mentioned below:

- Check users
- Fetch configuration
- Update existing database content

4.7.3 Test case: Bookface

The project will be designed and optimized to work with a website called Bookface. Bookface is a website developed by Kyrre Begnum, and shares similar functionality as Facebook. When browsing Bookface, one has the ability to: add users, download the front-page, create a post and post a comment to an existing post.

Bookface will be used as a test case when developing tests for the prototype. The first weeks of implementation will focus on optimising the framework to work against the Bookface site.

4.7.4 Multi-Language support

The tests will be written in different programming languages. When developing a tool with ability to execute scripts written in different programming languages, it is important to think simple. The first step to ensure multi-platform support is to make sure that output from different scripts are in the same format. By using the same formatting on the output, one can create generic parsing mechanisms with ability to parse text written using a given standard.
JSON

"JSON is a text format that facilitates structured data interchange between all programming languages." [17] Output from the tests will be converted into JSON, before it is sent back to the Manager. JSON is a formatting standard used on text when it is sent between data instances. JSON consists of attribute-value pairs separated by colon. When the worker has successfully executed a test, the output is sent to a converting mechanism that returns the data as JSON-objects.

![Diagram of data flow](image)

Figure 4.3: A figure showing how output is converted to JSON on the workers and sent back to the Manager.

4.8 Automation

During runtime, the system should be able to deploy jobs, along with creating reports. The goal is to let the system run continuously for longer periods of time without human interference. When the Manager is started, a set of workers will be spawned instantly into listening state. Once a
worker is spawned, it will start fetching jobs from the work queue, and create a job-report once a job is finished.

4.8.1 Exception handling on the workers

If something unexpected happens during runtime that prevents the worker from executing a job successfully, an error report will be written and sent back to the Manager. If a worker enters a state where it is no longer is fully functioning, a kill-signal is sent. A new worker will instantly be spawned replacing the old worker. Several mechanisms can be used to ensure that the workers are doing their job properly. Some examples are mentioned below.

4.8.2 Time-check

When the Manager posts new elements to the queue, a mechanism can be implemented to ensure that jobs is fetched from the queue within a threshold. If there is a drastic increase in number of waiting jobs on a queue within a short time interval, more workers can be spawned automatically. If the number of jobs on the queue is to low, and the workers enters idle state, some of the workers can be terminated in order to save performance. If a worker fetches a new job and fails to successfully execute the task, it will be terminated. The managers will be deploying jobs on the queue using different deployment-algorithms and datasets. The datasets contains numbers that will help the Manager to calculate the number of tests to deploy per user. When one of the jobs fail, the job is ignored due to low importance.

4.8.3 Queue-monitoring

A mechanism monitoring the queues may be implemented. The mechanism can monitor the number of elements on the queues, and raise an alarm if the number of elements exceeds a certain limit. Depending on the number of jobs on the work queue, the Manager should be able to deploy more workers to finish new incoming jobs, or terminate idle workers when they are not needed anymore.

4.8.4 ICMP-check

When the workers are running, an ICMP-check can see if the workers are up. If a workers stops responding to the ICMP-requests, it will be terminated. A new worker will spawn immediately to replace the old one.

4.9 Job-deployment algorithms

In order to calculate the number of jobs to deploy per user, different mechanisms must be implemented on the Managers. If the type of job requires heavy performance from the worker, a small set of jobs may be
sufficient. Some of the jobs may be very brief, and simple to execute. Several jobs can therefore be deployed each deployment. The next section describes different algorithms to follow when deciding the number of jobs to deploy per user.

4.9.1 Webuse

Webuse is a script with ability to perform a number of different tasks on Bookface. Some of the tasks include:

- Download the front-page
- Create a post
- Create a new user
- Create a comment to an existing post

Webuse performs small tasks on the Bookface-site. None of the tasks are performance-consuming, and doesn’t take long to complete. The goal with using Webuse is to simulate human traffic on a web site. When running Webuse over time, one can analyse patterns and see how the activity per user changes over time.

Randomizing

Webuse supports randomizing task-assignments. This means that one task may differ from the previous one. When using the randomizing option, the output will differ each time for each test-execution. Some of the tasks carry more weight in the election-process of choosing the Webuse-function to execute. The more weight a task carries, the bigger the probability that the task is chosen.

A data-file containing 288 entries governs the number of Webuse-tasks to deploy per user. Each entry represents 5 minutes of a day. (This is described into detail in the Workload-profile section)

4.9.2 Httperf

Httperf is used to generate traffic to Bookface. (Httperf is described into detail in the background-chapter.) Httperf has the ability to generate traffic over a given set of time. In order for this to work properly, the following must be configured:

- Rate- The number of connections per second
- Total - The total number of connections
- Calls per connection
Httpperf uses workload-profiles to determine the connection-rate. The total number of connections is calculated after the formula:

\[ \text{Total connections} = 300 \times \text{rate} \]

300 is used to represent the number of seconds in a five-minute interval. Every Httpperf-test is executed for a five-minute interval. This test is considered to be a time, and performance-consuming job. Only one test per user is therefore deployed.

### 4.9.3 Rewarding

In order to give students instant feedback on their test-results, a balance can be set for each group. The balance represents a number that will increase or decrease based on test-results. The rewarding mechanism will be used by the Purser-test and Clerk-test. The balance is meant to simulate a bank-account, where the user is rewarded with extra cash, if the tests results are positive. The groups must also pay an hourly fee for their running infrastructure. The Clerk check will keep track of the number of virtual machines running per group-infrastructure, and calculate an hourly fee that will be withdrawn from the group’s balance. Purser and Clerk are described into detail below.

### 4.9.4 Purser

The Purser is responsible for evaluating the Bookface site of the groups. The Purser check will connect to Bookface, download the content of the front-page, evaluate the site latency and search for a word on the site. If the site contains a given word, a status of OK will be given. Based on the given parameters, a reward will be calculated. If the site is down, the Purser will withdraw points from the group’s balance. Downtime is a costly affair, and the intention of running Purser, is to give the students valuable feedback in form of a reward or punishment cost. A full reward can be calculated for the group if the download-time is beneath a certain limit and if a given word is present on the site. If the HTTP-response code is implicating that the site is down (503). A partial reward will be given.

Purser will run alongside Clerk, and maintain the groups balance.

### 4.9.5 Clerk

According to Oxford Dictionaries a clerk is defined as "A person employed in an office or bank to keep records, accounts, and undertake other routine administrative duties"[19]. In the IT-industry, a clerk is often a person with responsibility of keeping track of the physical inventory in a company. The clerk keeps track of the financial spending of an infrastructure-department. One example duty can be to keep track of the number of servers running
in an environment and calculate a cost of keeping the servers running over time.

A Clerk test will be used in the project. The Clerk test will execute against the groups OpenStack accounts, and count the number of virtual servers running. In OpenStack, different servers are tagged with a flavor that describes the hardware specifications. (I.E. A flavor of type Medium is operating with 2 CPUs and 4 GB RAM). The Clerk test operates with a price-list for the different flavors. The Clerk counts the number of instances running with different flavors, and calculates a cost based on the following formula:

\[
\text{Price}_{\text{servers}} = \frac{\text{Unitprice}_{\text{flavor}} \times \text{Count}_{\text{units}}}{12}
\]

The Clerk is set to execute every fifth minute. To calculate a price for five minute, the hourly price is divided by 12.

### 4.9.6 Leeshore

A lee shore is defined as "A shore lying on the side of a ship."[21]. During the project period, a Leeshore will operate as a script with intention of powering of virtual machines for different groups. When the Leeshore-script is running, it connects to the different OpenStack environments belonging to each group, and turns off a random server-instance. The Leeshore-script will execute once per group every second hour. The intention of running the Leeshore-script is to take down the Bookface sites to each group. By running Leeshore, one can evaluate the infrastructure running Bookface for each group, and see if the site is still running if a server is turned off.

### 4.10 Workload-profiles

The workload-profiles play a big part in the prototype for this project. Workload-profiles will be used as the reference dataset for Webuse, and Httperf. This section will cover in detail how workload-profiles are used, and show the generated numbers correlate.

#### Definitions

<table>
<thead>
<tr>
<th>Variables</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Total number of groups used in the calculation</td>
</tr>
<tr>
<td>g</td>
<td>Unique group-identifier</td>
</tr>
<tr>
<td>i</td>
<td>Workload-profile interval [1,288]</td>
</tr>
<tr>
<td>c</td>
<td>Cut-off = offset = [1,288]</td>
</tr>
</tbody>
</table>
Introduction

The definitions above illustrate some of the key terms used in the next sections. The next sections will illustrate how workload-profiles can be calculated and their usage. The next paragraphs will introduce some new terms, and illustrate their usage from formulas and specific examples. The definitions above can be used as reference to understand the key-terms needed in the calculations below.

Base number and noise

A base number is a number found in one of the workload-profiles. The base number represents a number that later will be used to generate a strength-number. The base number is used along with a second number referred to as noise. The noise-number is used along with the base-number to generate numbers within certain ranges. (I.E. Base number = 5 and noise = 3 can generate a number in the range [5-3,5+3]=[2,8])

The base number and noise are static numbers already decided in workload-profiles. To be able to decide what base number to use, the workload-profile interval and cutoff are used. The workload-profile interval represents 24 hours in five-minute intervals. (There are 288 five-minute intervals per 24-hour period). The cutoff is used to make sure that different groups have different starting-points in the workload-profile. The workload profiles are designed to be followed by reading one line per calculation. Below are specific examples how this is achieved:

Interval and offset calculation example

This subsection shows an example on how a starting-point \( i \) in the workload-profile can be calculated. Each workload-profile is 288 lines. 288 represents the interval-length of the workload-profile interval \( i \). This example illustrates a group referred to as groupX. GroupX wants to use workload-profiles in their project. Before they can start the calculation, they need to know the following:

- Workload-profile interval
- Decided cutoff, also referred to as offset.

GroupX has a cutoff set to 20. The cutoff will be used to decide a starting point in the workload-profile interval range \([1,288]\). Before the cutoff can be used, the group must use a Unix-timestamp (Calculated machine-readable integer) to calculate what five-minute interval they are starting in. The timestamp formula is used as part of the initiation process of the calculations. This means that a program using workload-profiles can be restarted, causing the program to continue from the same place in the workload-profile interval upon next program execution. The calculation
described is performed using the following formula:

\[ i_{start} = \frac{(timestamp_{unixtime} \mod 86400)}{300} \]

The start-interval is always represented as an integer.

By having a Unix-timestamp of 1459859383, a number representing the five-minute interval count \( j \) can be calculated:

\[ j_{start} = \frac{(1459859383 \mod 86400)}{300} \approx 150 \]

The newly calculated interval-number can then be used along with the cutoff = 20 to calculate a starting point \( i_g \) in the workload-profile. This is calculated using the formula below:

\[ i_{start+offset} = (j_{start} + cutoff) \mod 288 \]

By using the calculated interval-number \( j \) from the previous example, one can calculate a starting-point in the workload-profile:

\[ i_{start+offset} = (150 + 20) \mod 288 = 170 \]

The modulo ( \( \mod \) ) operation is used to ensure that the workload-profile interval is reset to 0 if the calculated starting point is equal to 288.

**Base number and noise usage example**

From the previous section a starting-point of 170 was calculated. To use this starting point, one must have a predefined workload-profile consisting of 288 lines. Each line in the workload-profile are built using the following structure:

\[ \text{startingpoint : basenumber : calculation} - \text{algorithm : noise} \]

The calculation-algorithm is a predefined algorithm that will be used to generate a strength in a given range. In the workload-profiles used during the development-phase, the algorithms used are:

- Gaussian distribution algorithm (g)
- Uniform distribution algorithm (r, also known as random)
The starting-point calculated from the example above can be used directly to decide the starting-line in the workload-profile. The base number(a) and noise(n) are based from the following formula:

\[
\begin{align*}
a(i, c_g) \\
n(i, c_g)
\end{align*}
\]

c represents the given cutoff for a group. In the example, the cutoff is set to be 20 for groupX.

By setting i to 170 (As calculated in the previous example). The represented base number and noise can be written as:

\[
\begin{align*}
a(170, 20) \\
n(170, 20)
\end{align*}
\]

The base number and noise will vary from profile to profile. The workload-profile entry used in the examples are based on the workload-profiles that will be used during the development-phase.

**170 : 5 : r : 3**

The entry above states that the uniform algorithm should be used to generate a strength using base number=5 and noise=3. The uniform algorithm will generate a strength(t) after the following formula.

\[
t_{\text{uniform}} = a + \text{uniform}([-n, n])
\]

By using base number a =5 and noise n =3, this can be set into the formula above to calculate the strength t.

\[
t_{\text{uniform}} = 5 + \text{uniform}([-3, 3])
\]

By predefining a generated noise of (-2), the strength can be calculated to be

\[
t_{\text{uniform}} = 5 + (-2) = 5 - 2 = 3
\]

This means that groupX has used the workload-profiles to generate a strength of 3, using the conditions above.
Next iteration

The examples listed above illustrate one strength-calculation for one group. Normally several groups will use the same workload-profile to calculate a strength using the methodology above. The examples illustrate the initiation process. When one calculation has been performed, and it is time to calculate a new strength number for the same group, the starting-point(i) is increased by one. Using a starting-point(i) from the examples above, it would mean that i increases to 171. The content on line 171 in the workload-profile will then be used to calculate a new strength using the steps provided in the previous examples. When the starting-point has increased to 288. The process is reset back to the start of the workload-profile interval (1).

4.10.1 Total base number-sum and noise-sum

It is possible to get the total sum of what the base number and the noise will be. To make it possible to calculate the total base number-sum and noise-sum. A base number must be found for each group using the workload-profile, and summarized. The sum can be calculated after the following formula:

- $A_i$ represents the base number-sum for all groups using a workload-profile
- $g$ represents one group, $G$ is the total number of groups
- $c_g$ represents the cutoff

\[
A_i = \sum_{g=0}^{G} a(i, c_g)
\]

The same calculation is performed to find the sum for the noise for all groups:

\[
N_i = \sum_{g=0}^{G} n(i, c_g)
\]

- $N_i$ represents the noise-sum for all groups using a workload-profile

Sum example

Using the formulas above, the base number-sum and noise-sum can be calculated: In this example two groups (groupX and groupY) groupX has base number=5 and noise=3 with cutoff=20 groupY has base number =4 and noise =2 with cutoff =40 . The total basenumbersum and noisesum can then be calculated for the groups:
\[ A_i = \sum_{g=2}^{g=1} a(i, c_g) = 5 + 3 = 8 \]

\[ N_i = \sum_{g=2}^{g=1} n(i, c_g) = 4 + 2 = 6 \]

By using different basenumbers and cutoffs from the workload-profile, one can see that the base number-sum always will be close to the previous sum.

### 4.10.2 Strength sum

One of the previous examples gave an example how a strength-number could be calculated. The strength number is the indicated strength that will be used by different tests to decide connection-rate, and number of tests to deploy. For every test-deployment, a new strength is calculated for every group. By summarizing the calculated strength-number for each group, one can get an overview of the performance-requirements a manager must have to be able to deploy an equal amount of tests as the calculated strength-sum. This number will vary a little from different deployments. The generated strength for each group will differ every time a calculation is executed, because the calculation-algorithm (Gaussian or Uniform) used may produce different noise to be used in the strength-calculation for each group. This may cause the strength sum to differ more than the base number-sum.

### 4.10.3 Capacity

In order for a manager to be able to predict the number of workers needed to complete all deployed tests within a deployment interval, one can define a capacity (Ca). The capacity is a predefined unit used to define the minimum technical requirement the Manager need for all jobs to finish within one deployment interval. The sections above will outline different approaches on how workload-profiles can be used to estimate the proper capacity for a manager. The subsection below will outline the relationship between the strength-sum (T) and the Capacity (Ca)

**Strength-sum versus capacity**

The strength-sum(T) is outlined in the previous section and indicates the actual strength used for a given deployment for all groups. In normal system, a system administrator will predefined the capacity. This cannot be altered or changed. To make sure that all deployed jobs will be executed within the deployment interval, one can use the following expression:
The expression above indicates that the set capacity for a workload-profile (Ca) should strive to be bigger than the total strength-sum for the same workload profile T. In order to not set the capacity too high, an overcapacity probability (OCP) is introduced to make sure that the system only runs with the needed performance. Sometimes, this may result in capacity-problems. The overcapacity-factor describes how much overcapacity the system can tolerate over time. The overcapacity-probability should not exceed 0.2

\[ OCP \leq 0.2 \]

The probability of having more capacity than waiting jobs is equal to 1-OCP.

### 4.10.4 Sample space

A sample space is a collection of all possible outcomes of a random experiment. [1]. A sample space can be used to represent a range of possible outcomes of a calculation. When calculating the strength for each group in a deployment, it is possible to predict the probability of what the generated number will be based on the following criteria:

- noise
- base number

By illustrating with an example:

#### Strength probability calculation

GroupX has got a base number=2 and noise=1 from the workload-profile. When using the uniform algorithm to generate strength. The probability for each strength can be represented as the following sample space:

\[ S_{\text{strength}} = 3, 2, 1 \]

When using the uniform algorithm, the probability of getting strength =3 equals to 0.33 since there are 3 total finite elements in the sample space.

The example can be used further to calculate the probability of getting a certain strength for a group over several deployments. This can be illustrated in a new sample space:

\[ S_{gli} = \{t_0 : P(t_0), t_1 : P(t_1), t_2 : P(t_2), \ldots t_{\text{Max}} : P(t_{\text{Max}})\} \]

The sample space above list the probability of getting a given strength every deployment. The probability of getting strength=2 in the first deployment and then strength=3 the next deployment can be illustrated in the following sample space.
\[ S_{gi} = \{ 2 : P(2), 3 : P(3), \} \]

Each probability within the sample-space above have different possible outcomes because the base number and noise may vary on different workload-profile entries. Each probability must therefore be calculated separately.

### 4.10.5 Total strength probability

The previous section described the probability of getting a given generated strength for one group. To be able to estimate a given capacity. The strength-sum(T) can be used along with the predefined capacity(Ca). By using the overcapacity probability(OCP), it is possible to set up an expression stating a technical requirement for the Manager. The expression can be seen below.

\[ P(Ca_i \geq T_i) = 1 - OCP \]

The expression states that the probability of having enough capacity(Ca) available for the generated strength-sum(T) should be equal to 1-OCP. OCP illustrates an allowed deviation where the capacity is to low for the generated strength-sum.

### 4.10.6 Total strength probability sample space

By using the normal-distribution model it is possible calculate the total strength-probability for all groups. The next sections will describe this into detail.

**Total strength sum**

To be able to calculate the probability of getting a given total strength sum. One can use the formula defined in the previous section. The total strength-sum T(i) can be calculated by using the formula:

\[ T_i = \sum_{g=1}^{G} t(i) \]

The variance T(gi) must then be element in a normal distribution set that can be seen below.

\[ T_{gi} \sim N(a_{gi}, \sigma) \]

\[ T_i = \sum_{g=G}^{g=1} t(i) \sim N(\sum_{g=G}^{g=1} a_g(i)) \]
The variance \( T(g_i) \) can be used to express the standard deviation using
the following formulas

\[
Var(T_{gi}) = \sigma_{gi}^2
\]

The total standard deviation for each strength-sum for every deployment

\[
Var(T_{1i} + T_{2i} + \ldots + T_{Gi}) = \sqrt{\sigma_{1i}^2 + \sigma_{2i}^2 + \ldots + \sigma_{Gi}^2}
\]

The standard deviation can be used to set up the normal distribution set

\[
T_{gi} \sim N(\sum_{g=1}^{G} G(i), \sqrt{\sigma_{1i}^2 + \sigma_{2i}^2 + \ldots + \sigma_{Gi}^2})
\]

The formula above can be used directly to calculate the probability of

\[\text{a number being generated using the uniform distribution model. This}
\]

\[\text{probability can then be used to project the number of workers needed for}
\]

\[\text{that specific job-deployment.}
\]

**Negative values**

The Gaussian distribution model allows negative numbers to be generated.
When negative numbers is used with Webuse, the absolute value will be
used as the number of test to be generated.

### 4.11 Determining live-evaluation criteria

The system developed during the project phase aims to be used by teachers
to evaluate student projects. To create a simple evaluation-methodology,
one can use a balance. The balance is a number that will be maintained
by the Purser check, and Clerk check. The Purser check is evaluating the
Bookface site every fifth minute. Based on the results, a reward/penalty
charge is calculated. This number will be updated against the balance.
While Purser has the ability to reward the users, the Clerk will charge
each group based on the number of running servers per OpenStack-flavor
in their infrastructure. The balance represents live-values over how the
evaluation results have been over time. Students has the ability to monitor
each others balance online, and they can keep track on how they are doing
compared to their classmates. The balance will be reset every week, in
order to make this a week to week-based competition for the students.
4.12 Historical evaluation

Every time a job has executed, the result is stored persistently in a database. Each report contains a timestamp, and the groupname. This can be used later to generate graphs looking at a group’s test-results over time. Each test contains a given set of parameters, that can be used to look at progression over time. Below an evaluation example is provided.

Evaluation example

GroupBox has been running their Bookface site for a four-week period. During that time, the Purser, Webuse, Httperf, Clerk and Leeshore-tests have been running constantly against the site. During the test-period, the group has taken the liberty to increase the size of their virtual environment, by adding extra database-instances for redundancy and more Bookface-webservers behind a loadbalancer.

When a teacher evaluates the Bookface site to the students, the teacher must look at the uptime status of Bookface over time in addition to evaluating the infrastructure of the students. The teacher can then look at Leeshore-reports, to see the servers that were turned off, and evaluate the response-time it took for the students to get the site up and running again. If the site still was up and running after Leeshore turned off one of their servers, it should count as a plus. If the Purser-test failed to access the student site, it would mean that the site is down. By looking at the timestamps from Webuse, and Purser, one can see when the site was up and running again.

The teacher also wants to look at the site-latency(response-time) when the Webuse-check, and Httperf check executed against the site. If Purser manages to download the front page of the site fast, it means that the latency is good. If the download-time takes to long, it would mean that the site is not working properly, even though the HTTP-response code is 200(OK).

By looking at the factors mentioned above. The teacher can evaluate the results against the Clerk reports for the infrastructure. If the students are running several servers at the same time, but the Bookface-site is down, it should count negatively. Running several servers in a production-environment could be a costly affair, and the students should thrive to run as few servers as possible with the highest uptime-percentage and lowest latency as possible.
Chapter 5

Implementation

This chapter will describe the implementation-phase of the project. It will go into detail on different tools used, in addition to how they are implemented. The chapter will focus on the model described in the design chapter, and how the infrastructure is built.

The first sections will introduce different technologies used, and how they fit into the model. The next sections will describe the different classes, and services running on the different instances in the infrastructure into detail.

5.1 Problem statement

During the design-phase, the sole focus was to design a framework that could help answer the problem statement listed in the introduction chapter. The first thing that was designed was the infrastructure. It was decided that the infrastructure would be built after a microservice-model, separating different modules into small individual services that runs on different instances. Each module was then planned and designed separately, before it was assembled together as a complete system. The primary objective during implementation was to develop a prototype framework optimized for educational tests.

5.2 Technologies

This section will go into detail on the technologies used in the implementation phase.

5.2.1 Github

Github is used as the revision control platform during the project period. Github was founded in 2008, and has per 2016 over 12 million users. Github uses git as their version control tool. Github is a version control platform, where users can store their code repositories. The users can choose between having public and private repositories. Github supports integration
between technologies (Jenkins, Puppet, Docker). By using hooks, one can trigger events and builds on other platforms, which is again synchronised with Github. The use of Github makes it possible for developers to keep track of their code-progression, along with documenting the process.

5.2.2 Trello

Trello is a project-management application launched in 2011 by Fog Creek Software. [52] Trello opens up for the users to define Kanban-boards with issue-tasks that must be completed. Each issue-task can be linked to a column and moved when one is working on the task. (I.E. To Do- Doing- Q and A - Done) Trello will be used as issue tracking to keep track of the implementation process. A Trello board can be shared between users. Several users can contribute on the same project using Trello. Trello will be used as an issue tracking platform during the development phase. Each module that must be implemented will be divided into smaller sub-tasks. Each sub-task will be represented as a Trello card that must be implemented. Trello enables the possibility of adding comments and attachments to cards. This makes it possible to document the progression, when working on a specific card.

5.2.3 Python

The main programming language during the implementation phase is Python. Python is an object oriented programming language developed by Guido van Rossum in early 1990s. Python emphasizes code readability [54], and aims to be expressed in fewer lines of code than similar programming languages. During the project, Python will be used as the main programming tool to develop the infrastructure. The managers, workers and API will be developed using Python 2.7.

Building

Python is compiled to byte-code during code-execution. When programming object-oriented with Python. The main-class will be recompiled upon every program-execution. The imported classes will be compiled and stored in separate .pyc-files. Upon next program execution, the .pyc-files can be used directly, unless the code has been changed or altered since the last program execution.

Version

Python distributes several version and frameworks. It was decided early in the implementation phase to use Python 2.7. Python 2.7 is one of the most supported versions, and has the ability to run in different environments.
5.2.4 RabbitMQ

RabbitMQ is a message-broker software that makes it possible to configure a queue-based system, where elements can be posted to a queue, and fetched later for processing. RabbitMQ offers the possibility of distributing jobs in a large scale environment where several nodes connect to the same RabbitMQ-instance. The managers will deploy new jobs for the workers on a RabbitMQ-based queue. Job-reports will also be generated using the same technology. RabbitMQ makes it possible to store waiting jobs over time, until an available worker fetches the job. If a manager stops working temporarily, the reports will still be waiting on the job-queue. RabbitMQ can therefore work as the communication-bottleneck between the workers and the Managers.

5.2.5 Docker

Docker is an open-source based project that allows the user to execute code inside closed containers. Docker was released in March 2013, and has since the release date been used more and more in large-scale corporations and projects.[25] Docker is often used to deploy code into production-environments. Docker removes the need to take the hardware specifications into account when deploying new code.[25] Services is executed inside a closed environment when using Docker. This makes it easier for developers to debug code inside Docker-instances, as a result of Docker being hardware-independent.

5.2.6 CouchDB

CouchDB is an HTTP-based NO-SQL database technology. CouchDB saves documents as JSON, and distributes a RESTful API supported by Python. The API supports calls directly using HTTP-GET/POST calls. CouchDB offers a web-GUI, where changes to databases and tables can be done manually directly in the web-browser. When running large-scale infrastructures with CouchDB, it is easy to add another CouchDB instance that automatically will cluster with the other CouchDB instance. CouchDB can easily replicate its data to another database-host by using the web-GUI.

5.3 Infrastructure

During the implementation, the following infrastructure has been used for the research project. The entire system is executed across two OpenStack virtual environments: Alto Cloud at Oslo and Akershus University College of Applied Sciences (HiOA) and the OpenStack cloud at NTNU Gjøvik. The infrastructures consists of the following inventory:
5.3.1 Alto Cloud HiOA

The infrastructure consists of 9 virtual machines running different OpenStack flavors. The technical specifications for each flavor is listed below:

- Medium flavor - 4 GB RAM, 2 VCPUs, 40 GB HDD
- Large flavor - 8 GB RAM, 4 VCPUs, 80 GB HDD
- Extra Large flavor - 16 GB RAM, 8 VCPUs, 160 GB HDD

Below each server is described into more detail. Each server has a dedicated responsibility in the infrastructure.

<table>
<thead>
<tr>
<th>Server-name</th>
<th>OpenStack Flavor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RabbitMQ</td>
<td>Extra Large</td>
<td>Server hosting the RabbitMQ-service</td>
</tr>
<tr>
<td>Interpreter</td>
<td>Large</td>
<td>Server hosting the API</td>
</tr>
<tr>
<td>CouchDB</td>
<td>Extra Large</td>
<td>Server hosting the CouchDB</td>
</tr>
<tr>
<td>Clerk-Manager</td>
<td>Medium</td>
<td>Server acting as the Clerk-Manager</td>
</tr>
<tr>
<td>Leeshore-Manager</td>
<td>Medium</td>
<td>Server acting as the Leeshore-Manager</td>
</tr>
<tr>
<td>Webuse-Manager</td>
<td>Medium</td>
<td>Server acting as the Webuse-Manager</td>
</tr>
<tr>
<td>Httperf-Manager</td>
<td>Medium</td>
<td>Server acting as the Httperf-Manager</td>
</tr>
<tr>
<td>Purser manager</td>
<td>Medium</td>
<td>Server acting as the Purser-Manager</td>
</tr>
<tr>
<td>IMT3441 External</td>
<td>Large</td>
<td>External gateway to NTNU Gøvik</td>
</tr>
</tbody>
</table>

5.3.2 OpenStack Cloud NTNU Gjøvik

The infrastructure at NTNU consists of the workers. Below a detailed list is provided:

<table>
<thead>
<tr>
<th>Server-name</th>
<th>OpenStack Flavor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMT3441-Manager</td>
<td>Extra Large</td>
<td>Clerk, and Leeshore-workers</td>
</tr>
<tr>
<td>Worker 20</td>
<td>Medium</td>
<td>Puppet Master for all workers</td>
</tr>
<tr>
<td>Worker 1-19</td>
<td>Medium</td>
<td>Worker instances</td>
</tr>
<tr>
<td>Worker 21-30</td>
<td>Medium</td>
<td>Worker instances</td>
</tr>
</tbody>
</table>

5.4 Program structure

This section goes into detail on the structure of the application. Each subsection contains information about the different instances, and the classes running on the nodes.

5.4.1 The managers

Each manager has different responsibility-areas. Each of the Managers governs one test, and should make sure that the test is deployed correctly after a given model. The managers are developed using Python 2.7.
General structure

Every manager consists of the same backend logic. Each manager is deployed with the same set of functions. These functions include the following:

- Fetch elements from the CouchDb.
- Write elements to the CouchDb.
- Post elements to RabbitMQ.
- Fetch elements from RabbitMQ

The managers are built using the same data-model. Each manager is set to govern an account (equal to a teacher). An account consists of a semester, course and groups. Each group has a set of members. The group-name is the primary key during runtime. And it is used to identify the owner of the different test-reports. Each managers implements its own way of deploying tests. The sections below will go into detail how this is accomplished on the different managers.

Webuse-manager

The Webuse-manager is responsible for simulating activity on Bookface. Webuse is implemented with four different functions: Add users; add post; add comment; download the front-page. Webuse has a function that makes it possible to randomize what function is executed in runtime. Webuse uses an algorithm to deploy a given number of tests per user. The number of tests deployed per user follows a 24-hour model to simulate traffic through a 24-hour day. Each model follows a dataset with 288 entries, representing every 5-minute period per 24-hour. The datasets are constructed after different activity models for a web site. Every user is configured with an offset, which is used when the number of tests to deploy is calculated. The offset is different for every user. This is to make sure that no user has maximum activity at the same time. (More information about the workload-profile and Webuse can be found in the Design-chapter)

Queue

The Webuse-manager differs between two states: posting new jobs to the job-queue and listening on the report queue. In a separate configuration file, an interval needs to be set. The interval decides how long the Manager should be in listening state for new reports. Before entering listening state, the Manager performs a calculation per user to decide the number of jobs to deploy per user. The jobs are added to a work list, which is posted to the queue. When all generated jobs are added to the queue, the Manager enters listening state. When the time exceeds the interval set, the procedure is repeated.
Reports

The Webuse-manager fetches reports from the report queue. Before a report is written to the database, it is parsed by the Manager. The Manager has the ability to change field-values in the report. When the Manager treats a report, it is sent on to the interpreter to be stored in the database.

Purser manager

The Purser manager is responsible for running uptime-tests against Bookface. The Purser managers deploy one test per user within a given interval. Each job contains parameters needed to execute a Purser-job against a user. After the Purser manager has initiated its configuration, it starts to iterate through the user-list. The manager deploys one Purser-job per user, per iteration. When the Manager has finished job-deployment, it goes into listening state for as long as the interval is set. As the Bookface-database grows bigger and bigger for every user, the longer it takes to execute the Purser-test.

Reward

Each time the Manager fetches a job from the report queue, a reward is calculated. Each user is configured with a set of fields in the user-account profile in CouchDB. These values are used to calculate the reward each user will get per test-execution. If the status of the job is OK, the values used in the calculation include the following:

- Hourly rate.
- Time since last check

For status OK, a bonus and a reward are calculated. First the reward is calculated using the following formula:

\[
\text{Reward}_{OK} = \frac{(\text{Hourly\_rate}_{group} \times \text{Time\_since\_last\_check})}{3600}
\]

The bonus is then calculated using the formula below and added to the reward.

\[
\text{Bonus}_{group} = \frac{\text{Bonus\_time\_cutoff}}{\text{time\_used\_to\_download\_site}} \times \text{Bonus\_value}
\]

For status Partial OK, the following values are used:

- Hourly rate
- Partial OK punishment decrease - A decimal used to determine the new "punishment reward"
- Time since last check
The formula used to calculate reward for status Partial OK is below:

\[ \text{Reward}_{\text{Partial\_OK}} = \left( \frac{\text{Hourly\_rate} \times \text{Time\_since\_last\_check}}{3600} \right) \times \text{Partial\_OK\_punishment\_decrease} \]

If a job has status NOT APPROVED, the reward is subtracted from the user-account balance. When a test gets status NOT APPROVED, it would mean that the Bookface site is down. To calculate the punishment-cost for status NOT APPROVED, the following formula is used:

\[ \text{Cost}_{\text{Not\_approved}} = \left( \frac{\text{Hourly\_rate}_{\text{group}} \times \text{Time\_since\_last\_check}}{3600} \right) \times (-1) \]

**Reports**

When a Purser worker has finished a job. It is sent back on the Purser-report queue. The Purser-report queue contains information about which user the job is performed against, and the what Bookface site that is connected to the specific user. During test-execution, the Purser-script has a checklist to follow, when checking Bookface. Some of these items include:

- Download index.php, along with styles.css and all pictures.
- Measure the time it took to download the site.
- Check if index.php exists in the local directory downloaded from Bookface.
- See if a given word exists in index.php- This is used to determine if the site is really up
A time-stamp is also published in the report. The time-stamp is used by the Manager to calculate the time since the last check. This value is then used to calculate a reward, which is then posted to the user-account balance.

**Httperf-manager**

The Httperf-manager is developed to simulate traffic to the Bookface site. One Httperf-test runs constantly to each Bookface-site for five minutes at a time. When the test is finished, a new one is deployed instantly. The Httperf-manager is responsible for calculating the number of connections that should be deployed to the site. The Httperf-manager uses the same workload-profiles as the web-use managers. The algorithm used in the Webuse-check to determine the number of tests to execute per user, is used to determine the connection rate, and number of connections on the Httperf-manager. (Httperf is described in detail in the background-chapter.)

When the Manager has finished job-deployment. It goes into listening state where it fetches reports from the Httperf-report queue. The Manager then parses each report before it is stored in the report-database.

**Leeshore-manager**

The Leeshore-manager is developed to connect to the OpenStack tenant space to each group. Leeshore then finds a random virtual machine and turns it off. Leeshore executes a new job per group every second hour. To determine the interval between each job-execution, 2 hours is converted to seconds, and divided by the length of the group-list. Only one job is posted to the Leeshore-queue per iteration.

**Clerk-manager**

The Clerk manager is responsible for deploying the Clerk jobs. The Clerk script logs into each group’s tenant-space in OpenStack, and calculates the number of running machines based on OpenStack flavors. Clerk then calculates a price to withdraw from the group-balance, based on the number of running virtual machines.

The Clerk manager is set to deploy new jobs to the work queue every 5 minutes. Every group gets one job deployed per iteration.

### 5.4.2 The workers

**Webuse worker**

The Webuse-script is configured using the programming language Perl. When the web-use script is called the output is printed to the console containing information about the function executed. The output is formatted using colon. In order to be able to read and parse the output into a python script, a separate Webuse-worker has been made. The Webuse-worker is
configured to poll the Webuse work-queue for new jobs. An element on
the queue consists of an executable bash-string. The function call is then
executed by using the Python library Subprocess. When the program exe-
cution has finished. The output is parsed and converted to a Python dic-
tionary and sent back to the Webuse-manager as a report.

Report models

This section contains examples of different reports that the Webuse-worker
can generate after a test has finished executing. Different formatting rules
have been defined for the different functions executed.
Add user

One of the functions that Webuse offers is to add a new user. The perl-script connects to an online name-generator that generates a random name for the user to be added. A profile-picture is then downloaded from avatarsdb.com and added to the user profile. When the user is added, a report is created by the Purser worker. Below, an example report is shown.

Listing 5.1: Webuse new user report

```json
{
    "task": 'new user",
    "group": 'groupA',
    "url": 'http://10.0.0.0/newuser.php?user=julian santana\&image=http://www.avatarsdb.com/avatars/dont_tell_me.jpg',
    "worker": 'd77e27b95489',
    "random algorithm": '10101010',
    "new name": 'julian santana',
    "avatar": 'http://www.avatarsdb.com/avatars/dont_tell_me.jpg'
}
```

Create Post

Bookface supports creating new posts. The Webuse script is shipped with a text-file containing sentences from the William Shakespeare play Hamlet. Each time a new post is made, a random sentence from Hamlet is chosen as the post-content. The post is then added to an existing users profile. When the test has finished execution, a report is generated. The report contains information about which user who created a new post, and what the post contained. Below, an example report is presented.

Listing 5.2: Webuse new post report

```json
{
    "status": '<html>ok</html>',
    "task": 'new post',
    "group": 'groupL',
    "url": 'wget -t 2 -T 5 -o - -q http://10.0.0.0/addpost.php?user=2909\&post=and%20therefore%20as%20a%20stranger%20gives%20it%20welcome.',
    "worker": 'd77e27b95489',
    "random algorithm": '10101010',
    "post by": '2909'
}
```
Add comment

Comments can be added to existing posts. When a new comment is added to a post, a sentence from Hamlet is chosen as reply to the post. The report generated contains what post the comment is added to. A report-example can be seen below.

Listing 5.3: Webuse new comment report

```json
{
    'task': 'new comment',
    'group': 'groupI',
    'url': 'wget -t 2 -T 5 -o /dev/null -q http://10.0.0.0/addcomment.php ?user=1082\&postid=2001\&comment=%20will%20prophesy%20he%20comes%20to%20tell%20me%20of%20the%20players.',
    'worker': '422 af0f13a24',
    'random algorithm': '10101010',
    'message': 'picking postnumber 1'
}
```

Front-page

The last function that Webuse supports is to download the front-page. The index.php file of Bookface is downloaded to simulate traffic on the site. An example report can be seen below.

Listing 5.4: Webuse download front page report

```json
{
    'worker': '0d20a60307c0',
    'task': 'frontpage',
    'group': 'groupL',
    'random algorithm': '10101010'
}
```

Purser worker

The Purser worker is the worker responsible for executing Purser-jobs. The Purser worker is developed differently from the Webuse worker, because the Purser-script is written in Python. The Purser worker imports the Purser script class, and executes a Purser-script call every-time a new job is received. When a job is executed, the Purser worker is responsible for writing a report that the Purser manager later will validate.

An example Purser-report is listed below:

Listing 5.5: Purser report with status OK

```json
{
    'group': 'groupL',
    'File exists': 'True',
    'Http response code': 200,
    'Check timestamp': 1453611987.84,
    'Hostname': '10.0.0.0',
    'worker': '4c736821979d',
    'Time used to download': 20.2922101021,
    'File': 'index.php',
    'Lookup status': 'Word is found in /root/
}
```
When the Purser manager deploys a new job on the work queue, the following are attached:

- IP of Bookface
- Searchword
- Executable filepath to download the Bookface site to.
- Timestamp when the script is added to the queue.

The information is read by the Purser worker and sent as parameters in the Purser-call.

The Purser worker is programmed to handle unexpected exceptions that may arise during test-execution. If the target host is shut-down, the Purser script will raise an IOException stating "No route to host". A report is then written, stating that the job is NOT APPROVED.

**Httpperf worker**

The Httpperf worker is responsible for executing the Httpperf jobs against the Bookface sites. The Httpperf-jobs contains an executable path that the workers use directly. The Httpperf worker executes a bash-script with three arguments:

- IP of Bookface
- Connection rate
- The total number of connections to generate

When the job has finished execution. The output is parsed by the worker, converted to a dictionary and sent back to the Manager. A sample report can be seen below:

```
Listing 5.6: Httpperf report

```

```
{
  'errors': ' fd-unavail 0 addrunavai 0 ftab-full 0 other 0',
  'group': 'groupT',
  'reply time [ms]': ' response 546.8 transfer 1080.8',
  'worker': '7ec2712076ac',
  'connection time [ms]': ' connect 223.1',
  'net i/o': ' 17440.6 kb/s (142.9*10^6 bps)',
  'connection rate': ' 6.9 conn/s (144.4 ms/conn, <=22 concurrent connections)',
  'request size [b]': ' 65.0',
  'connection length [replies/conn]': ' 1.000',
  'maximum connect burst length': ' 1',
  'reply rate [replies/s]': ' min 5.6 avg 7.0 max 8.2 stddev 0.6 (60 samples)',
  'reply size [b]': ' header 179.0 content 2578676.0 footer 2.0 (total 2578857.0)',
  'message': 'httplib --client=0/1 --server=10.0.0.0 --port=80
--uri=/ --rate=7 --send-buffer=4096 --recv-buffer=16384 --num-conns=2100 --num-calls=1',
  'total': ' connections 2100 requests 2100 replies 2100 test-duration 303.246 s',
  'cpu time [s]': ' user 21.21 system 273.09 (user 7.0% system 90.1% total 97.0%)',
  'reply status': ' 1xx=0 2xx=2100 3xx=0 4xx=0 5xx=0',
  'request rate': ' 6.9 req/s (144.4 ms/req)'}
```

```
Leeshore worker

The Leeshore worker is programmed to execute the Leeshore-script. The Leeshore worker listens to the Leeshore work-queue, and fetches the job when a new job arrives. The content received from the work queue is an executable string that points to a location where the Leeshore-script is located in the Docker instance. The Leeshore-script is developed to output its result in JSON. The output is parsed by the Leeshore worker, and sent back to the Leeshore-manager. An example Leeshore-report can be seen on the next page:
Clerk worker

The Clerk worker is designed to execute jobs received from the Clerk manager. Each job contains an executable string pointing to the location where the Clerk script is located. When a job has finished executing, the Clerk script is designed to output its contents as JSON. The worker parses the output, and sends it back to the Clerk manager. A sample report can be seen below.

5.4.3 Database Layout

CouchDB is used as the database-technology during the project phase. (More details about CouchDB can be found at the beginning of the implementation chapter.) CouchDB refers to table content as documents. Each document contains key, value pairs. Each key, value pair can be queried using JavaScript functions.

5.4.4 Querying documents

CouchDB is HTTP-based, and offers a web-based GUI where queries can be defined in the browser. A query is often called a view. A view is generated
from a predefined Javascript query, and can be called directly using REST-calls later. When operating with large-scale databases, two different query functions can be defined:

- Map function
- Reduce function

When a map function is executed, it has the ability to cache data to save performance in the next query. The reduce function is used to optimize searches and save performance when big queries is defined.

5.4.5 Temporary views

CouchDB supports temporary views. When data needs to be retrieved once, a temporary view can be used. A temporary view is a predefined query to stored in the CouchDB instance. Python supports attaching a temporary view in a query. Below an example temporary view is provided.

Listing 5.9: CouchDB query

```java
//This function fetches the document for group=groupA
function(doc) {
    if(doc.group=="groupA")
        emit(doc.group, doc);
}
```

Account structure

An account-document consists of several tags needed by the Manager to create jobs. An account-document example is listed below.

Listing 5.10: User account object example

```json
{
    "_id": "123456789101919231",
    "_rev": "16151-bcc3fb7ecfad429c15e67d77e478da45",
    "members": [
        "Ole Jensen",
        "Stian Anderssen"
    ],
    "bonus": 1,
    "tenant_name": "OpenStack_groupF",
    "last_check": 1455544751,
    "course": "IMT3441",
    "semester": "V16",
    "file": "index.php",
    "offset": 98,
    "Balance": -7.776514880744596,
    "ipaddress": "10.0.0.0",
    "teacher": "Kyrre",
    "hourly_rate": 3,
    "Leeshore_enabled": 1,
    "group": "groupT",
    "filepath": "/root/uptime_challenge_master/worker/",
    "bonus_time_cutoff": 5,
    "partial_ok_punishment_decrease": 0.1,
    "Sentence": "Users:",
    "configfile": "transsine.dat",
    "enabled": 1
}
```
5.4.6 Interaction

The infrastructure as a whole, has several dependencies that needs to operate together to work properly. To enable instances to communicate, several communication mechanisms have been developed. (More details about the communication layers can be found in the Design-chapter) This section will go into detail on API and RabbitMQ functionality that has been developed. Several code examples will be shown to demonstrate how the functionality is achieved.

Interpreter API

The communication between the managers and the API, is developed using an API called Pyro4. Pyro4 is a Python library released in 2010. Pyro4 is developed specifically for Python API calls, and is supported on different Python platforms. (Python 2 and 3, IronPython, Pypy). Pyro4 supports all Python data-types, and allows remote procedure calls to be executed using correct data-types. Pyro4 supports different serializers (JSON, serpent, pickle and marshal)[31]

The Interpreter API is developed to run constantly while the Managers are running. To be able to let the Managers execute properly, the API must be up and running.

Change documents in the database

When the Managers needs to update a given value in the database, it needs to make a function call to the API containing the following:

- Database server-name
- Database name
- Key to search for
- Value to search for
- Key to update
- Value to update

This function is implemented to be a general function that can be used across different databases and values. The code can be seen below:

```
Listing 5.11: Function to modify a value in CouchDB

def modify_key(self, dbservername, dbname, key, value, keytoupdate, valuetoupdate):
    """
    This function updates any given key, value in the database. A unique key, value is passed as arguments to the function. This is necessary to get the correct document back from the database. The document is then updated with a given key, value.
    :param dbservername:
    :type dbservername:
    :param dbname:
```
Get Account Object

During startup, the managers need to fetch the user-accounts. A function has been created in the API with ability to fetch all enabled accounts. The account object contains meta-information that the Managers use to create jobs for the workers. Before a new job is deployed, the account object for the given user is fetched from the database.

The code to fetch the account object can be seen below.

Listing 5.12: Return all enabled accounts

```python
def getEnabledUsers(self):
    '''
    This function connects to the CouchDB and fetches all enabled accounts. The account objects are returned in a list.
    '''
    couch = couchdb.Server("http://USERNAME:PASSWORD@couchdb:DBPORT/")
    db = couch["accounts"]
    map_fun = "'''
    function(doc) {
        if(doc.enabled==1)
            emit(null, doc);
    }
    ''''
    result = db.query(map_fun)
    returnvalues = []
    for element in result:
        returnvalues.append(element["value"])
    return returnvalues
```

5.4.7 RabbitMQ

The managers and workers communicate using RabbitMQ. The managers connect to the workers over the work queue. While the communication
back to the Managers is sent through the report queue. By using RabbitMQ, the managers and workers can communicate asynchronously. This section will contain two code examples:

- Posting elements to a queue
- Fetching elements from a queue

**Posting elements from a queue**

When the Managers want to post elements to the work queue, it uses the following function:

```python
    def createWorkQ(self, queuename, job_list):
        
        ""
        This function post a job to the work-queue
        :param queuename: 
        :type String: 
        :param job list: 
        :type list: 
        :return: 
        :rtype: 
        ""
        queue = Queues()
        for group, job in job_list.iteritems():
            for j in job:
                jobdict = {}
                jobdict.update((group: j))
                queue.createQueue(queuename, jobdict)
```

A queue is created, and an element containing a job list is attached. The job list contains different jobs that the worker needs to execute. Along with the work-list, the group-name is attached, in order for the workers to determine which groups the job belongs to.

**Fetching elements from the queue**

The workers and managers use the same function to fetch elements from the queues. RabbitMQ implements two different functions to retrieve elements. The first function is developed to constantly listen to a queue, and parse elements as they are received. The second function is developed to use a type of busy-waiting that enables the managers to inspect the queue for a short period of time before it reconnects. This methodology is used to let the Manager exit listening-state more easily. The code can be seen below:

```python
    def receiveOneMessageFromQ(self, queuename, interval):
        
        ""
        Recursive listen-method
        that is used to continuously
        listen to the report queue
        within the execution interval
        :param queuename: 
        :type queuename: 
        :param timevalue: 
```

80
```python
timestart = time.time()
while (time.time()-timestart) <= float(interval):
    stringValue = ""
    connection = self.connectToRabbitMQ()
    channel = connection.channel()
    channel.queue_declare(queue=queuename)
    try:
        method_frame, header_frame, body = channel.basic_get(queue=queuename)
        if method_frame.NAME == 'Basic.GetEmpty':
            connection.close()
        else:
            basic_ack(delivery_tag=method_frame.delivery_tag)
            connection.close()
            report = Reports()
            report.buildReport(body)
    except AttributeError:
        logging.critical("Waiting for answer... time used: "+
                          str((time.time()-timestart)) + " interval: " + str(interval))
        time.sleep(1)
    connection.close()

5.4.8 Workers
The tests developed in the project period are written using different
programming languages. Some of the tests are written using Perl. Some
output is printed as JSON, and can be parsed directly. For the Httpperf test,
and Webuse-tests, different parsers have been implemented. The parsers
read the output from the tests, and converts the output to JSON before it
is sent back to the Managers. Below the parser used to read the web-use
output is provided.

Listing 5.15: Webuse output parser
```
statusstring.replace("\n","")

elif "ok\t<html>" in templist:
    statusstring="".join(templist[templist.index("<html>")+1])
    del[templist[templist.index("<html>")+1]]
    templist.insert(templist.index("ok\t<html>")+1])
    statusmessage= statusstring.replace("\n","")
    for i in templist:
        content = i.replace("\n", "")
        if content != "";
            list.append(content)
    for i in list:
        j = i.split(":")
        if len(j) == 1:
            reportdict.update({"message": j[0]})
        elif (len(j) >= 2):
            reportdict.update({j[0]: ".join(j[1: len(j)])")
            reportdict.update({"status": statusmessage})
        return reportdict

The Clerk, and Leeshore test outputs the data as JSON. The worker uses a built-in Python function to evaluate the JSON output before it is sent to the managers.

5.4.9 Docker images

To deploy workers, Docker is used as the container-technology. Before the workers are spawned, a Docker image needs to be built containing the necessary files and libraries in order to execute the worker scripts, and tests. This section will go into detail how the Dockers images are compiled, and how they are used to deploy Docker instances.

Dockerfile

In order to be able to build Docker-images, a Dockerfile must be created. A Dockerfile is an instruction file with commands that is used to configure the Docker image. A Dockerfile use a set of commands to install and copy files needed to build an image, some of these commands include:

- **RUN** - Commands to be executed during image build.
- **FROM** - specifies which image to build from.
- **MAINTAINER** - The owner and maintainer of the Dockerfile
- **ADD** - Add attached files
- **EXPOSE** - Expose a given port
- **ENV** - Can be used to set environmental paths
- **CMD** - Commands to execute upon docker-instance deployment
Below, an example Dockerfile is provided. The Dockerfile below downloads the Ubuntu 14:04 Docker-image, and installs the necessary packages to compile a Docker-image for the Httperf worker. A connection to Github is performed to download the code needed for the worker to execute the Httperf worker-script.

Listing 5.16: A Dockerfile example

```
FROM ubuntu:14.04
MAINTAINER Stian Strom Anderssen <s177437@student.hioa.no>
RUN apt-get update && apt-get install -y openssh-server python-pip git git-core httperf libwww-perl
RUN pip install pika==0.9.8
RUN mkdir -p /root/.ssh/
ADD id_rsa /root/.ssh/id_rsa
RUN chmod 700 /root/.ssh/id_rsa
RUN echo "Host github.com
  StrictHostKeyChecking no"
>> /root/.ssh/config
RUN cd /root/ && git clone git@github.com:s177437/uptime_challenge_master.git
EXPOSE 22
CMD ["/bin/bash", "/opt/startup.sh"]
```

In order to be able to build custom images, one must build a new image on top of an existing image. All existing Docker-images can be used as a reference-image when compiling new images.

**Dockerhub**

Webuse, Purser and Httperf gets their images compiled on Dockerhub. Dockerhub is Dockers official image platform. On Dockerhub, one can choose to compile one private Docker-image for free, or as many public ones as the user may need. The public images can be downloaded, and used by anyone.

**Triggered builds**

Github is used as the revision control platform during development. Github operates with different branches, where the user can push different code to. Dockerhub can be configured to trigger an image build if code is pushed to a certain git-branch. Every time code is pushed to the master-branch on Github, a trigger is configured to start a new image-build on Dockerhub for Purser, Webuse and Httperf. When a build is triggered, Dockerhub needs to pull down a Dockerfile from the Github branch, in order to be able to create the build.
5.4.10 Deployment map

Figure 5.1: The figure shows the stages from code-push to a git-branch into Docker deployment.

5.4.11 Puppet

Puppet is used as configuration management tool when Docker instances are deployed. Every virtual machine that run the worker-instances are configured to pair with the same Puppet master. The Puppet master is responsible for distributing manifests to the Puppet-agents, and to make sure that the correct Docker-images is downloaded and deployed. Each virtual machine is configured to make sure that a given number of Docker-instances is running.

Puppet Manifests

Puppet has created a library that opens up to administer Docker deployment with Puppet. This has been used to deploy Docker instances for the project. A sample Puppet manifest can be found below.
The Puppet manifest above downloads an image for the Purser worker, and deploys two instances from the image. The next time the Puppet agent runs, it will redeploy the purserworker-instances, if they have stopped or died.

5.4.12 Logging

The managers and the interpreter are configured to send output to a local logfile. This log can later be used to debug what happened if the manager/API service stops working. To achieve logging, the Python library Logging has been used.

Log severity

The Python Logging library allows to tag log-entries with a severity. The categories available are:

- Emergency
- Alert
- Critical
- Error
- Warning
- Notice
- Informational
- Debug

When the Manager-service is executing. The log-output will look like the following:

<table>
<thead>
<tr>
<th>Listing 5.18: Manager log output</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFO:root:Waiting for answer... time used: 3.86491799355 interval: 300</td>
</tr>
<tr>
<td>INFO:root:Waiting for answer... time used: 5.23730015755 interval: 300</td>
</tr>
<tr>
<td>INFO:root:Waiting for answer... time used: 6.25063300133 interval: 300</td>
</tr>
</tbody>
</table>
The log output above shows an example from the Webuse manager in listening state. The log output is printed once per loop iteration too let the user keep track on how long it is until new jobs are deployed.

The interpreter is configured to log all reports written to the database, and a status saying if the report was written to the database successfully. An example can be found on the next page:
The log output above shows four different reports written to the database for different groups. To be able to monitor live the status of each test-execution, the entire report is printed in the log.
Chapter 6

Analysis

This chapter aims to analyse the prototype, and experiences made from the execution-period of the system. The analysis will outline positive aspects and problems that arose during execution of the prototype developed. The chapter will go into results found from using the prototype in a course at NTNU Gjøvik, and look at the results. The last section of the chapter will analyse statistics gathered from reports generated by the prototype for different groups. The analysis-chapter will outline the main topics that will be discussed in the discussion-chapter. The analysis-chapter focuses on the deployment phase and execution phase. Information about the implementation-phase can be read in the implementation-chapter. The analysis-chapter is a continuation of the implementation-chapter, and will assume that the reader use the implementation-chapter as reference for more information about the technologies mentioned in the analysis.

6.1 Test case

A technical requirement before the implementation started was to have a running prototype by the end of January 2016. The goal was to be able to test the prototype in a course at NTNU Gjøvik. The course is called Database and application administration (coursecode IMT3441) and is taught by Kyrre Begnum. The course is a bachelor course and aims to teach students about database and web-server installations. During the course the students delivers three projects that needs to be evaluated. Bookface is used as "proof-of-concept" service that the students need to install. The students are divided into groups, and are granted a tenant in an OpenStack virtual cloud at NTNU Gjøvik. Each tenant is granted a quota of virtual servers to deploy. As the course progresses, the students infrastructures become more complex. The course teaches the students different technologies and tools that may come useful when working with infrastructures.

The prototype developed is optimised for the course, and customised tests have been written to help evaluate the student’s progression.
6.1.1 Infrastructure configuration

The prototype consists of two infrastructures:

- Alto Cloud - HiOA
- OpenStack Cloud - NTNU Gjøvik

RabbitMQ is configured as communication channel between the infrastructures. All traffic between the infrastructures are protected with username and password.

Alto Cloud

The main infrastructure is running in the Alto Cloud. Each manager running in Alto Cloud governs one test, and has to make sure that new jobs is deployed in addition to parsing reports from the workers. Each managers is configured with a service responsible for governing the manager-script. RabbitMQ is set up on a separate instance, and is exposed for the Internet with a public IP-address. The interpreter-API is running on a separate instance, and has to be running to open up a communication channel to the database from the managers. The interpreter and RabbitMQ are the main bottlenecks in the Alto-cloud infrastructure. If the API stops working, the managers will stop distributing new jobs. The managers rely heavily on the RabbitMQ service to work as well. RabbitMQ is the only communication channel to NTNU Gjøvik and has to be up and running to make sure that jobs are deployed against the Bookface-sites.

NTNU Gjøvik Cloud

All the workers are configured to run in the cloud at NTNU Gjøvik. Each Bookface-site is running in a local network and the workers need access to the local network to execute evaluation tests against the sites. 31 virtual machines are configured deploy a given set of Docker-instances per machine. The Docker-deployment is automated and relies on Dockerhub to fetch the newest version of Docker-images. Puppet is used to deploy the Docker-instances on the virtual machines. Scaling of Docker instances must be achieved by adding new instances to the Puppet-manifests. Puppet will then apply the new manifests on the virtual machines. Puppet restricts the use of automatic redeployment of Docker-instances if the Docker-images are recompiled in Dockerhub. To update the Docker instances on the worker-VMs, the images has to be deleted locally before Puppet downloads the new images from Dockerhub.

6.2 Methodologies to evaluate value

The Purser check and Clerk check are responsible for updating the balance of each group. The balance is constantly updated by the managers, and allows for the students to keep track of their current progress and compare
it to the other groups. The balance is contributing to give a valuable live feedback to the groups. A decrease in the balance could mean that the Bookface site is down. If the balance increases slowly it could mean that the connection to the database is down, resulting in "Partial OK" from the Purser check. If students are running big infrastructures with big virtual machines running on high performance, the cost that the Clerk check calculates will be high. To achieve a total value-perspective for the progress of a group, one need to look at results from different tests separately, and analyse the results in a time-perspective. The sections below will give some examples how this can be achieved.

**Workload-profile**

The workload-profile is the dataset file, which is used to calculate the strength of Httperf and the number of Webuse-tests to deploy per user per deployment. The workload-profiles contains 288 lines with a combination of digits (More information about workload-profiles can be found in the design-chapter). There are created several workload-profiles that can be used to simulate different traffic-activity for a site. Some of the profiles are meant to simulate low activity on the site, while others are meant to stress test sites by deploying a big number of jobs per deployment. To look at value from a workload-profile perspective, one can analyse the combination between Purser-checks, Webuse-checks, and Httperf. If Webuse and Httperf tests are deployed heavily over a short period of time on a group site, it would mean that the Purser check uses longer time to download the Bookface site, due to heavy activity. This would increase the site-latency (response-time), which again affects the end-user performance. If the Bookface downloads fast while being exposed for stress testing, it would imply that the end-user experience still would be satisfactory since the site is responding quickly.

Purser also checks the HTTP-response code of Bookface. While being exposed for stress by Webuse and Httperf, the site could temporarily stop responding to new requests. (HTTP-responce code 500,503). The site is then unusable, and something must be done to get the site up and running again.

**Leeshore**

The Leeshore-test aims to turn off a virtual machine for an OpenStack group. Leeshore connects to the group’s virtual environment and picks a random server to turn off. The server is then stored in the generated report. If the students are running small infrastructures, it could stop Bookface from being accessible. By looking at the Leeshore-results, along with the Purser results. One can see if Bookface is still running even if one server is taken down, to help evaluate students dedication to make sure the site is running at all times.
6.3 Test deployment data

This section will go into detail on data gathered during the project. Some of the graphs will go into detail on deployment history over time for a given test. Data presented in this section is gathered over longer time-periods.

Webuse deployment

The Webuse-test is configured to calculate the number of tests to deploy from a dataset-file. This file contains a set of parameters to help the managers calculate the correct number of tests per group. The calculated numbers of tests are then sent out to the workers. When the workers have finished executing a job, a report is saved in the database. The results can later be analysed and looked at.

Each Webuse-report contains a timestamp to show when the test was executed. A query was executed against the database to retrieve Webuse-reports over a three day interval for a group. (February 13, 2016 - February 16, 2016) The goal was to calculate the maximum, median and minimum number of deployed test over five-minute intervals per hour. An algorithm was made, counting the number of reports per five-minute intervals. Each five minute interval number was then added to a list containing 12 entries (12 per hour). From the entries, the maximum, minimum and median value were calculated.
The values were added to a Python dictionary containing correlating timestamp for the hours. The procedure was repeated for every hour within the calculation interval. The end-result was a big Python dictionary containing the maximum, minimum and median value of the number of Webuse test-deployments for a group in an hour interval. From the dictionary, an errorbar graph was made. The graph can be seen below.

![Graph showing number of deployed Webuse-tests per hour](image)

**Figure 6.1**: A graph presenting maximum, minimum and median value of Webuse-test deployment per hour in a three day interval

**Commonalities**

When looking at the Webuse-deployment for a group over a three day interval. One can see a repeating pattern when the number of deployed checks maximizes. One can see from the graph that some of the errorbars differs a lot between minimum, median and maximum. This corresponds to the use of noise in the workload-profiles. When the noise is a big number. The deviation will increase in size. This makes it possible to generate larger numbers, causing the big gap on the errorbar. The simulation above illustrates three days of Webuse activity on the Bookface site. The results from the graph above can be used in close relation to a web-site’s actual visits during a 24 hour period. Normally a web site would peak at a given point during the day where the number of visits increases. The workload-
profile used to generate the graph above simulates a web-site with 2-35 new visitors every fifth minute. The result is in close relation to what a real web site could have.

6.4 Uptime and Balance relation

To help evaluate the uptime over a longer period for a group, two scripts were implemented. The scripts show different reports-analysis for the same groups in the same time-interval. The last section compare the results found from both graphs, and outline how they can be used to evaluate the groups progression in uptime.

Leeshore and Purser relation

The first script queries all Leeshore and Purser-reports for a given group. Each report is parsed and the report-timestamp, along with the report is added to a Python dictionary. Each timestamp is rounded up or down to the closest two-hour interval. (7200 second intervals) The dictionary is then iterated in sorted order. If the report originates from Purser, the status is checked. The site is reported as up for status OK, and reported as down for status Not Approved and Partial OK. Every Purser-report within a given interval gets a tag with "site is up" or "site is down." The tags are then counted for each interval. The "site is up" count is then divided on the total number of Purser-reports within the two-hour interval, to get an uptime-percentage for the given interval. If it exists a Leeshore-report within the interval, 1 is added to the count-list to indicate that Leeshore has executed. The end-result is a timestamp-interval based dictionary that contains corresponding lists with the uptime-percentage and a number stating if Leeshore has executed within the interval-period. The dictionary is presented as an errorbar below. The intersections between the markers, and the errorbar represents that Leeshore has executed in the given interval. Each marker represents a two-hour interval. The graph presented below is taken from an 8-day interval. (February 8, 2016-February 16, 2016)
Figure 6.2: A graph showing the uptime percentage for group. Every intersection indicates that Leeshore has executed. If there are no intersection, it would mean that Leeshore has not executed during this interval.

Balance

The second script made in the analysis outlines the balance over time for a group. A query was performed for a given group to fetch all Purser, and Clerk reports. The reports were iterated in sorted order based on timestamp. Separate calculation-mechanisms were implemented for Purser and Clerk. For each report a price/reward was calculated, this was then added/subtracted from the total balance. For each report, the new balance was added to a dictionary with the report-timestamp as a key, and the balance as value. The dictionary was then presented in a plot, outlining the same time-interval as the Leeshore-Purser graph. The graph can be seen below.
Commonalities

By looking at the Leeshore-graph and the balance-graph from the section above. Several interpretations can be made. There is a time-period (13 -14 February 2016) where Leeshore did not execute for the given group. On Friday 12 February 2016. There was a lecture held at NTNU Gjøvik in the course that uses the prototype. During the lecture, the students got new assignments to solve for their Bookface-project. Leeshore does not execute on Fridays, because the course lectures is being held on Fridays. The students have enough to focus on in this time-period. One can see this in the balance-graph as well. In the same time-period, the balance increases drastically compared to the rest of the period. This correlates with the Leeshore-check being turned off.

One methodology to use when evaluating the group-effort during the project is to look at the uptime-percentage for the hour after Leeshore has executed. If the uptime-percentage increases, it means that the site was down before Leeshore executed. If the uptime-percentage remains the same, it would mean that the students did one out of two things:

- Used a load-balancer to run a multi-site Bookface
• Got notified instantly when the Bookface-site stopped working and reacted.

Both approaches are commonly used when hosting web sites. When running a loadbalanced environment, one removes the instant need for a person to get the site up and running. The site is already running on another instance. The other approach requires the students to pay more attention to their site to see if something goes wrong. The last approach requires fewer virtual machines and services, but more manual labour. The first approach is the best approach, but the most costly. When running a multi-site infrastructure. The Clerk check will charge more since there are more virtual machines needed to host the site.

6.5 Workload-profiles

To determine the number of Webuse-tests to deploy per user, a calculation algorithm is used. (More information about the algorithm can be found in the Design-chapter). The algorithm uses a property-file containing a combination of numbers and digits. The property-file contains 288 lines, which is meant to represent every 5-minute interval per 24-hour. Before deployment, every group is assigned an offset to help the Manager determine which line in the property file, to start calculations at. Once the line-number is decided, the number of tests to deploy is calculated. By using offsets, every user has different peaks in number of tests deployed. The workload-profiles are meant to simulate a normal day of traffic for a site, by generating more tests at certain times of the day. In order to save performance of the workers, the peak is different for every group. By looking at statistics for a group. One can see that the number of tests deployed over a 2-day period follows the same model, and peaks on the same time every day. The peaks represent times when traffic on the site increases drastically. Below a graph containing simulation from three groups is provided. The algorithm was executed to generate a number of tests to deploy for three groups. The groups had the following offsets:

• GroupA - Offset 0
• GroupB - Offset 70
• GroupC - Offset 140
In the graph one can see that each group follows the same deployment-pattern, but they peak differently. The graph is represented as an errorbar. Each marker on the graph represents an hour-long interval. Inside each interval, 12 measurements was made (Every fifth minute). From the 12 measurements, maximum, minimum and median were calculated. The markers represent the median of deployed tests per hourly interval. The bars pointing up from the marker represents the maximum deployed tests, and the bars pointing down from the marker represents the minimum amount of deployed tests.

![Webuse Deployment Simulation](image)

**Figure 6.4: An errorbar representing Webuse-deployment simulation for three groups**

### 6.6 Httpperf connection rate modelling

To be able to calculate the total Httpperf rate bandwidth over time for all groups, different simulations were performed with different workload-profiles. The simulations below are conducted for 15 groups with different offset for each group, with intention to simulate the total strength the Webuse and Httpperf managers have been dealing with during the course. The offset has been used by the managers to make sure that every group starts at different time of day. When calculating the sum from the base-
number for every group, one can see that the sum is almost equal for every calculation. When calculating the sum for the generated number, the sum will vary more. This section contains graphs showing the sum of generated number for all groups. The graphs represent a five day long simulation period for each workload-profile used in the project. The graph data is presented as the Httperf connection rate over time. The data gathered from this simulation can also be used directly to project the number of deployed Webuse-tests for all groups over a longer period. The graphs below is in close relation to the section covering workload-profiles in the Design-chapter. The graphs outlines a more practical approach for a system administrator to follow, when deciding the total number of workers a manager must govern. (Capacity, refer to the workload-profile section in the Design-chapter for more information about capacity)

![Httperf rate pr hour](image)

Figure 6.5: The sum of the base-number and calculated number used to deploy Webuse-tests, and Httperf connection rate from the file transssine.dat
Figure 6.6: The sum of the base-number and calculated number used to deploy Webuse-tests, and Httperf connection rate from the file transsine2.dat
Figure 6.7: The sum of the base-number and calculated number used to deploy Webuse-tests, and Httperf connection rate from the file lowtranssine.dat
Calculating the number of workers from the workload-profile

To be able to determine the number of Webuse and Httperf workers, one can use the total connection rate from Httperf and the total number of Webuse-tests deployed per deployment. Both test-deployments use the same deployment algorithm with workload-profile dataset files. During development, Httperf is configured to execute for 300 seconds, using the generated rate as connection rate-step.

The Webuse-test has four possible outcomes that decides the execution time-usage. Webuse expands the Bookface-database by adding new users, posts and comments that will eventually cause Bookface to react slower.

To simulate real traffic, one can vary which workload-profile to use for different users. To be able to change different workload-profiles, the number of workers must be scaled up or down based on the total number of generated connections. A change from the workload-profile "transsine.dat" that generates approximately 100 Webuse-tests for 15 groups per deployment to the workload-profile "transsine2.dat" with ability to generate over a 1000 tests under the same conditions would require massive up-scaling of worker-instances. Httperf would require a lot more performance to be able to execute with the new generated rates from
"transsine2.dat". The graphs presented over can give a better overview, on the number of workers the different tests requires based on the workload-profile dataset-files.

### 6.7 Uptime evaluation

One of the main criteria to evaluate when looking at the progression of a group is the uptime-percentage. Earlier sections of the analysis-chapter referred to the Bookface as up, if a Purser-report had test-status "OK". When evaluating the total uptime-percentage during the project period, status "Partial OK" is considered as up, along with status "OK". This is done to give the students credits for at least having a HTTP-response code telling the user that something is wrong instead of not responding to the incoming HTTP-request.

In order to evaluate the uptime-percentage over time for different groups, all Purser-reports were queried from the database for each group. The balance for each report were stored and multiplied with 100 to make it easier to illustrate in a one-percent distribution-plot. The distribution-plot divides the total reward/cost interval into a 100 one-percent intervals. Each interval contains the number of balance-occurrences within the interval-delimiters. The uptime-percentage can be seen by looking at the positive interval-delimiters.

Below, two distribution plots are shown for two different groups. One of the groups has been showing good progression during the uptime challenge, and the uptime-percentage is therefore high. The other group has a lower uptime-percentage, meaning that they have not paid enough attention to their site being taken down by the Leeshore-test. If a group doesn’t check continuously if their site is up and working, they might risk that their site might be down for a longer period of time.

### Graphs

This section will contain an analysis of two different graphs presenting the uptime-percentage for two different groups through a longer period in the uptime challenge. (10 January 2016- 26 March 2016) Each graph will contain a longer explanation below that may be necessary to read, in order to properly understand the graph.
6.7.1 GroupE

Figure 6.9: GroupE one-percent distribution plot over all rewards/costs from Purser and Clerk

The graph over presents the reward/cost distribution plot to groupE. The balance-delimiters range from -120 to 4340. The big range represents the span in what has been rewarded through the uptime challenge for groupE. To calculate the real balance-range, one must divide the delimiters by 100. Meaning that the actual reward/cost range is -1.2 to 43.4.

The biggest calculated reward for groupE is located in the range [43.4>. To illustrate an example where the reward is 44.5. One has to figure out what is bonus and what is reward. The report for the example can be found below:

Listing 6.1: GroupE Purser report

```python
{'group': 'groupE',
 'File exists': 'True',
 'Http response code': 200,
 'Check timestamp': 1456309828.52,
 'Hostname': '10.0.0.0',
 'worker': 'f39c6929ab491',
 'Time used to download': 0.00960493087769,
 'File': 'index.php',
 'Test status': 'OK',
 'Lookup status': 'Word is found in /root/uptime_challenge_master/worker/10.0.0.0/index.php at line 36'}
```

The report over was used to calculate the top reward for the distribution plot for groupE. When the reward is calculated by the Purser manager, a query is performed to the database to determine the time-length since the last Purser check was performed. This value is stored temporarily in the
account profile, and is updated every time a new Purser-report is written to the database for a given group. This value must therefore be simulated to be able to calculate a reward + bonus. The time-length since last download is given as a random number between 10 and 2000 to illustrate the actual time-length interval that the system has provided earlier. When this value is set, it is possible to analyse the report outcome into more detail. This section provides a detailed look at how a reward of 44.5 can be calculated and what the different variables in the calculation equals:

### 6.7.2 Reward and bonus calculation groupE

For status "OK" the Purser manager use the following formula to calculate a reward:

\[
Reward_{\text{report}} = \frac{(\text{Hourly rate}_{\text{group}} \times \text{Time since last check})}{3600}
\]

Below is a table containing the variable-values for this calculation.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hourly rate</td>
<td>3.0</td>
</tr>
<tr>
<td>Time since last check</td>
<td>1400</td>
</tr>
<tr>
<td>Calculated reward</td>
<td>1.167</td>
</tr>
</tbody>
</table>

For status "OK" a bonus must also be added to the reward to give credits to the group that the site is working properly. To calculate bonus the following formula is used:

\[
Bonus_{\text{group}} = \frac{\text{Bonus time cutoff}}{\text{Time used to download site}} \times \text{Bonus value}
\]

To calculate the given bonus for the report. The following values were used:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bonus time cutoff</td>
<td>5</td>
</tr>
<tr>
<td>Time used to download bookface frontpage</td>
<td>0.0096</td>
</tr>
<tr>
<td>Bonus value</td>
<td>1</td>
</tr>
<tr>
<td>Calculated bonus</td>
<td>43.38</td>
</tr>
</tbody>
</table>

When bonus and reward is calculated, one can calculate the final reward for the report which is:

\[
43.38 + 1.167 = 44.547
\]

The calculation provided above shows the role bonus can play if the download-time is small. As the Bookface-site grows bigger over time, due
to more user being added, the download-time will decrease accordingly. Most of the gathered data for groupE is located between 0 and 200 which means a total reward between 0 and 2. This reward means that the site is not working properly (HTTP response-code 500 or 503) or the site is slow to download. Normally this could be due to the fact the Httperf and Webuse-tests are running continuously against the site to simulate activity.

When looking more closely at the graph, it is possible to calculate the uptime-percentage from the one-percent distribution dataset. Part of the distribution spreadsheet is provided below:

<table>
<thead>
<tr>
<th>1% categories</th>
<th>Frequency</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>-120</td>
<td>122</td>
<td>1.23</td>
</tr>
<tr>
<td>-74</td>
<td>104</td>
<td>2.28</td>
</tr>
<tr>
<td>-27</td>
<td>134</td>
<td>3.64</td>
</tr>
<tr>
<td>19</td>
<td>1489</td>
<td>18.67</td>
</tr>
<tr>
<td>66</td>
<td>2461</td>
<td>43.53</td>
</tr>
<tr>
<td>112</td>
<td>2438</td>
<td>68.14</td>
</tr>
<tr>
<td>159</td>
<td>2585</td>
<td>94.05</td>
</tr>
<tr>
<td>205</td>
<td>450</td>
<td>98.60</td>
</tr>
<tr>
<td>251</td>
<td>12</td>
<td>98.71</td>
</tr>
<tr>
<td>298</td>
<td>6</td>
<td>98.77</td>
</tr>
</tbody>
</table>

From the table above, one can see that the probability of ending up in the interval \([-27,-19]\) is 3.64 % which can be used used to calculate the uptime percentage for the group.

\[
100 - 3.64 = 96.36
\]

The calculation above states that the probability is 96.36 % for getting a response, when sending a request to the groupE Bookface, this can therefore be used as the total uptime-percentage for groupE.

### 6.7.3 GroupC

GroupC is the other example in this section. GroupC is one of the groups that had a lower uptime-percentage compared to other groups. The graph below outlines the one-percent reward/cost distribution for the same time-period as groupE.
From the distribution plot for groupC, one can see that most of the rewards and costs are located in the range \([-1.49,2.37]\). (The Category delimiter is divided by 100 to get the actual cost) This result would indicate that the Purser-status often is "Partial OK" when the Purser check has executed. When partial OK is returned, it means that the site is not working properly, and an error-response is sent back to the users. For status "Partial OK", there is no bonus given, just a small reward calculated after the following formula:

\[
\text{Reward}_{\text{Partial_OK}} = \frac{(\text{Hourly\_rate} \times \text{Time\_since\_last\_check})}{3600} \times \text{Partial\_OK\_punishment\_decrease}
\]

To illustrate an example how the reward is calculated, a report for groupC was fetched from the database. The next section will go into detail on this.
**GroupC Partial OK status example**

A report with status "Partial OK" was fetched from the database to illustrate how the reward correlates with different values on the variables on the calculation formula. The report is attached below.

**Listing 6.2: GroupC Purser report**

```json
{
    "group": "groupC",
    "File exists": "True",
    "Http response code": 503,
    "Check timestamp": 1458312933.8,
    "Hostname": "10.0.0.0",
    "worker": "85228c5d96ef",
    "Time used to download": 162.557851076,
    "File": "index.php",
    "Test status": "Partial OK",
    "Lookup status": "Word not found in /root/uptime_challenge_master/worker/10.0.0.0/index.php"
}
```

The following attributes were set for the account when the calculation took place:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial OK punishment decrease</td>
<td>0.1</td>
</tr>
<tr>
<td>Hourly rate</td>
<td>3</td>
</tr>
<tr>
<td>Time since last check</td>
<td>1325</td>
</tr>
</tbody>
</table>

The variable that contains the time-length since the last check has to be generated as a random number between 10 and 2000 (Same as for the groupE simulation). Using the information and formula above, one can calculate an award of 0.1104.

**Distribution dataset groupC and uptime percentage**

This section provides a sample of the dataset that was used to generate the graph for groupC. The data can be used to calculate an uptime-percentage, with the same methodology that was used for groupE in an earlier example.
Table 6.5: GroupC Balance 1% distribution dataset

<table>
<thead>
<tr>
<th>1% categories</th>
<th>Frequency</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>-149</td>
<td>749</td>
<td>7.509524764</td>
</tr>
<tr>
<td>-130</td>
<td>809</td>
<td>15.6206136</td>
</tr>
<tr>
<td>-112</td>
<td>757</td>
<td>23.2103469</td>
</tr>
<tr>
<td>-94</td>
<td>777</td>
<td>31.00060156</td>
</tr>
<tr>
<td>-75</td>
<td>766</td>
<td>38.68056948</td>
</tr>
<tr>
<td>-57</td>
<td>792</td>
<td>46.62121516</td>
</tr>
<tr>
<td>-39</td>
<td>812</td>
<td>54.76238219</td>
</tr>
<tr>
<td>-20</td>
<td>787</td>
<td>62.65289753</td>
</tr>
<tr>
<td>-2</td>
<td>831</td>
<td>70.98455986</td>
</tr>
<tr>
<td>16</td>
<td>353</td>
<td>74.52376178</td>
</tr>
<tr>
<td>35</td>
<td>298</td>
<td>77.51152998</td>
</tr>
<tr>
<td>53</td>
<td>294</td>
<td>80.4591939</td>
</tr>
<tr>
<td>71</td>
<td>274</td>
<td>83.20633647</td>
</tr>
<tr>
<td>90</td>
<td>301</td>
<td>86.22418288</td>
</tr>
<tr>
<td>108</td>
<td>315</td>
<td>89.38239422</td>
</tr>
<tr>
<td>127</td>
<td>325</td>
<td>92.64086625</td>
</tr>
<tr>
<td>145</td>
<td>303</td>
<td>95.67876479</td>
</tr>
<tr>
<td>163</td>
<td>315</td>
<td>98.83697614</td>
</tr>
<tr>
<td>182</td>
<td>96</td>
<td>99.79947864</td>
</tr>
</tbody>
</table>

To calculate an uptime-percentage from the table above, one has to look at the interval from -2 to 16. According to the table, there is a probability of 70.98% to end up with a reward below -2. A negative reward would mean that the site is not responding. The uptime-percentage for groupC this period would therefore be:

\[
100 - 70.98 = 29.02
\]

An uptime-percentage of 29.02% indicates that the site has only been responding 29.02% every time the Purser check has sent a request to the groupC-Bookface. The Leeshore-check could have played a big role into making sure that the Bookface site is down. Leeshore connects to each group every second hour to turn off a virtual server; this may result in the Bookface-site being taken down. If the students don’t get the site up and running again, the Purser check will give status "Not Approved" continuously.

6.8 Statistics for the course

The prototype executed over a total period of 4 months. During this period, the application produced more than 3 million reports for 16 groups. A report is generated every time a test has executed. These reports have been analysed by the system before they have been saved. The reports have been used in the analysis section to provide statistics and graphs outlining different aspects of the project period.
6.8.1 Course survey

Students participating in the uptime challenge had their final exam in the course Friday 29 April 2016. At the end of the exam, a survey was conducted where each student got a chance to evaluate the course. Two of the questions in the survey asked the students for opinions on the usage of Bookface in the uptime challenge. The subsections below will outline the two questions and provide statistics from the survey outlining the student’s response. The survey was conducted in Norwegian. The questions are listed in Norwegian to present questions in original form. Along with the Norwegian translation, each question will be translated into English. Each question were listed as an assertion. The students had to choose a reply in range from Agree to Disagree. Each option corresponds to a grade, where Agree counts as 5 and Disagree counts as 1. The full list is provided below:

- Enig - Agree - 5
- Noe enig - Somewhat agree - 4
- Hverken enig eller uenig - Neither agree nor disagree - 3
- Noe uenig - Somewhat disagree - 2
- Uenig - Disagree - 1

6.8.2 Question one: "Verdspillet" med Bookface motiverte meg til å bruke mer tid på kurset/ The "value-game" Bookface motivated me to use more time on the course

23 students replied to this question. The results are provided below:

<table>
<thead>
<tr>
<th>Number of replies</th>
<th>Mean</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>4.4347</td>
<td>4</td>
</tr>
</tbody>
</table>

The full statistics can be seen in the bar plot below:
Analysis

The results for the question shows that the uptime challenge motivated the students to participate more in the course. Most students agree that the contest motivated them to improve their Bookface site, when getting continuous feedback from the prototype. One out of 23 students neither agreed of disagreed to the question. The results shows that the students eagerly wanted to improve their infrastructures to get better results in the uptime challenge.

The results give a strong indication that the students liked participating in the uptime challenge. By competing against each other, the students become more motivated to improve their solutions in order to get the best Purser balance.

6.8.3 Question two: "Verdispillet" med Bookface motiverte meg til å bruke mer tid på andre aktiviteter med kurset som ikke var direkte tilknyttet spillet. / The "value-game" Bookface motivated me to use more time on activities related to the the course that was not directly connected to the game.

23 students replied to this question. The results are provided below:

<table>
<thead>
<tr>
<th>Table 6.7: Question two statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of replies</td>
</tr>
<tr>
<td>Mean</td>
</tr>
<tr>
<td>Median</td>
</tr>
</tbody>
</table>
The full statistics can be seen in the bar plot below:

![Bar plot](image)

**Figure 6.12:** A bar plot outlining the frequency of grades that were conducted from question two in the student survey

**Analysis**

An average of 3.89 indicates that most students agree to the assertion that the uptime challenge motivated them to participate more in the course with elements not related to Bookface. Some of the students somewhat disagrees to the question. This could indicate that their site was not properly working over time, or they did not get the feedback they needed from the prototype to improve their solution. During the course, the prototype encountered some technical problems with the Purser test. When the Bookface database grew bigger, the Purser test would take longer to execute. This ended in some of the test results being outdated. This could again cause the feedback to show that the Bookface site was down, when it actually was working. Some of the students replied that they agreed and somewhat agreed to the question. The question investigates if the uptime challenge gave the students motivation to learn more about other course related topics. The results indicates that the students are satisfied with prototype and the concept. It motivates them to learn more about other topics in system administration.
Chapter 7

Discussion

This thesis aimed to investigate the possibility of implementing a scalable framework with ability to provide relevant QoS data in value-driven operations. The chapter will discuss the choices made in the project period and the prototype developed. The discussion will start at the end of the project period, discussing the project outcome.

7.1 Student survey

At the end of the project a survey was conducted at NTNU Gjøvik for the course where the uptime challenge contest was held. Two questions regarding the uptime challenge were asked. (The questions with statistics can be seen in the Analysis chapter.) The questions investigated the student motivation for improving the Bookface sites, and if the uptime challenge motivated the students to participate more in the course topics. The question regarding the motivation to improve the Bookface site, gave a clear indication that the students liked participating in the contest. The reply average to this question was 4.44 (5 = Agree, 1 = Disagree), which indicates that most students agrees that the contest was motivational. The second question gave a little more unclear result regarding the motivation to learn more about other related topics in the course. (3.87) Most students agreed to some extent that the uptime challenge motivated them to learn more on other course topics.

7.1.1 Problem statement versus result

Part of the problem statement presented in the introduction chapter states that the framework may give better insight into value-driven operations. The results conducted from the survey showed that the prototype used in the uptime challenge helped some of the students to be more motivated to get better insight into IT-operations. By getting continuous feedback from the prototype on their Bookface sites, the students could improve their solutions by looking at the feedback over time. The results show that the uptime challenge helped the students to spend more time on the course, and that it motivated them to get better insight into new
system administration related topics. The results indicate that the students achieved motivational value from using the prototype.

7.2 Final result

The prototype developed during the project period offers the user the functionality of automating IT test-deployment against IT-services. The system is optimised and developed for performing educational evaluation tests against students-projects. The design-phase outlines several technical requirements and proposals that are implemented in a working prototype. The problem statement for the project used adjectives such as value-driven and scalable. Scalability and value has been key-terms for the entire project. The next sections will outline the scalability and the achieved value one can get from using the prototype.

7.2.1 Educational value

A prototype of the application has been used for educational purposes at NTNU Gjøvik. (More information about the development-phase can be found in the Implementation chapter.) The initial goal was to develop a prototype with ability to be used in a real course. The data produced from using the prototype, provides real statistics from student projects, and can be analysed directly in the analysis. The data gathered during the course has been used to create graphs for the Analysis-chapter, and outlines the final results of the project. By having students participating in the uptime challenge with their own Bookface-sites. It has become possible for the students to suggest improvements during the test-period. Some of the students have created their own web site showing progression of different groups during the contest. The site outlines if the Bookface-site is up, the actual balance and the balance-development the last 30 minutes. By getting feedback from students and the supervisor, it was possible to improve and fine-tune the system, while the prototype was running. Part of the project focused on using deployment-methodologies that allowed parts of the application to be taken down temporarily while system upgrades and changes to the code took place.

7.2.2 Pedagogical value

The system developed in this project requires some technical maintenance to be operated properly. The system is developed with intention to be expanded with new functionality. The next sections describes the difficulty of setting up different modules in the infrastructure. There are some technical requirements to get the system up and running. The teacher using the system must have some technical background using Linux, and should be capable of doing some maintenance and configuration on the Linux-platform. Research conducted during the project shows that students can get instant feedback from the system. Some of the tests can also work as
robots interfering with student infrastructures to simulate power outages and other technical challenges real infrastructures may meet. The goal of using the application developed in the research project is to help students prepare for real situations that a system administrator may end up in. A system administrator may meet different problem-scenarios at work, and by participating in the uptime challenge, students learn how to prepare for unexpected events that may occur on large-scale infrastructures. The students compete against each other of operating the infrastructures the best possible way. The competition can be used as motivation for students, and allows students to continuously improve their infrastructures by getting constant feedback from tests running against their site.

7.2.3 Scalability

The solution is easy to scale. The only requirement to scale the solution is to install some small scripts on the new nodes. When using Docker to administrate the workers, Dockerfile can be used to build images with ability to deploy new instances. Automation tools such as Puppet or Chef can govern the number of running workers. The final prototype does not automatically scale the number of workers. This has to be done manually using Puppet or Chef. It was prioritised to focus on implementing more tests that could be used by the prototype. This was of high importance since the prototype was going to be used in a course at NTNU Gjøvik. Workload-profiles are used for some of the tests to calculate connection-rates and to decide the number of tests to deploy. The workload-profiles could easily be used to estimate the number of needed workers. When an automated scaling-mechanism has been implemented. This could be linked together with the workload-profiles to estimate the exact number of workers needed per job-deployment.

7.3 Maintenance and configuration

The prototype is developed with intention to be used by teachers evaluating student projects. To set up the infrastructure and application from scratch should not be very difficult to do. During the development period, the entire infrastructure was taken down and deleted several times for testing purposes. Depending on the desired size of the infrastructure, it should not be difficult get the services up and working. For courses where a student-assistant is hired, it could be possible to let the student-assistant take responsibility for maintaining and operating the prototype.

7.3.1 Managers

The managers are configured to be easy to set up. If the goal is use a manager out of the box without modifying the code, the only requirement is to set some variables to point to the RabbitMQ-server and the interpreter API. Along with every manager package, comes a Linux init-script with
ability to start, stop and restart the service. This makes it possible to run the Manager as a Linux service, and it can be configured to start as part of the boot process. The managers are built to use logging, and by changing the logging level in the code to debug-mode, one can get more output of what is happening. If there is a need to build new managers from scratch, it is important to understand how the other managers are built. There are some technical requirements for the initiation process, and for communication with the API that can be found by using other managers as reference.

7.3.2 Workers

The workers come shipped as Docker-instances. The Docker-instances contains a script that executes the worker. The worker-script is responsible for executing the test, and must therefore contain a valid, executable path to where the test is located on the Docker instance. Predefined Docker-files have been created for each worker. Some of the Docker-files are already linked up to Dockerhub and Github, and are triggered to rebuild its Docker-image every time new code is pushed to a specific Github-branch. By using existing worker-packages, the only needed requirement is to set a valid IP-address of the RabbitMQ server in the worker-script. This IP-address is necessary to ensure communication between the workers and the job-queue.

7.3.3 Interpreter API

The API should be simple to set up. The API implements several Python libraries that must be installed on the Interpreter server. The API can very easily be extended with new functionality. The API is configured to log all transactions to a log-file that can be used for debugging if something goes wrong. The API is built to support asynchronous function calls, meaning several managers can call the same function at the same time.

7.3.4 CouchDB

The process of setting up and configuring a CouchDB-instance is well documented on the home-page of CouchDB. There are three databases needed for the system: Accounts, Reports, and Config. The Account objects must be built following a strict structure. The structure is described into more detail in the analysis-chapter. A CouchDB-administration script has been created to make it possible to create accounts from the command-line. CouchDB supports clustering. When operating in large-scale infrastructures, the clustering-feature can be used to solve database performance problems. A new CouchDB-instance can easily be set up to replicate all CouchDB data, and to work as a slave for the CouchDB-master instance.
7.3.5 Overall setup

To set up the infrastructure with one manager and corresponding workers, should not take more than two working days. (One working-day=8 hours) There will always be things happening that are difficult to foresee, and one must therefore take this into account when setting of the infrastructure. A teacher normally has approximately 430-450 hours in budget to use in a course per semester. To maintain the entire infrastructure, the teacher must at least budget two hours per week for system-maintenance.

7.4 Analysis of graphs

The analysis chapter outlined four different analyses of the prototype and its data. The data gathered during the analysis-period has been used to project student-progress, and to evaluate the performance of the prototype. The analysis section focused on outlining approaches relevant for the problem statement. The workload-profile graphs illustrated the strength used for the Webuse, and Httpperf test. The strength could again be used by the managers in the future to deploy the exact number of needed workers to execute waiting jobs, in order to not allocate more performance than necessary.

The Purser, Clerk and Leeshore graphs outlined crucial evaluation-data one could look at when evaluating a group’s effort to keep the Bookface site up and running. The correlation between the graphs can be used by a teacher to look at historical test data for a group, and how results from different tests are related.

The last analysis covered two uptime-percentage evaluations where the total number of Purser reports was looked at. From the result one could calculate an actual uptime-percentage for the group for the entire course-period.

The results from the different analyses have helped to conclude the analysis-period and outline the results discussed in the Discussion-chapter. The analysis-data can be used as a reference when concluding if the project outcome reflects the problem statement, or not.

7.5 Implementation

The prototype developed during the project was developed with the intention to be used in educational courses at Oslo and Akershus University College of Applied Sciences(HiOA) and NTNU Gjøvik. A design-model was set in place at an early stage, and has been used during the entire development phase. Scalability, QoS and value-driven have been important keywords during the design-phase. The design-model outlines a prototype with ability to scale based on need. The solution developed makes it possible to create custom QoS tests that can used directly without to much reconfiguration. This section will go into detail on the development process and outline advantages and future improvements.
Infrastructure

The prototype is built and developed using two infrastructures: ALTO-cloud at HiOA and the OpenStack cloud at NTNU Gjøvik. The managers are running in the ALTO-cloud with the static infrastructure (API, CouchDB, RabbitMQ), while the workers are running at the Cloud at NTNU Gjøvik. The only communication channel between the infrastructures is the RabbitMQ server. The managers and workers connect to the RabbitMQ server to post/fetch jobs/reports.

Complexity

The solution is designed and built using very specific design-requirements. The solution is developed using Python and open source technologies. When the project started, there was little research on the field that could help building a more documented and standardised solution. Different technologies and solutions had to be tested thoroughly, in order to solve some of the challenges faced during implementation. The aim during implementation was to implement a framework that could be maintained and developed further by teachers. All code is documented and commented. This was a technical requirement before the implementation started. The project aimed to build a prototype that could easily be maintained and further improved by adding new scripts. The decision to use Python was made early in the project. Python is a lightweight programming-language that does not require heavy technical competence to learn. This makes it easier for teachers and end-users of the application to debug the code if problems arise. Along with being easy to use, Python also supports integration across different platforms. This made Python a good choice to use when building the framework.

Workflow

Each node is configured to follow a specific workflow where some steps are predefined upon installation of new workers and managers. Some of the steps in the workflow need to be defined by the end-users. The workflow for each role in the project was predefined as part of the design-model. If new tests are implemented, new managers and workers can easily be configured by following a set of instructions. Each manager is equipped with a set of functions to fetch users, deploy jobs and reading reports. The workers are configured to listen to a given queue for new incoming jobs, and execute the received string.

7.5.1 Data

All data is stored in a non-SQL database. By using CouchDB all data is stored as JSON. By creating pre-defined queries in the CouchDB using Javascript, Python can make a function call directly to the predefined query. JSON is used as the data-format for all data sent between instances. By using JSON, one can more easily send data between system developed in
different programming languages. Output data from a Perl-script can be parsed directly by Python code and used without type-conversion. This reduces complexity and performance usage.

7.6 Approach and design

The approach of this thesis outlines the design of the prototype. The approach focuses mainly on the design-phase, and the methodology behind. When looking at the implementation of the product along with the design-plans, one can see that the product satisfies the design-requirements most of the time. In the original design-plan, more functionality was planned, but not prioritised. The design was made before the implementation started. As a research-project, it was important to have a good approach with strict design-plan before the implementation started. In the corporate world, the developed product will always be the main focus. In a research project, the main focus will be on finding a problem, and suggest an approach how the problem can be solved. The product developed during the implementation-period is made with the intention of mapping students progression in system administration-related courses, and to conduct research on automated evaluation-processes for student projects. As a master-thesis in network and system-administration at UiO, the design focuses on the infrastructure along with the technical requirements for the application. The product aims to be configured using good IT-operational standards for automation, security and network.

7.7 Impact

The prototype developed during the development phase is optimised and developed to be used for educational purposes. The prototype takes the evaluation-aspect of student projects into consideration, and thrives to give students continuous feedback on their projects. The system developed is built using different technologies. The project was divided into smaller projects that were solved individually before they were assembled together as one application. Each module in the system is built to execute separately without much maintenance from the user of the system. The system implements use of new, stable technologies that has proven to work well as a part of the infrastructure.

7.7.1 New thinking

By taking the educational evaluation-aspect into consideration, the system is unique when it comes to providing students continuous automated feedback on student projects. The system uses Docker as deployment-platform for the workers. Docker has become more and more used in enterprise projects all over the world, and aims to be a replacement for virtual machines. Automated deployment using Docker is commonly used by companies delivering IT-services. There are several similarities between
the prototype and open-source IT-services. The prototype speaks for itself when it comes to evaluation by the use of managers and workers. The manager-worker methodology is a new way of thinking in educational IT. The concept of letting a manager-instance deploy jobs over a queue to waiting workers is a new way of thinking. Queues have been used earlier as a way to store waiting jobs for an application. The prototype developed aims to investigate the need for automated feedback for students. The prototype is not developed with the intention of being a complete product. It is part of a larger research project that takes the entire evaluation process of students into consideration.

Assessment

Quality of Service (QoS) is used heavily in the corporate world. QoS makes it possible for companies to read evaluations of new products before a decision to invest is made. QoS in education is not frequently used for IT-students. For students taking system administration-related courses. It is important to learn how to debug and quality-check infrastructures. The prototype aims to prepare students for a career in system administration after they finish their education. The checks developed for the prototype prepares the students for maintaining infrastructures with high traffic and activity. By using a check that turns off a random server in the infrastructure, the students are forced to implement monitoring functionality to make sure that the student is alerted when a server stops working. The students learn how to maintain the infrastructure without taking down their services. When a service stops working, it could be a costly affair. The Clerk check charges the student-groups an hourly fee for the size of their running infrastructures, while the Purser check charges the students a punish-cost if the site is down. This simulates a real situation where a company could lose money if the system administrators don't consider uptime as an important factor when maintaining and upgrading an infrastructure.

7.8 Future work

This section will discuss future potential improvements for the prototype. Problems that arose during development will be discussed and how they were solved.

7.8.1 Scalability projections

The finished prototype does not contain a way to analyse the amount of workers needed to make sure that every test deployed by the Managers are executed successfully. Currently there are five different managers, governing five tests. Each test has a different responsibility, and is deployed using different deployment-mechanisms. Leeshore follows a regime where a new test is executed every second hour for each group. Purser, Httpperf
and Clerk follow the same deployment regime, where one test is deployed every fifth minute for each group. Webuse deploys a calculated number of tests every fifth minute per group. The challenge arises when there comes a need to automate scaling globally without taking the test-type into consideration. The next sections will take each test into consideration, and elaborate how an automated scaling-methodology can be applied.

Choosing a platform

The virtual platform to run workers on plays a big part in the scalability problem. When using containers (Docker), it becomes easy to scale the amount of workers. If there is a need to use virtual machines as worker instances, one must implement functions that use virtual platform tools to be able to scale the number of running instances. If there is a test that requires high security and heavy human monitoring, automated scaling may be a bad choice. When implementing checks such as Webuse, it is possible to use Docker because of the low security need. Docker distributes a build-platform called Dockerhub where users can compile their own Docker images online, and download the image to any running machine anywhere. Docker images on Dockerhub are most commonly open for the public, meaning anyone can download the image. Dockerhub supports private image-repositories, but this can be an expensive affair when operating with several different Docker-images over time. When using Dockerhub as the image-compilation platform, the scaling-opportunities become bigger. Several third-party tools supports integration with Dockerhub, and has the ability to re-deploy all instances every time a Docker image is recompiled. By using Puppet along with Dockerhub, one govern the number of running workers automatically.

Webuse

In the statistics-part of the analysis-chapter, workload-profiles was discussed in detail. The section showed some graphs projecting the total amount of jobs Webuse deploys per deployment for all groups. Depending on the workload-profile, the number of workers needed can vary drastically. To be able to project beforehand the number of workers needed, one can implement a check that gathers what workload-profile each group is set to use, and runs a simulation for 24 hours of deployment to see how the number of tests may vary. Each workload-profile is created with the intention that no matter what offset different groups have, the total number of deployed tests should be approximately constant every deployment. As seen on the workload-profile graphs in the analysis-chapter, the total number may vary a little due to the use of noise. (The last number in a workload-profile calculation, e.g. 5:g:3. The number is used to generate a number+3). If Docker is used as the worker platform. The number of running instances could be governed by a third-party mechanism, which uses the numbers from the progressions as a template for the number of workers to deploy every hour. A challenge one faces with Webuse, is the increasing
size of the Bookface-database. As the Bookface-database grows bigger, the Webuse-checks will take longer to execute. This must be taken into consideration. Webuse implements a function to randomize what function to execute every time the script runs. The different functions use different amount of time to finish.

Httperf

The strength of the Httperf connection-rate is governed by workload-profiles. Httperf uses the same mechanism to calculate the connection-rate as Webuse implements to determine the number of jobs to deploy per group. To be able to foresee the amount of workers needed for Httperf, one need to look at the performance-usage on the hardware where the workers are running. During development, tests showed that Httperf consumes a lot of performance when running with high connection rate. If Docker is used to deploy worker instances, the number of running Httperf-workers running per virtual machine must be looked at along with the connection-rate configured for the Httperf-test. With Docker, one can scale the number of running instances vertically inside a virtual machine, but there might come a need to scale up the amount of virtual machines to execute Docker-worker instances on. Normally a virtual machine contains different workers running different checks. They run separately in a sandbox-environment where they can run undisturbed over time. Httperf-workers have been famous for allocating all available performance-resources on virtual machines, when running with high connection rates. To be able to automatically scale the number of instances needed when the connection rate changes over time, there are three different methodologies to follow:

- Increase/Decrease the number of running instances per virtual machine
- Add more virtual machines to run Docker-instances on - horizontal scaling of virtual machines
- Allocate more virtual resources to each virtual machine running Docker-instances as the connection rate increases. - Vertical scaling

Changing the deployment-interval, may also affect the scalability-need. Each Httperf check is set to execute for the length of a deployment interval. The minimum number of Httperf workers will therefore be equal to the total number of groups.

Purser

Purser is the most important check developed for the prototype. Purser is responsible for evaluating different QoS criteria for the site. Purser interacts with the site, and inspects certain elements on the site to evaluate if the site is working properly. At the end of the project period, the Bookface sites of the groups got bigger and bigger. This resulted in the site reacting
slower, causing Purser to take longer and longer to execute. Purser needs to download the entire front-page with all images to evaluate the latency of the Bookface-site. When the site is large in size, this may take long, causing a Purser check to use longer time than a deployment interval lasts. This causes the Purser work queue to increase in size. The Purser manager will continue to deploy new jobs, but there are no available Purser-workers to take the job. When creating an automated scalability regime for the Purser check, it will be hard to estimate the number of needed workers. Sometimes, the Purser check is fast to execute and download the site. If Webuse-checks and Httperf-checks are running against the site at the same time Purser executes, it may cause the test execution time to be longer.

A way to scale the number of Purser workers is to monitor the download-time of the frontpage in each report. This will help the Manager to keep track of the response-time of the Bookface-site, and the Manager can scale the number of workers accordingly.

Leeshore

Leeshore is set to execute every second hour. The Leeshore check needs access to an OpenStack user-account with administrator-access to allow the script to turn of a random server in each groups virtual environment. The Leeshore script must be executed with caution, since administrator-privileges are given to an automated mechanism. Leeshore does not take long to execute, and deploys only job per deployment. (Deployment-interval=120 minutes/Total amount of groups). Only one worker is needed for Leeshore to execute. Most of the time, the Leeshore worker will be in listening state, waiting for a new job to be posted on the work queue.

Clerk

Clerk is set to execute for each group every fifth minute. Clerk needs a user-account in the OpenStack environment with permission to access each groups tenant and count the number of running virtual machines with their flavor type. A price is then calculated from the virtual-machine count. Clerk does not need very long to execute. For a system with 15 groups, only two Clerk workers are needed.
7.8.2 Scaling by monitoring RabbitMQ

One way to globally scale is to monitor the different work queues. The following command will outline all the queues on the system, and the number of elements on each queue.

```
Listing 7.1: RabbitMQ queue overview
1 root@rabbitmq:/home/ubuntu
2 # rabbitmqctl list_queues
3 Listing queues ...
4 clerk_report_q 0
5 clerk_reportq 6
6 clerkq 0
7 httprfq 0
8 httpsfreportq 0
9 leeshore_reportq 0
10 leeshoreq 0
11 purser_report_q 0
12 purserq 6
13 testq 0
14 webuseq 21
15 webuserreportq 0
16 ... done.
```

A script implementing this command can be used to constantly make sure that the queues are maintained constantly. If a report queue keeps increasing, it would mean that the Manager is not working properly. If the job-queue keeps increasing more workers are needed. It is possible to use the queue as a mechanism to downscale the amount of workers if the number of elements on a queue almost constantly is 0.

7.8.3 Evaluation and valuable feedback

During the development phase, the most valuable live-feedback to the groups was through the rewards that the Purser check gave in each report. The reward was then added to a balance to show progress over time for each group. The uptime-percentage has also been used to evaluate the overall group-performance on their Bookface site. When creating a system with ability to deploy tests against student projects and giving instant feedback on the result automatically, there are several challenges that arise. One of the main challenges one phase is to make the evaluation understandable for human eyes. There is a limit how complex evaluation a system can do before it becomes difficult to understand for the students. For future related projects, it could be smart to implement different evaluation checks to evaluate different aspects of the user-site. Important QoS elements to evaluate would not only be the site itself, but the environment it runs on. By analysing technologies used to keep the site up and running, determine technologies used to optimize site latency, it is easier to get a broader picture of the effort students put down in their work.

Cost

When running a large-scale site, cost is considered to be a critical aspect one must keep in mind. During the project, some of the groups had
highly complex sites running on several virtual servers. If Leeshore took down one of the servers, the site would not go down, due to redundant database, and front-end servers. Even though the site is still up, the cost of running the infrastructure behind might be to high. Clerk has taken care of charging the groups a cost for all running virtual machines. In the future, other cost-calculating functions could be implemented. Network-bandwidth monitoring and power-consummation are examples. When companies outsource their virtual environments in the cloud, they often pay a fee for the amount of sent and received data per month. To make it more realistic for the students, a network monitoring service that monitors the amount of data received from the Internet and calculates a cost could be made.

7.8.4 Campaign

During the development phase of the prototype, there had been set up plans to implement a check to execute in short periods to work as a campaign on the Bookface site. A campaign would be a short period of time where the intensity of checks increased drastically. The point of implementing a campaign is to see how the site operates under extreme pressure, and to check the latency of the site. A campaign would be similar to a web-shop having a sales-campaign with increased visits from customers, causing the web-shop to use more performance to keep the site up and running at a 100% capacity. There are two ways a campaign could be implemented:

- Traffic-based campaign by increasing the Httpperf connection rate drastically
- Activity-based campaign by increasing the number of deployed Webuse-tests against each group per deployment.

A campaign can already be set in motion for the groups by changing the workload-profile for an account. This would cause Httpperf to increase connection-rate and Webuse to increase the number of deployed tests.

7.8.5 Integration against other sites

During the project, Bookface has been used as test-site. Some have been developed specifically to integrate with Bookface. The Purser-tests evaluates specific criteria for the Bookface site by searching for a given word on the site. It is not a big change to make Purser evaluate other websites the same way. The Httpperf-test are more generic because it only generates a set of empty HTTP-GET calls to Bookface. Webuse is developed purely for Bookface, because it uses PHP-scripts to add users, comments and posts. In the future, similar sites can be developed to improve the test environment, and open up the possibility of testing new functionality through the use of the prototype developed in the project. Testing against other sites has not been performed during the development-phase. Httpperf
is a powerful traffic-generation tool with ability to create massive loads of connections to any site wanted. Earlier testing with Httperf has shown that it can easily take down websites, due to the number of incoming connections to the site. All tests executed during the project has been in a local network at NTNU Gjøvik and the ALTO cloud at HiOA.

7.8.6 Automation

The workers used in the project have all been deployed using Docker and Puppet. The managers have been running as virtual machines. In the future, there is nothing stops the Managers from being executed as Docker-instances. When using Docker, the instance runs in a closed environment, and connects with other nodes through the underlying virtual machine that hosts the Docker-instances. To make is easier to maintain and test, the Managers were executed inside virtual machines. This makes it easier to connect to the servers, and monitor log-output live. Tests during development of the workers showed that it was harder to debug events and errors that occurred, when the workers executed inside Docker-instances. When an event happened, causing the program to stop, the Docker instance would die, and the error output would be lost.

7.8.7 Creating new workers and managers

The managers and workers follow the same workflow and must be implemented to work with the rest of the infrastructure. A new manager can very easily be developed using the framework developed in the prototype. Functions to post/fetch elements from the queue, listen to the queue and API functions are implemented. The workers execute from a script locally, and a template-script can be used to create new workers more easily. There was originally a plan to implement a set of more generic functions that could be used more widely across all managers. It was prioritised during development to focus on implementing more tests to be used when the course was held at NTNU Gjøvik. In the future, one can easily add more functionality to the managers by extending the framework with more functions.

7.8.8 REST

The API used between the interpreter and the managers, are developed using the Python-API library Pyro4. For the future a REST-API could be more effective, opening up for better communication with the database. CouchDB has developed a REST-API that can be called upon directly and used for queries and transactions. The interpreter can therefore easily be modified or removed in future projects. There was a demand during development, that all communication to/from the database should go through one entity, therefore the interpreter was created.
7.8.9 Computer security

Computer-security related tests could easily be implemented to fit with the application. Some student projects require students to build and configure complex firewalls to protect an infrastructure. By using the prototype, automated checks can be written and executed against the student-firewalls. The checks can try to penetrate the firewall on different ports, and map security flaws in the firewalls. Such tests can be developed as scripts, and can be made to be used along with the existing prototype.

7.8.10 Other programming languages

The tests executed on the workers today are written using Perl, Bash, and Python. The workers execute some of the test-scripts using an interactive bash-shell, enabling other programming languages to be used. Managers and workers can easily be re-written in other programming languages as long as they send data as JSON.
Chapter 8

Conclusion

This project aimed to develop an application with ability to automate evaluation-processes of student projects. The problem statement states:

"Design and develop a scalable framework that provides relevant QoS data which may give better insight into value-driven operations."

The problem statement covers a large problem-domain, and opens up for different interpretations. The research has focused mainly on educational related projects. The goal was to make a prototype that can be used directly by teachers in courses without big modifications. During the analysis period, the system was continuously patched and improved based on feedback from the students and teacher.

The final result reflects the problem statement and design plans. The infrastructure is included as a part of the end-product. This thesis is research based, and investigates the possibility of automated-evaluation in IT-projects.

The implemented solution validates the problem statement. A fully functioning scalable framework is implemented and it provides feedback to the end-user that can be user to foresee problems ahead. The feedback focuses on the uptime-aspect of web-sites. The prototype developed is optimized to run in an OpenStack environment. Several tests are developed specifically for OpenStack-related checks. At the end of the project, a survey was conducted for the students. The survey results gave a clear indication that the students became more motivated to learn more about system administration topics when participating in the uptime challenge.

The solution is made open-source and available to the public, which opens up the possibility for future improvements.
Bibliography


[36] Learn Amazon Web Services - simpleNeasyBook by WAGmob: WAGmob, 2013. URL: https://books.google.no/books?id=O6tuAgAAQBAJ.
[40] Peter Mell and Tim Grance. ‘The NIST definition of cloud computing’. In: (2011).


Appendices
Appendix A

Managers classes

The following appendix contains the manager-classes and subclasses used by the managers. Each class is provided is printed in landscape-mode, and can be used as a reference when reading the implementation-chapter. The Manager code directory can be found on Github by using the following URL: https://github.com/s177437/uptime_challenge/tree/master/managers
from Queues import *
from Config import *
import pika
import couchdb
import Pyro4
import logging
import time

__author__ = 'Stian Stroem Andersen'

class Httpmanager():
    
    """
    This module is the main class for the 
    purser manager. One test is deployed for 
    each user per deployment.
    """

    interpreterServer = Pyro4.Proxy( 
        "PYRONAME: interpreter"
    )

    def fetch_config(self):
        """
        This function is the main method where 
        everything is governed on the manager.
        """
        :return:
        :rtype:
        """

        math = WebUseMath()
        config = Config()
        newconfig = config_init_db_config()
        grouplist = newconfig_get_account()

        for i in grouplist:
            userconfig = \ 
            self.interpreterServer.getUserConfig( 
                i, "couchdb"
            )
            ip = userconfig["ipaddress"]
            worklist = []
            worklist = ["ip": ip, "sentence": userconfig["Sentence"], 
                "filepath": userconfig[ 
                    "filepath"], 
                "file": userconfig[ 
                    "file"], 
                "timestamp": 
                    time.time()}
            groupdict = {} 
            groupdict.update({i: worklist })
            logging.critical( 
                str(i) + " " + str( worklist ))
            newconfig.create_work_queue( 
                newconfig.get_queue_name(), 
                groupdict)
            worklist = []
            queue = Queues()
            queue.receive_one_message_from_q( 
                "purser_report_q", 
                str(newconfig.get_interval()))

manager = Httpmanager()
manager.fetch_config()
from Queues import *
from Config import *
import pika
import couchdb
import Pyro4
import logging
import time
import datetime
import sys

__author__ = 'Stian Stroem Anderssen'

class LeeshoreManager:
    pass

    """
    This module is the main class for the leeshore manager. This module takes care of the execution of the functions.
    """

    interpreterServer = Pyro4.Proxy("PYRONAME: interpreter")
    
    def run_leeshore(self):
        """
        This function deploys one leeshore-test per user every second hour.
        """

        day = int(sys.argv[1])
        config = Config()
        newconfig = config.init_db_config()
        grouplist = newconfig.get_account().get_groups()
        path = newconfig.get_script_path()
        runinterval = int(newconfig.get_interval()) / len(grouplist)
        logging.info("Interval: "+ str(newconfig.get_interval()))

        while True:
            for i in grouplist:
                userconfig = 
                    self.interpreterServer.getUserConfig(i, "couchdb")
                if datetime.datetime.today().weekday() == day or 
                    userconfig["leeshore_enabled"] == 0:
                    logging.critical("Today is a day off for ".append("leeshore")
                    time.sleep(runinterval)
                else:
                    tenant_name = userconfig["tenant_name"]
                    ip = userconfig["ipaddress"]
                    executable_string = 
                        "/root/uptime_challenge_master/" + 
                        "/testscript/leeshore_short.pl -n " + 
                        tenant_name
                    worklist = []
                    groupdict = {}
                    groupdict.update({i: worklist})
                    newconfig.create_work_queue(newconfig.get_queue_name(),
                        groupdict)
                    worklist = []
                    queue = Queues()
                    queue.receive_one_message_from_q("leeshore_reportq",
                        str(runinterval))

            manager = LeeshoreManager()
            manager.run_leeshore()
from Queues import *
from Config import *
import logging
from WebUseMath import *
import pika
import couchdb
import Pyro4
import time

__author__ = 'Stian Stroem Andersen'

class Httpmanager(
    
    "This module is the main class for the manager project. This module takes care of the execution of the functions.
    "
)
    
    interpreterServer = Pyro4.Proxy("
"PYRONAME: interpreter"
    
    def fetch_config(self):
        
        "This function is the main method where everything is governed on the manager."' +
        
        :return:
        :rtype:
        ""
        "This function is the main method where everything is governed on the manager."' +
        
        math = WebUseMath()
        strengthlist = math.create_time_list()
        position = 0
        config = Config()
        newconfig = config.initDbConfig()
        grouplist = newconfig.getAccount()
        path = newconfig.get_script_path()
        index = 0
        logging.info(" Interval " + str(newconfig.get_interval()))
        positiondict = {}
        ip = ""
        for i in grouplist:
            userconfig = ""
            self.interpreterServer.getFileAndOffsetFromUser("
"i"
            ipconfig = ""
            self.interpreterServer.getIpFromUser(""
"i"
            ip = ""
            ip = ipconfig["ipaddress"]
            executable_string = path + " 
"webuse.pl -U " + 
"i" + " -r " + 
"'10:10:10:10'"
            logging.info(str(userconfig))
            index = int(userconfig["offset"])
            logging.info("INDEX " + str(index))
            content = math.decideEntry(""
"strengthlist, index"
worklist = []
listvalues = math.convertToList(""
"listvalues"
position = int(listvalues[0])
strength_number = math.calculateList(""
"listvalues"
worklist = ""
"math.create_number_of_scripts(""
"strength_number," +
"executable_string"
"groupdict = ()
"groupdict.update((""
"i", worklist))
"newconfig.createWorkQ(""
"newconfig.get_queue_name(),
"groupdict)"
worklist = []
positiondict.update((""
"i", position))"
while True:
    for i, position in positiondict.
    positiondict.
    :return:
    :rtype:
    ""
    "This function is the main method where everything is governed on the manager."' +
    
    math = WebUseMath()
    strengthlist = math.create_time_list()
    position = 0
    config = Config()
    newconfig = config.initDbConfig()
    grouplist = newconfig.getAccount()
    path = newconfig.get_script_path()
    index = 0
    logging.info(" Interval " + str(newconfig.get_interval()))
    positiondict = {}
    ip = ""
    for i in grouplist:
        userconfig = ""
        self.interpreterServer.getFileAndOffsetFromUser("
"i"
        ipconfig = ""
        self.interpreterServer.getIpFromUser(""
"i"
        ip = ""
        ip = ipconfig["ipaddress"]
        executable_string = path + " 
"webuse.pl -U " + 
"i" + " -r " + 
"'10:10:10:10'"
        logging.info("USER: " + str("
"i") + " POSITION: " + str(
"position))"
        strength_value_as_string = ""
        math.jumpToNextEntry(""
"strengthlist, int(position)"
values_in_value_string = ""
math.convertToList(""
"values_in_value_string"
strength_value_as_string)
strength_number = \nmath.calculateList(values_in_value_string)
worklist = \nmath.create_number_of_scripts(strength_number, executable_string)
groupdict = {}
groupdict.update({i: worklist})
if position == 288:
    positiondict[i] = 0
else:
    positiondict[i] = position + 1
newconfig.createWorkQ(newconfig.get_queue_name(), groupdict)
queue = Queues()
queue.receive_one_message_from_q("webuserreportq", newconfig.get_interval())
manager = Httpmanager()
manager.fetch_config()
```python
from Queues import *  
from Config import *  
import pika  
import couchdb  
import Pyro4  
import time  

__author__ = 'Stian Stroem Andersen'

class Httpmanager():
    """
    This module is the main class for the manager project. This module takes care of the execution of the functions.
    """
    interpreterServer = Pyro4.Proxy("PYRONAME: interpreter")
    # comment
    def fetch_config(self):
        """
        This function is the main method where everything is governed on the manager.
        """
        return:  
        :type:
        """
        while True:
            config = Config()  
            config.get_account()  
            newconfig = config.init_db_config()  
            userinfo = \n
            config.request_user_creation(  
                newconfig)  
            grouplist = newconfig.find_groupnames(  
                newconfig.get_account(  
                    ).get_groups())  
            self.interpreterServer.createAccounts(  
                userinfo)  
            path = newconfig.get_script_path()  
            worklist = [  
                "python " + path +  
                "check_http.py db.no"]
            for i in grouplist:
                groupdict = {}  
                groupdict.update({i: worklist})  
                newconfig.create_work_queue(  
                    newconfig.get_queue_name(),  
                    groupdict)
            queue = Queues()  
            timestart = time.time()  
            while time.time() - timestart <= \n                float(  
                    newconfig.get_interval()):
                queue.receive_one_message_from_q(  
                    "reportq",  
                    time.time() - timestart,  
                    newconfig.get_interval())
```
from Queues import *
from Config import *
from WebUseMath import *
import pika
import couchdb
import Pyro4
import logging
import time

__author__ = 'Stian Stroem Andersen

class ClerkManager():
    """
    This module is the main class for the clerk manager. This module takes care of the execution of the functions.
    """
    interpreterServer = Pyro4.Proxy("PYRONAME: interpreter")
    logging.basicConfig(filename="/var/log/manager.log", level=logging.CRITICAL)

    # comment
    def run_clerk(self):
        """
        This function deploys one test per user by creating an executable string which is sent to the RabbitMQ instance.
        """
        :return:
        :rtype:
        math = WebUseMath()
        config = Config()
        newconfig = config.init_db_config()

        grouplist = newconfig.get_account().get_groups()
        path = newconfig.get_script_path()
        logging.info("Interval: " + str(newconfig.get_interval()))
        positiondict = {}
        while True:
            for i in grouplist:
                userconfig = self.interpreterServer.getUserConfig(i, "couchdb")
                ip = userconfig["ipaddress"]
                tenant_name = userconfig["tenant_name"]
                executable_string = "/root/uptime_challenge_master/" + path + "testscript/clerk.pl -n " + tenant_name
                worklist.append(executable_string)
                groupdict.update({i: worklist})
                logging.critical(str(i) + " " + str(worklist))
                newconfig.create_work_queue(newconfig.get_queue_name(), groupdict)
                worklist = []
                queue = Queues()

                queue.receive_one_message_from_q("clerk_reportq", "clerk_reportq", str(newconfig.get_interval()))

                manager = ClerkManager()
                manager.run_clerk()
Listing A.6: invoice.py

```python
import pika
import Pyro4

__author__ = 'Stian Stroem Andersen'

class Invoice:
    ""
    This class contains a function to calculate
    the rental price for different OpenStack
    flavors
    ""
    pricelist = {
        "m1.micro": 0.05,
        "m1.tiny": 0.1,
        "m1.small": 0.2,
        "m1.medium": 0.4,
        "m1.large": 0.8,
        "m1.xlarge": 1.6,
        "m1.2xlarge": 3.2
    }

    def calculate_price(self, accountingdict):
        price = 0
        for flavor, numberofunits in
            accountingdict.items():
            if flavor == "":
                continue
            else:
                unitprice = self.pricelist[flavor]
                price += unitprice * numberofunits
        return price / 12 * (-1)
```

Listing A.7: account.py

```python
__author__ = 'Stian Stroem Andersen'

class Account():
    """
    This class build the manager account based
    on the configfile content
    """
    course = 
    groups = []
    teacher = 
    semester = 
    students = []
    balance = 0
    configfile = 
    offset = 0
    ip = 

    def set_ip(self, i):
        """
        Set ip of the account-object
        :param i:
        :type i:
        :return:
        :rtype:
        """
        self.ip = i

    def get_ip(self):
        """
        Return ip of account-object
        :return:
        :rtype:
        """
        return self.ip

    def set_configfile(self, c):
        """
        Set config-file name for account
        :param c:
        :type c:
        :return:
        :rtype:
        """
        self.configfile = c

    def get_configfile(self):
        """
        Return configfile
        :return:
        :rtype:
        """
        return self.configfile

    def set_offset(self, o):
        """
        Set offset for account
        :param o:
        :type o:
        :return:
        :rtype:
        """
        self.offset = o

    def get_offset(self):
        """
        Return offset
        :return:
        :rtype:
        """
        return self.offset

    def set_semester(self, s):
        """
        Set semester
        :param s:
        :type s:
        :return:
        :rtype:
        """
        self.semester = s

    def get_semester(self):
        """
        Get semestername
        :return:
        :rtype:
        """
        return self.semester

    def get_course(self):
        """
        Return coursename
        :return:
        :rtype:
        """
        return self.__course

    def get_groups(self):
        """
        Return all student groups
        :return:
        :rtype:
        """
        return self.students
```

return a list of groupmembers
:**return:**

**rtype:**

```
return self.__groups
```

```python
def set_course(self, value):
    """Set coursename
    :param value:
    :type value:
    :return:
    :rtype:
    ""
    self.__course = value
```

```python
def set_balance(self, b):
    """Set balance value based on the report.
    :param b:
    :type b:
    :return:
    :rtype:
    ""
    self.balance = b
```

```python
def get_balance(self):
    """Return balance
    :return:
    :rtype:
    ""
    return self.balance
```

```python
def set_groups(self, value):
    """Set group members.
    :param value:
    :type value:
    :return:
    ""
    self.__groups = value
def set_teacher(self, value):
    """Set teachername
    :param value:
    :type value:
    :return:
    :rtype:
    ""
    self.teacher = value
def get_teacher(self):
    """Return the teachername
    :return:
    :rtype:
    ""
    return self.teacher
def set_student_list(self, studentlist):
    """Set the list of students.
    :param studentlist:
    :type studentlist:
    :return:
    :rtype:
    ""
    self.students = studentlist
def get_student_list(self):
    """Return a list of the members to a group.
    :return:
    :rtype:
    ""
    return self.students
Listing A.8: config.py

```python
from Queues import *
from Account import *
import json
import ConfigParser
import couchdb
import ast
import Pyro4
import random
import logging

__author__ = 'Stian Stroem Anderssen'

class Config:
    '''
    This module aims to build and keep track of
    the manager object during runtime. Every
    time a user creation is
    requested. This module is called.
    '''
    queue_name = ""
    scriptpath = ""
    interval = 0
    configdbname = ""
    dbserver = ""
    queserver = ""
    configinstance = "None"
    account = Account()
    interpreterserver = Pyro4.Proxy("PYRONAME: interpreter")

    @staticmethod
    def convert_json_to_dictionary(self, data):
        ""
        A function to convert JSON contents to
        a dict.
        ""
        return json.loads(data)[0]

    @staticmethod
    def request_user_creation(configobject):
        ""
        Build dictionary containing
        course information that is sent to the
        interpreter based on the content of the
        config object.
        ""
        teachedict = {}
        listtosend = []
        teachedict = configobject.get_account().get_teacher()
        listtosend.append(teachedict)
        semesterdic = configobject.get_account().get_semester()
        listtosend.append(semesterdic)
        coursedic = configobject.get_account().get_course()
        listtosend.append(coursedic)
        groupsandmembers = configobject.get_account().get_groups()
        tempdict = groupsandmembers[0]
        groupsdict.update(tempdict)
        listtosend.append(groupsdict)
        return listtosend

    @staticmethod
    def send_users_to_queue(accountlist):
        ""
        OUTDATED - This function converts the
        accountlist to a string that can be
        sent over a queue.
        ""
        listtosend = str(accountlist)
        logging.debug("listtosend: ", listtosend)
        listtosend = Queue()
        queue = Queue("createuserq")
        queue.create_queue()
        str(listtosend)

    @staticmethod
```

Listing A.8: config.py

```python
from Queues import *
from Account import *
import json
import ConfigParser
import couchdb
import ast
import Pyro4
import random
import logging

__author__ = 'Stian Stroem Anderssen'

class Config:
    '''
    This module aims to build and keep track of
    the manager object during runtime. Every
    time a user creation is
    requested. This module is called.
    '''
    queue_name = ""
    scriptpath = ""
    interval = 0
    configdbname = ""
    dbserver = ""
    queserver = ""
    configinstance = "None"
    account = Account()
    interpreterserver = Pyro4.Proxy("PYRONAME: interpreter")

    @staticmethod
    def convert_json_to_dictionary(self, data):
        ""
        A function to convert JSON contents to
        a dict.
        ""
        return json.loads(data)[0]

    @staticmethod
    def request_user_creation(configobject):
        ""
        Build dictionary containing
        course information that is sent to the
        interpreter based on the content of the
        config object.
        ""
        teachedict = {}
        listtosend = []
        teachedict = configobject.get_account().get_teacher()
        listtosend.append(teachedict)
        semesterdic = configobject.get_account().get_semester()
        listtosend.append(semesterdic)
        coursedic = configobject.get_account().get_course()
        listtosend.append(coursedic)
        groupsandmembers = configobject.get_account().get_groups()
        tempdict = groupsandmembers[0]
        groupsdict.update(tempdict)
        listtosend.append(groupsdict)
        return listtosend

    @staticmethod
    def send_users_to_queue(accountlist):
        ""
        OUTDATED - This function converts the
        accountlist to a string that can be
        sent over a queue.
        ""
        listtosend = str(accountlist)
        logging.debug("listtosend: ", listtosend)
        listtosend = Queue()
        queue = Queue("createuserq")
        queue.create_queue()
        str(listtosend)

    @staticmethod
```
def find_groupnames(self, grouplist):
    ""
    Return a list of the groupnames
    :param grouplist :
    :type grouplist :
    :return groupnames :
    :rtype :
    ""
    groupnames = []
    for i, j in grouplist[0].iteritems():
        groupnames.append(i)
    return groupnames

def write_config(self, configdata):
    ""
    Write config from the database to the
    config.ini file
    :param configdata :
    :type configdata :
    :return :
    :rtype :
    ""
    for key, value in configdata.iteritems():
        self.write_to_file(key, str(value))

@staticmethod
def write_to_file(self, key, value):
    ""
    Write config to config.ini
    :param key :
    :type key :
    :param value :
    :type value :
    :return :
    :rtype :
    ""
    with open('/root/uptime_challenge_master/manager/config.ini', 'w') as f:
        words = f.read().split()
        if key in words:
            logging.info("No need to write to file, "
                            "key is present")
        else:
            file = open('/root/uptime_challenge_master/manager/config.ini', 'a')
            file.write(key + ' = ' + value + '\n')

def read_config_from_file(self):
    ""
    Read static config provided in
    config.ini and build an account object.
    :return configclass :
    :rtype Config :
    ""
    config = ConfigParser.SafeConfigParser()
    config.read('/root/uptime_challenge_master/manager/config.ini')
    configclass = Config()
    configclass.set_account(config.get(Account, "course"))
    configclass.set_account(config.get(Account, "groups"), json.loads(config.get(Account, "groups")))
    configclass.set_teacher(config.get(Account, "teacher"))
    configclass.set_semester(config.get(Account, "semester"))
    configclass.set_dbname(config.get(Global, "configdbname"))
    configclass.set_script_path(config.get(Global, "scriptpath"))
    configclass.set_dbserver(config.get(Global, "dbserver"))
    configclass.set_queserver(config.get(Global, "queserver"))
    return configclass

def init_db_config(self):
    ""
    Build the final config object after the
    DbConfig is initialized along with the
    local config
    :return config :
    :rtype Config :
    ""
    config = self.initiate_users()
    configdict = self.interpreterserver.fetchConfig(config.get_account().get_teacher())
    self.write_config(configdict)
    configparser = ConfigParser.SafeConfigParser()
    configparser.read('/root/uptime_challenge_master/manager/config.ini')
    configparser.write_config(configdict)
    configclass = ConfigParser.SafeConfigParser()
    configparser.read('/root/uptime_challenge_master/manager/config.ini')
def create_user_list(self, usersdictlist):
    userlist = []
    for userdict in usersdictlist:
        userlist.append(userdict['group'])
    return userlist

@staticmethod
def create_work_queue(self, queuename, joblist):
    queue = Queues()
    for group, job in joblist.items():
        for j in job:
            jobdict = {
                group: j
            }
            queue.create_queue(queuename, jobdict)

    def get_queue_name(self):
        return self.__queue_name

    def get_interval(self):
        return self.__interval

    def set_config_dbname(self, value):
        self.configdbname = value

    def get_config_dbname(self):
        return self.configdbname

@staticmethod
def create_work_queue(self, queuename, joblist):
    ...
return self.configdbname

def set_queue_name(self, value):
    """
    Set queuename to be used
    :param value:
    :type value:
    :return:
    :rtype:
    """
    self.__queue_name = value

def get_account(self):
    """
    Return accountobject.
    :return:
    :rtype:
    """
    return self.account

def set_account(self, a):
    """
    Set accountobject
    :param a:
    :type a:
    :return:
    :rtype:
    """
    self.account = a

def set_interval(self, value):
    """
    Set interval
    :param value:
    :type value:
    :return:
    :rtype:
    """
    self.__interval = value

def set_dbserver(self, value):
    """
    set dbservername
    :param value:
    :type value:
    :return:
    :rtype:
    """
    self.__dbserver = value

def set_queserver(self, value):
    """
    set queueservername
    :param value:
    :type value:
    :return:
    :rtype:
    """
    self.__queserver = value

def set_config_instance(self, value):
    """
    Set the configobject before the
database config is applied. This is
    used instead of using a global variable
    :param value:
    :type value:
    :return:
    :rtype:
    """
    self.configinstance = value

def get_dbserver(self):
    """
    Return dbservername
    :param value:
    :type value:
    :return:
    :rtype:
    """
    return self.__dbserver

def get_queserver(self):
    """
    Return queueservername
    :param value:
    :type value:
    :return:
    :rtype:
    """
    return self.__queserver

def get_script_path(self):
400
401  def set_script_path(self, path):
402  
403  def return_load(self, x):
404      answer = []
405      load = self.function(x)
406      derivative = self.derived_off_function(x)
407      answer.append(float(load))
408      answer.append(float(derivative))
409      return answer
410
411  def function(self, x):
412      return -0.25 * x ** 2 + 50
413
414  def derived_off_function(self, x):
415      return -0.5 * x
416
417  return self.scriptpath
418
419  @staticmethod
420  def function(self, x):
421      return -0.25 * x ** 2 + 50
422
423  @staticmethod
424  def derived_off_function(self, x):
425      return -0.5 * x
426
427  def return_load(self, x):
428      answer = []
429      load = self.function(x)
430      derivative = self.derived_off_function(x)
431      answer.append(float(load))
432      answer.append(float(derivative))
433      return answer

401  
402  Return the path to the executable
403  location of the test on the
404  LeeshoreWorker.
405  :return:
406  :rtype:
407  
408  def set_script_path(self, path):
409  
410  Set the script path on the LeeshoreWorker.
411  :param path:
412  :type path:
413  :return:
414  :rtype:
415  
416  self.scriptpath = path

410  
411  Set the script path on the LeeshoreWorker.
412  :param path:
413  :type path:
414  :return:
415  :rtype:
416  
417  self.scriptpath = path
import random
import time

__author__ = 'Stian Stroem Andersen

class WebUseMath:
    
    """
    Math-class that contains different
    functions used to calculate the given
    number of tests to deploy per user
    for different managers.
    """

    @staticmethod
    def create_time_list(self):
        """
        Read transsine-files and return a list
        of each line.
        """
        content = [line.strip("\n") for line in
                   open("/root/uptime_challenge_master/"+
                        "manager/transsine.dat")]
        return content

    @staticmethod
    def decide_entry(self, filelist, offset):
        """
        Decide workload-profile line-entry for
        a user
        """
        mytime = time.time()
        now = int((mytime % 86400) / 300)
        entry = (now + offset) % 288
        content = filelist[entry - 1]
        return content

    @staticmethod
    def jump_to_next_entry(self, filelist, position):
        """
        Jump to next entry in the workload-profile
        """
        if position == len(filelist):
            return filelist[0]
        else:
            return filelist[int(position)]

    @staticmethod
    def convert_to_list(self, strength):
        """
        Convert Python string to list and
        return it
        """
        listvalues = strength.split(":"))
        return listvalues

    @staticmethod
    def calculate_list(self, listvalues):
        """
        Calculate workload-profile strength
        """
        del listvalues[0]
        number = int(listvalues[0])
        calculation_algorithm = listvalues[1]
        variation = int(listvalues[2])
        if calculation_algorithm == "g":
            strength_number = int(random.gauss(number, variation))
        elif calculation_algorithm == "r":
            strength_number = int(random.uniform(number, variation))
        return strength_number
```
return strength_number

@staticmethod
def create_number_of_scripts(self, number_of_scripts_to_make, executable_string):
    """
    Generate a given number of test executable string from the generated strength to webuse.
    :param number_of_scripts_to_make:
    :type number_of_scripts_to_make:
    :param executable_string:
    :type executable_string:
    :return:
    :rtype:
    """
    if number_of_scripts_to_make <= 0:
        scriptlist = []
        number_to_run = number_of_scripts_to_make * (-1)
        for i in range(0, number_to_run):
            scriptlist.append(executable_string)
        return scriptlist
    else:
        scriptlist = []
        for i in range(0, number_of_scripts_to_make):
            scriptlist.append(executable_string)
        return scriptlist

@staticmethod
def create_httperf_string(self, ip, strength, executable_string_start):
    """
    Generate executable String for Httperf
    :param ip:
    :type ip:
    :param strength:
    :type strength:
    :param executable_string_start:
    :type executable_string_start:
    :return:
    :rtype:
    """
    if strength <= 0:
        new_strength = strength * (-1)
        scriptlist = []
        full_exec_string = executable_string_start + " " + ip + " " + str(new_strength)
        scriptlist.append(full_exec_string)
        return scriptlist
    else:
        scriptlist = []
        full_exec_string = executable_string_start + " " + ip + " " + str(strength)
        scriptlist.append(full_exec_string)
        return scriptlist
```
Appendix B

Workers

This appendix contains all the worker-scripts used in the prototype. The Worker code directory can be found on Github by using the following URL: https://github.com/s177437/uptime_challenge/tree/master/workers
Listing B.1: webuse_worker.py

```python
import pika
import ast
import subprocess
import time
import StringIO

__author__ = 'Stian Stroem Andersen'

class WebuseWorker:
    
    """This is the webuse - worker class"
    groupname = ""

    def fetch_job_from_queue(self):
        """Listen to the workqueue, perform a job and return the report."
        try:
            credentials = pika.PlainCredentials('USER', 'PASSWORD')
            connection = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', credentials))
            channel = connection.channel()
            channel.queue_declare(queue='webuseq')
            method_frame, header_frame, body = channel.basic_get(queue='webuseq')
            if method_frame.NAME == 'Basic.GetEmpty':
                connection.close()
            else:
                channel.basic_ack(delivery_tag=method_frame.delivery_tag)
                self.reply_to_master(body)
        except AttributeError:
            print "No content"
            connection.close()
        except pika.exceptions.ConnectionClosed:
            print "You get Connection Closed"
            continue

    def do_job(self, command):
        """Execute received job and convert the result to a report."
        try:
            dict = ast.literal_eval(command)
            dict.update({'Worker': self.get_host_name()})
            dict.update({'group': self.get_group_name()})
            return str(dict)
        except SyntaxError:
            print "This is not a dict"
            dict = self.convert_output_to_dict(command)
            dict.update({'Worker': self.get_host_name()})
            dict.update({'group': self.get_group_name()})
            dict.update({'check_timestamp': time.time()})
            return str(dict)

    def reply_to_master(self, content):
        """Post report to the queues"
        dict = ast.literal_eval(content)
        job = dict.values()[0]
        self.set_group_name(dict.keys()[0][0])
        message = self.do_job(job)
```

credentials = pika.PlainCredentials('USER', 'PASSWORD')
connection = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', credentials))
channel = connection.channel()
channel.queue_declare(queue='webusereportq')
channel.basic_publish(exchange='', routing_key='webusereportq', body=message)
connection.close()

@staticmethod
def get_command_output(self, command):
    """
    Execute bash command and save output
    :param command :
    :type command :
    :return :
    :rtype :
    """
    p = subprocess.Popen(command, stdout=subprocess.PIPE, shell=True)
    (output, error) = p.communicate()
    return output

@staticmethod
def convert_output_to_dict(self, content):
    """
    Convert test output to a dictionary
    :param content :
    :type content :
    :return :
    :rtype :
    """
    lower_content = content.lower()
    buf = io.StringIO(lower_content.strip())
    templist = []
    list = []
    reportdict = {}
    for i in buf.readlines():
        if "\n" in templist:
            templist = []
        templist.append(i)
        if "<html\n" in templist:
            templist.index("<html\n") + 1
        if "ok\t\n" in templist:
            templist = []
            templist.append(i)
    for i in templist:
        content = i.replace("\n", "")
        if content != "":
            list.append(content)
    for i in list:
        j = i.split("\t")
        if len(j) == 1:
            reportdict.update({"message": j[0]})
        else:
            reportdict.update({j[0]: "".join(j[1:len(j)])})
    reportdict.update({"status": statusmessage})
    return reportdict

@staticmethod
def run_command(self, command):
    """
    Execute bash command in python
    :param command :
    :type command :
    :return :
    :rtype :
    """
    subprocess.call(command, shell=True)

def get_host_name(self):
    """
    return the hostname of the worker
    """
    return

@staticmethod
def get_command_output(self, command):
    """
    Execute bash command
    :param command :
    :type command :
    :return :
    :rtype :
    """
    p = subprocess.Popen(command, stdout=subprocess.PIPE, shell=True)
    (output, error) = p.communicate()
return self.get_command_output("hostname").strip("\n")

def set_group_name(self, name):
    
        Set groupname to the report.
        
    :param name:
    :type name: 
    :return: 
    :rtype: 
    
    self.groupname = name

def get_group_name(self):
    
        Return the groupname to the report
        
    :return: 
    :rtype: 
    
    return self.groupname

worker = WebuseWorker()
worker.fetch_job_from_queue()
import pika
import ast
import subprocess
import time
import StringIO
__author__ = 'Stian Stroem Andersen

class HttperfWorker():
    ""
    This is the httperf - worker class
    ""
    groupname = ""

    def fetch_job_from_queue(self):
        ""
        Listen to the workqueue, perform a job
        and return the report.
        :return:
        :rtype:
        ""
        while 1:
            time.sleep(2)
            try:
                credentials = pika.PlainCredentials('USER', 'PASSWORD')
                connection = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', credentials))
                channel = connection.channel()
                channel.queue_declare(queue='httperfq')
                method_frame, header_frame, body = channel.basic_get(queue='httperfq')
                if method_frame.NAME == 'Basic.GetEmpty':
                    channel.basic_ack(delivery_tag=method_frame.delivery_tag)
                else:
                    print 'Received job:', body
                    body, "starting job to \""
                    "report"
                    connection.close()
            except AttributeError:
                print "No content"
            except pika.exceptions.ConnectionClosed:
                print " You get Connection Closed"
            continue
        self.reply_to_master(body)
        except AttributeError:
            print "No content"
        connection.close()
    def do_job(self, command):
        ""
        Execute received job
        :param command:
        :type command:
        :return:
        :rtype:
        ""
        outputdata = self.getcommandoutput(command)
        try:
            dict = ast.literal_eval(outputdata)
            dict.update({'worker': self.get_host_name()})
            dict.update({'group': self.get_group_name()})
            return str(dict)
        except SyntaxError:
            print 'This is not a dict'
            dict = self.convert_output_to_dict(outputdata)
            dict.update({'worker': self.get_host_name()})
            dict.update({'group': self.get_group_name()})
            dict.update({'check_timestamp': time.time()})
            return str(dict)
        self.reply_to_master(self, content):
            ""
            Port report to report-queue
            :param content:
            :type content:
            :return:
            :rtype:
            ""
            dict = ast.literal_eval(content)
            job = dict.values()[0]
            self.set_group_name(dict.keys()[0])
            message = self.do_job(job)
            credentials = pika.PlainCredentials('USER', 'PASSWORD')
            connection = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', credentials))
            channel = connection.channel()
            channel.queue_declare(queue='reportq')
            method_frame, header_frame, body = channel.basic_get(queue='reportq')
            if method_frame.NAME == 'Basic.GetEmpty':
                channel.basic_ack(delivery_tag=method_frame.delivery_tag)
            else:
                print 'Received job:', body
                body, "starting job to \""
                "report"
                connection.close()
`USER`, `PASSWORD`

```python
collection = pika.BlockingConnection(pika.ConnectionParameters('IP', 5672,
 credentials))
channel = collection.channel()
channel.queue_declare(queue='httpperfreportq')
channel.basic_publish(exchange='',
routing_key='httpperfreportq',
body=message)
collection.close()
```
def set_group_name(self, name):
    """Set groupname to the report.
    :param name: 
    :type name: 
    :return: 
    :rtype: 
    """
    self.groupname = name

def get_group_name(self):
    """Return the groupname to the report
    :return:
    :rtype:
    """
    return self.groupname

worker = HttperfWorker()
worker.fetch_job_from_queue()
import pika
import ast
import subprocess
import time
import StringIO

__author__ = 'Stian Stroem Andersen'

class LeeshoreWorker():
    """
    This is the leeshore - worker class.
    """
    groupname = ""

    def fetch_job_from_queue(self):
        """
        Listen to the workqueue, perform a job
        and return the report.
        :return:
        :rtype:
        """
        # do:
        # sloop = False
        while 1:
            time.sleep(2)
            try:
                credentials = pika.PlainCredentials('USER', 'PASSWORD')
                connection = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', credentials))
                channel = connection.channel()
                channel.queue_declare(queue='leeshoreq')
                method_frame, header_frame, body = channel.basic_get(queue='leeshoreq')
                if method_frame.NAME == 'Basic.GetEmpty':
                    connection.close()
                else:
                    channel.basic_ack(delivery_tag=method_frame.delivery_tag)
                    print 'Received job:',
                # print 'sloop', sloop
                try:
                    outputdata = self.getcommandoutput(command)
                except AttributeError:
                    print 'No content'
                    connection.close()
                except pika.exceptions.ConnectionClosed:
                    print 'You get Connection Closed'
                    continue
                try:
                    dict = ast.literal_eval(outputdata)
                    dict.update({'worker': self.get_host_name()})
                    dict.update({'group': self.get_group_name()})
                    return str(dict)
                except SyntaxError:
                    print 'This is not a dict'
                    dict = self.convert_output_to_dict(outputdata)
                    dict.update({'worker': self.get_host_name()})
                    dict.update({'group': self.get_group_name()})
                    return str(dict)
            except AttributeError:
                print 'No content'
                connection.close()
            except pika.exceptions.ConnectionClosed:
                print 'You get Connection Closed'
                continue
            except SyntaxError:
                print 'This is not a dict'

    def do_job(self, command):
        outputdata = self.getcommandoutput(command)
        try:
            dict = ast.literal_eval(outputdata)
            job = dict.values()[0]
            self.set_group_name(dict.keys()[0])
            message = self.do_job(job)
            print message
            credentials = pika.PlainCredentials('USER', 'PASSWORD')
            connection = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', credentials))
            channel = connection.channel()
            channel.queue_declare(queue='leeshoreq')
            method_frame, header_frame, body = channel.basic_get(queue='leeshoreq')
            if method_frame.NAME == 'Basic.GetEmpty':
                connection.close()
            else:
                channel.basic_ack(delivery_tag=method_frame.delivery_tag)
                print 'Received job:',
        except AttributeError:
            print 'No content'
            connection.close()
        except pika.exceptions.ConnectionClosed:
            print 'You get Connection Closed'
            continue

    def reply_to_master(self, content):
        try:
            dict = ast.literal_eval(content)
            job = dict.values()[0]
            self.set_group_name(dict.keys()[0])
            message = self.do_job(job)
            print message
            credentials = pika.PlainCredentials('USER', 'PASSWORD')
            connection = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', credentials))
            channel = connection.channel()
            channel.queue_declare(queue='leeshoreq')
            method_frame, header_frame, body = channel.basic_get(queue='leeshoreq')
            if method_frame.NAME == 'Basic.GetEmpty':
                connection.close()
            else:
                channel.basic_ack(delivery_tag=method_frame.delivery_tag)
                print 'Received job:',
        except AttributeError:
            print 'No content'
            connection.close()
@staticmethod
def get_command_output(self, command):
    p = subprocess.Popen(command,
                        stdout=subprocess.PIPE,
                        shell=True)
    (output, error) = p.communicate()
    return output

@staticmethod
def run_command(self, command):
    """Execute bash command in python
    :param command:
    :type command:
    :return:
    :rtype:
    """
    subprocess.call(command, shell=True)

def get_host_name(self):
    """Return the hostname of the worker
    :return:
    :rtype:
    """
    return self.get_command_output('hostname').strip('
')

def set_group_name(self, name):
    """Set groupname to the report.
    :param name:
    :type name:
    :return:
    :rtype:
    """
    self.groupname = name

def get_group_name(self):
    """Return the groupname to the report
    :return:
    :rtype:
    """
    return self.groupname

worker = LeeshoreWorker()
worker.fetch_job_from_queue()
Listing B.4: clerk_worker.py

```python
import pika
import ast
import subprocess
import time
import StringIO

__author__ = 'Stian Stroem Andersen'

class ClerkWorker():
    
    """This is the clerk-worker class."""
    groupname = ""

    def fetch_job_from_queue(self):
        """Listen to the workqueue, perform a job and return the report."""
        while 1:
            time.sleep(2)
            try:
                credentials = 
                pika.PlainCredentials('USERNAME', 'PASSWORD')
                connection = 
                pika.BlockingConnection( 
                    pika.ConnectionParameters( 
                        'rabbitmq', 5672, '/', credentials))
                channel = connection.channel()
                channel.queue_declare(
                    queue='clerkq')
                method_frame, header_frame, 
                body = channel.basic_get( 
                    'clerkq')
                if method_frame.NAME == 
                    'Basic.GetEmpty':
                    connection.close()
                else:
                    channel.basic_ack(
                        delivery_tag=method_frame.delivery_tag)
                    print "Received job ", 
                    body, "starting job to ", 
                    "reply"
                    connection.close()
                    self.reply_to_master(body)
                    print "No content"
                    connection.close()
                except pika.exceptions.ConnectionClosed:
                    print "You get Connection Closed"
                    continue
                except AttributeError:
                    print "No content"
                    connection.close()
                except pika.exceptions.ConnectionClosed:
                    print "You get Connection Closed"
                    continue
                else:
                    channel.basic_ack(
                        delivery_tag=method_frame.delivery_tag)
                    print "Received job ", 
                    body, "starting job to ", 
                    "reply"
                    connection.close()
                    self.reply_to_master(body)
                    print "No content"
                    connection.close()
                    except pika.exceptions.ConnectionClosed:
                        print "You get Connection Closed"
                        continue
            except AttributeError:
                print "No content"
                connection.close()
            except pika.exceptions.ConnectionClosed:
                print "You get Connection Closed"
                continue

    def do_job(self, command):
        """Process and tag the report"""
        try:
            dict = ast.literal_eval(outputdata)
            dict.update(
                {'worker': self.get_hostname()})
            dict.update(
                {'group': self.get_group_name()})
            return str(dict)
        except SyntaxError:
            print "This is not a dict"
            dict = self.convert_output_to_dict(outputdata)
            dict.update(
                {'worker': self.get_hostname()})
            dict.update(
                {'group': self.get_group_name()})
            return str(dict)

    def reply_to_master(self, content):
        """Post report on the report-queue"""
        dict = ast.literal_eval(content)
        job = dict.values()[0]
        self.set_group_name(dict.keys()[0])
        message = self.do_job(job)
        print message
        credentials = pika.PlainCredentials('USERNAME', 'PASSWORD')
```

---

1. `import pika` - Imports the `pika` module.
2. `import ast` - Imports the `ast` module for the `ast.literal_eval` function.
3. `import subprocess` - Imports the `subprocess` module for executing shell commands.
4. `import time` - Imports the `time` module for time-related operations.
5. `import StringIO` - Imports the `StringIO` module for creating an in-memory string object.
6. `__author__ = 'Stian Stroem Andersen'` - Sets the author of the code.
7. `class ClerkWorker():` - Defines a class named `ClerkWorker`.
8. `"""This is the clerk-worker class.""""` - A docstring describing the purpose of the class.
9. `groupname = ""` - Sets the initial group name to an empty string.
10. `def fetch_job_from_queue(self):` - Defines a method `fetch_job_from_queue`.
11. `"""Listen to the workqueue, perform a job and return the report.""""` - A docstring for the method.
12. `while 1:` - Starts an infinite loop.
13. `time.sleep(2)` - Pauses the loop for 2 seconds.
14. `try:` - Begins a try block.
15. `credentials = pika.PlainCredentials('USERNAME', 'PASSWORD')` - Sets the credentials for the RabbitMQ connection.
17. `channel = connection.channel()` - Creates a channel on the connection.
18. `channel.queue_declare(queue='clerkq')` - Declares a queue named 'clerkq'.
19. `method_frame, header_frame, body = channel.basic_get('clerkq')` - Gets the next message from the queue.
20. `if method_frame.NAME == 'Basic.GetEmpty':` - Checks if the message is an empty fetch.
21. `connection.close()` - Closes the connection if the message is empty.
22. `else:` - Enters the else block if the message is not empty.
23. `channel.basic_ack(delivery_tag=method_frame.delivery_tag)` - Acknowledges the received message.
24. `print "Received job ", body, "starting job to ", "reply"` - Logs the received message details.
25. `connection.close()` - Closes the connection.
27. `"""Process and tag the report""""` - A docstring for the method.
28. `try:` - Begins a try block.
29. `dict = ast.literal_eval(outputdata)` - Attempts to parse the output data as a dictionary.
30. `dict.update({'worker': self.get_hostname()})` - Updates the dictionary with the current worker's hostname.
31. `dict.update({'group': self.get_group_name()})` - Updates the dictionary with the current group's name.
32. `return str(dict)` - Returns the formatted dictionary.
33. `except SyntaxError:` - Begins an except block for `SyntaxError`.
34. `print "This is not a dict"` - Logs an error message when the output data cannot be parsed.
35. `dict = self.convert_output_to_dict(outputdata)` - Attempts to parse the output data as a dictionary using a custom converter.
36. `dict.update({'worker': self.get_hostname()})` - Updates the dictionary with the current worker's hostname.
37. `dict.update({'group': self.get_group_name()})` - Updates the dictionary with the current group's name.
38. `return str(dict)` - Returns the formatted dictionary.
39. `def reply_to_master(self, content):` - Defines another method `reply_to_master`.
40. `"""Post report on the report-queue""""` - A docstring for the method.
41. `dict = ast.literal_eval(content)` - Attempts to parse the input content as a dictionary.
42. `job = dict.values()[0]` - Extracts the job's value from the dictionary.
43. `self.set_group_name(dict.keys()[0])` - Sets the current group's name from the dictionary.
44. `message = self.do_job(job)` - Calls the `do_job` method with the job.
45. `print message` - Logs the returned message.
46. `credentials = pika.PlainCredentials('USERNAME', 'PASSWORD')` - Sets the credentials for the RabbitMQ connection.
```
connection = pika.BlockingConnection(
    pika.ConnectionParameters('rabbitmq',
                             5672, '/',
                             credentials))
channel = connection.channel()
channel.queue_declare(queue='clerk_reportq')
channel.basic_publish(exchange='',
                      routing_key='clerk_reportq',
                      body=message)
connection.close()

@staticmethod
def getcommandoutput(self, command):
    """
    Execute bash - command and save output
    :param command :
    :type command :
    :return :
    :rtype :
    """
    p = subprocess.Popen(command,
                         stdin=subprocess.PIPE,
                         shell=True)
    (output, error) = p.communicate()
    return output

@staticmethod
def convert_output_to_dict(self, content):
    """
    Convert test result to a dictionary
    :param content :
    :type content :
    :return :
    :rtype :
    """
    reportdict = {}
    reportdict.update({
        "Message":
        "Something ",
        "went very ",
        "wrong ",})
    return reportdict

@staticmethod
def run_command(self, command):
    """
    Execute bash command in python
    :param command :
    :type command :
    :return :
    :rtype :
    """
    subprocess.call(command, shell=True)

def get_hostname(self):
    """
    return the hostname of the worker
    :return :
    :rtype :
    """
    return self.getcommandoutput("hostname").strip("\n")

def set_group_name(self, name):
    """
    Set groupname to the report.
    :param name :
    :type name :
    :return :
    :rtype :
    """
    self.groupname = name

def get_group_name(self):
    """
    Return the groupname to the report
    :return :
    :rtype :
    """
    return self.groupname

worker = ClerkWorker()
worker.fetch_job_from_queue()
import sys
from purser import *
import pika
import ast
import subprocess
import time
import StringIO

__author__ = 'Stian Stroem Anderssen'

class PurserWorker():
    """
    This is the purser - worker class.
    """
    groupname = ""

    def fetch_job_from_queue(self):
        """
        Listen to the workqueue, perform a job
        and return the report.
        """
        while 1:
            time.sleep(2)
            try:
                credentials = pika.PlainCredentials('USER', 'PASSWORD')
                connection = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', credentials))
                channel = connection.channel()
                channel.queue_declare(queue='purserq')

                method_frame, header_frame, body = channel.basic_get(queue='purserq')
                if method_frame.NAME == 'Basic.GetEmpty':
                    connection.close()
                else:
                    channel.basic_ack(delivery_tag=method_frame.delivery_tag)
                    print "Received job : ",
                    body, "starting job to 

        "reply"
        connection.close()
        self.reply_to_master(body)
        except AttributeError:
            print "No content"
            connection.close()
        except pika.exceptions.ConnectionClosed:
            print "You get Connection Closed"
            continue
        else:
            p = Purser()
            filepath = variabledict['filepath'] + \
                variabledict['ip'] + '/' + \
                variabledict['file']
            dict = p.run_purser(variabledict['ip'], \
                variabledict['file'], filepath, \
                variabledict['sentance'])
            dict.update({'worker': self.get_host_name()})
            dict.update({'group': self.get_group_name()})
            p.delete_directory("root/uptime_challenge_master" / worker/
                variabledict['ip'])
            return str(dict)

    @staticmethod
    def reply_to_master(self, content):
        """
        Send the report back to the report-queue
        """
        outerdict = ast.literal_eval(content)
        innerdict = outerdict.values()[0]
        self.set_group_name(outerdict.keys()[0])

        def do_job(self, variabledict):
            """
            Execute the purser-test by calling the
            purser check
            """
            :param variabledict:
            :type variabledict:
            :return:
            :rtype:
            p = Purser()
            filepath = variabledict['filepath'] + \
                variabledict['ip'] + '/' + \
                variabledict['file']
            dict = p.run_purser(variabledict['ip'], \
                variabledict['file'], filepath, \
                variabledict['sentance'])
            dict.update({'worker': self.get_host_name()})
            dict.update({'group': self.get_group_name()})
            p.delete_directory("root/uptime_challenge_master" / worker/
                variabledict['ip'])
            return str(dict)
message = self.do_job(innerdict)
credentials = pika.PlainCredentials('USER', 'PASSWORD')
connection = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', credentials))
channel = connection.channel()
channel.queue_declare(queue='purser_report_q')
channel.basic_publish(exchange='', routing_key='purser_report_q', body=message)
connection.close()

@staticmethod
def get_command_output(self, command):
    """
    Execute bash command and save output
    """
    p = subprocess.Popen(command, stdout=subprocess.PIPE, shell=True)
    (output, error) = p.communicate()
    return output

@staticmethod
def run_command(self, command):
    """
    Execute bash command in python
    """
    subprocess.call(command, shell=True)

def get_host_name(self):
    """
    return the hostname of the WebUseWorker
    """
    return self.get_command_output('hostname').strip('
')

def set_group_name(self, name):
    """
    Set groupname of the report.
    """
    self.groupname = name

def get_group_name(self):
    """
    Return the groupname to the report
    """
    return self.groupname

worker = PurserWorker()
worker.fetch_job_from_queue()
Appendix C

Purser-script

This appendix contains the Purser-script used in the tests. Some of the tests used in the prototype is written by Kyrre Begnum and are not included as part of the Appendices.
Listing C.1: purser.py

```python
import time
import httplib2
from BeautifulSoup import BeautifulSoup, SoupStrainer
import subprocess
import urllib
import os

__author__ = 'Stian Stroem Anderssen'

class Purser:
    __doc__ = 'This class represents the purser - check.
    
    def return_download_time(self, hostname):
        """Download the Bookface site using wget
        :param hostname:
        :type hostname:
        :return:
        :rtype:
        """
        starttime = time.time()
        output = ""
        output = self.get_command_output("wget -t 2 -T 5 -P /root/uptime_challenge_master/worker/-p -q http://" + hostname + "/index.php")
        if "No route" in output:
            return 0
        else:
            return time.time() - starttime

    def delete_directory(self, directory_name):
        """Delete a given directory using bash rm
        :param directory_name:
        :type directory_name:
        :return:
        :rtype:
        """
        self.runcommand("rm -rf " + directory_name)

    def check_if_file_exists(self, filepath, filename):
        """Check if a file exists in a directory.
        :param filepath:
        :type filepath:
        :param filename:
        :type filename:
        :return:
        :rtype:
        """
        ls_check = self.get_command_output("ls -t")
        if filename in ls_check:
            return "True"
        else:
            return "False"

    @staticmethod
    def check_if_word_exists(self, filepath, ip, sentence):
        """Check if a word exists on the frontpage
        :param filepath:
        :type filepath:
        :param ip:
        :type ip:
        :param sentence:
        :type sentence:
        :return:
        :rtype:
        """
        data = urllib.urlopen("http://" + ip + "/index.php")
        lines = []
        for line in data:
            lines.append(line)
        notfound = "Not found"
        for line in lines:
            if sentence in line:
                return "Word is found in " + filepath + " at line " + str(lines.index(line))
        return "Word not found in " + filepath

    @staticmethod
    def get_command_output(self, command):
        """Get command output from a bash command
        :param command:
        :type command:
        :return:
        :rtype:
        """
```

---

```
```python
p = subprocess.Popen(command,
                      stdin=subprocess.PIPE,
                      stdout=subprocess.PIPE,
                      shell=True)
(output, error) = p.communicate()
return output

@staticmethod
def get_response_code(self, sitename):
    try:
        return urllib.urlopen("http://" + sitename).getcode()
    except IOError:
        return 500

@staticmethod
def run_command(self, command):
    subprocess.call(command, shell=True)

def run_purser(self, ip, filename, filepath, sentance):
    result = {}
    file_found = "File not found"
    time_used_to_download = 
    self.return_download_time(ip)
    if time_used_to_download == 0:
        result["File exists"] = "True"
    result["Check timestamp"] = time.time()
    result["Lookup status"] = "Cannot find "
    file_exists_in_directory_tree = self.check_if_file_exists(filepath, filename)
    sentance_found = "Word not found"
    if file_exists_in_directory_tree == 
        self.check_if_word_exists(filepath, ip, sentance)
    result["Http response code"] = 500
    result["Time used to download"] = 
    time_used_to_download
    return result
```

"File exists" = \n  file_exists_in_directory_tree
result["File"] = filename
result["Hostname"] = ip
result["Check timestamp"] = time.time()
result["Lookup status"] = sentence_found
if result["File exists"] == "True" and "not found" in \\n  sentence_found:
  result["Test status"] = "Partial OK"
else:
  result["Test status"] = "Not Approved"
def download_url(self, hostname):
  """
  Download a given url by using inbuilt python tools
  param hostname:
  :type hostname:
  :rtype:
  """
  self.runcommand("\n  "mkdir " + \\
  "/root/uptime_challenge_master" \\
  "/worker/" + hostname)
  starttime = time.time()
  http = httplib2.Http()
  status, response = http.request( \\
  "http://" + hostname)
  urllib.urlretrieve(" \\
  "/http://" + hostname + \\
 ="/stylesheet.css",
  "/root/uptime_challenge_master" \\
  "/worker/" + hostname + \\
 ="/stylesheet.css")
  urllib.urlretrieve( \\
  "http://" + hostname + "/index.php", \\
  "/root/uptime_challenge_master" \\
  "/worker/" + hostname + "/index.php")
  index = 0
  for link in BeautifulSoup(response, \\
  parseOnlyThese=SoupStrainer( \\
  'img')):
    filename = \\
    "/root/uptime_challenge_master" \\
    "/worker/" + hostname + "/picture" \\
    + str(index)
    urllib.urlretrieve(link['src'], \\
    filename)
    index += 1
  return time.time() - starttime
Interpreter API

This chapter contains the code needed for the interpreter API.
import Pyro4
from DbConfig import *
import logging
import couchdb

__author__ = 'Stian Stroem Andersen

class Apilistener():
    ""
    This class distributes an API to the
    manager that can be used to communicate
    with the database.
    ""
    logging.basicConfig(
        filename='/var/log/interpreter.log',
        level=logging.DEBUG)

    @staticmethod
    def fetch_config(self, teacher):
        ""
        This function creates a new config
        object and calls the DbConfig
        fetchconfig class and returns a config
        object
        ""
        :return: config
        :rtype:
        :param accountlist:
        :type accountlist:
        :return:
        :rtype:
        :param dbservername:
        :type dbservername:
        :param dbname:
        :type dbname:
        :param key:
        :type key:
        :param value:
        :type value:

    def post_report_to_database(self, reportdict):
        ""
        This function posts a report to the
        database. A dictionary is sent as a
        parameter.
        ""
        :param reportdict:
        :type reportdict:
        :return:
        :rtype:
        :param dbservername:
        :type dbservername:
        :param dbname:
        :type dbname:
        :param key:
        :type key:
couch = couchdb.Server("http://USER:PASSWORD@couchdb:5984/")
db = couch[dbname]

map_fun = ''' function (doc) {
    if(doc.''' + key + ''' == ''' + value + '''){
        emit(doc.''' + keytoupdate + ''', doc._id);
    }
}'''

result = db.query(map_fun)
for element in result:
    documentid = element["value"]
doc = db[documentid]
doc[keytoupdate] = valuetoupdate
db[documentid] = doc

@staticmethod
def update_balance(self, dbname, groupname, value):
    """This test function updates the existing balance of a user. When a job is performed, the results are parsed, and this function is called to update the balance of the user.
    :param dbname: string
    :param groupname: string
    :param value: numeric
    :return: numeric
    """

couch = couchdb.Server("http://USER:PASSWORD@couchdb:5984/")
db = couch[dbname]

map_fun = ''' function (doc) {
    if(doc.group == ''' + groupname + '''){
        emit(doc.group, doc._id);
    }
}'''

result = db.query(map_fun)
documentid = """
for element in result:
documentid = element["value"]
doc = db[documentid]
try:
    oldbalance = doc["Balance"]
balance = oldbalance + value
    doc["Balance"] = balance
except KeyError:
    logging.info("groupname = " + groupname + " has no previous " + "balance", creating = "balance")
    doc["Balance"] = value
db[documentid] = doc
@staticmethod
def get_ip_from_user(self, username):
    """Return IP-address of a bookface-site to a user.
    :param username:
    :type username:
    :return:
    :rtype:
    """
    couch = couchdb.Server("http://USER:PASSWORD@couchdb:5984/")
db = couch["accounts"]
map_fun = '''function(doc) {
    if(doc.group == "" + username + "")
        emit("ipaddress",doc.ipaddress);
}'''
result = db.query(map_fun)
returnvalues = {}
for element in result:
    returnvalues.update({element["key"]:element["value"]})
return returnvalues

@staticmethod
def get_user_config(self, username, dbserver):
    """Return all config for a config-account.
    :param username:
    :param dbserver:
    :type username:
    :type dbserver:
    :return:
    :rtype:
    """
    couch = couchdb.Server("http://USER:PASSWORD@" + dbserver
        + ":5984/")
db = couch["accounts"]
map_fun = '''function(doc) {
    if(doc.group == "" + username + "")
        emit("userconfig",doc);
}'''
result = db.query(map_fun)
returnvalues = {}
for element in result:
    returnvalues.update({element["key"]:element["value"]})
return returnvalues["userconfig"]

daemon = Pyro4.Daemon("IP/hostname")
n = Pyro4.locateNS()
uri = daemon.register(Apilistener)
n.register("interpreter", uri)
logging.info("Ready " + str(uri))
daemon.requestLoop()}
Listing C.3: dbconfig.py

```python
import pika
import couchdb
import logging

__author__ = 'Stian Stroem Anderssen

class DbConfig():
    """
    This module is the logic class for the Config class in the couchdb. Some of the functions are a little outdated, because it was decided to use an API to handle the config interaction with the database.
    """
    logging.basicConfig(filename='/var/log/interpreter.log', level=logging.DEBUG)

    @staticmethod
    def callback(channel, method, properties, body):
        print "Received message .... " + body

    @staticmethod
    def get_config(self, teacher, dbname):
        couch = couchdb.Server('http://USER:PASSWORD@couchdb:5984/')
        db = couch[dbname]
        map_fun = ''' function (doc) {
            if( doc.teacher == "' + teacher + '") {
                emit(doc.teacher, doc._id);
            }
        } '''
        result = db.query(map_fun)
        documentid = """
        for element in result:
            documentid = element["value"]
        for element in result:
            doc = db[element["name"]]
            del doc["_id"]
            del doc["_rev"]
            del doc["teacher"]
            configdict = {}  
            for key, value in doc.items():
                configdict.update({key: value})
        return configdict

    @staticmethod
    def fetch_elements_from_couchdb(self):
        """
        This function fetches the config element from the couchdb, and sends it back to the manager as as a dictionary
        :rtype: dict:
        """
        couch = couchdb.Server('http://USER:PASSWORD@couchdb:5984/')
        fieldlist = []
        configdict = {}
        for element in db:
            fieldlist.append(element)
            for name in fieldlist:
                configdict.update(
                    {name: db[name][name]})
        return configdict

    def send_config_to_queue(self, quename):
        """
        This function sends the config dict to the configqueue. OUTDATED.
        :param quename:
        :type quename:
        :return:
        :rtype:
        """
        credentials = pika.PlainCredentials('guest', 'guest')
        connection = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', credentials))
        channel = connection.channel()
        channel.queue_declare(queue=quename)
        content = str(self.fetch_elements_from_couchdb())
        channel.basic_publish(exchange='', routing_key=quename, body=content)
        logging.info(content)
        connection.close()
```

Listing C.4: dblogic.py

import couchdb
from Course import *
from DbAccount import *
from Group import *
from Teacher import *
from collections import OrderedDict
import ast
import pika
import logging

__author__ = 'Stian Stroem Andersen

class DbLogic():
    ""
    This module is the logic for the interpreter. Here, the account object is received and created before it is written to the database.
    ""

    logging.basicConfig(
        filename='/var/log/interpreter.log',
        level=logging.DEBUG
    )

    def set_queue_content(self, value):
        ""
        A function to set the content to be sent to the queue for later usage
        ""
        self.queuecontent = value

    def get_queue_content(self):
        ""
        Return the queue content
        ""
        return self.queuecontent

    @staticmethod
    def build_account_object(self, received_list):
        ""
        This function builds an account object which can be written directly to the database. The function received a list containing the teacher, course, groups and students plus semester.
        Example list style: [{"teacher": ["Teachername"], "course": ["Coursename"]}, ...]
        ""
        account = DbAccount()
teacher = Teacher()
course = Course()
groupdict = {}

        for i in received_list:
            for key, value in i.items():
                if "teacher" in key:
                    teacher.set_name(value[0])
                elif "semester" in key:
                    teacher.set_semester(value[0])
                elif "course" in key:
                    course.set_coursename(value[0])
                elif "group" in key:
                    for groupkey, memberlist in value.items():
                        group = Group()
                        groupdict.update({groupkey: memberlist})
                        group.set_groupname(groupkey)
                        group.set_students(memberlist)
                        if any(x.get_groupname() == groupkey for x in course.get_groups()):
                            logging.info(groupkey + " exists in the object from before")
                        else:  # course.set_group_list(group)
                            groupdict.update({groupkey: memberlist})
                            group = Group()
                            course.set_group_list(group)
                            teacher.set_course(course)
                            account.set_teacher(teacher)
                            return account
            if any(x.get_groupname() == groupkey for x in course.get_groups()):
                logging.info("groupkey + " exists in the object from before")
            else:
                for x in course.get_groups():
                    groupdict.update({groupkey: memberlist})
                    course.set_group_list(group)
                    group = Group()
                    course.set_group_list(group)
                    teacher.set_course(course)
                    account.set_teacher(teacher)
                    return account
        return account

    def create_group_account(self, userlist):
        ""
        ""
Function to post the account object to the database.

Function to see if a user exist in the database from before.

Function to save the report objects in the couchdb.

```python
couch = couchdb.Server('http://USER:PASSWORD@couchdb:5984/')
teacheraccount = self.build_account_object(userlist)
db = couch['accounts']
logging.info(teacheraccount.get_teacher().get_course().get_groups())

for i in teacheraccount.get_teacher().get_course().get_groups():
    accountdict = OrderedDict()
    accountdict.update({'teacher': teacheraccount.get_teacher().get_name()})
    accountdict.update({'course': teacheraccount.get_teacher().get_course().get_coursename()})
    accountdict.update({'semester': teacheraccount.get_teacher().get_semester()})
    accountdict.update({'group': i.get_groupname()})
    accountdict.update({'members': i.get_students()})
    try:
        db.save(accountdict)
    except couchdb.http.PreconditionFailed:
        logging.info('Document saved in the database')
```
Listing C.5: dbaccount.py

```python
__author__ = 'Stian Stroem Andersen'

class DbAccount:
    
    This is a module for the DbAccount object

    teacher = None

    def get_teacher(self):
        
        Function to return teacher.

        :return teacher:
            :rtype:
        
        return self.__teacher

    def set_teacher(self, value):
        
        Function to set the teacher object

        :param value:
            :type value:
        
        self.__teacher = value

Listing C.6: group.py

```python
__author__ = 'Stian Stroem Andersen'

class Group:
    
    This function creates a group object

    groupname = ""
    students = []

    def get_groupname(self):
        
        Get the name of the group

        :return groupname:
            :rtype:
        
        return self.__groupname

    def get_students(self):
        
        Get the name of the students in a list.

        :return:
            :rtype:
        
        return self.__students

    def set_groupname(self, value):
        
        Set the groupname of the group

        :param value:
            :type value:
        
        self.__groupname = value

    def set_students(self, value):
        
        Set the studentnames.

        :param value:
            :type value:

        self.__students = value

```

__author__ = 'Stian Stroem Andersen'
Listing C.7: teacher.py

```python
__author__ = 'Stian Strøm Andersen'

class Teacher:
    name = 
    course = 
    semester = 

    def get_name(self):
        return self.__name

    def set_semester(self, s):
        self.semester = s

    def get_semester(self):
        return self.semester

    def set_name(self, value):
        self.__name = value

    def get_course(self):
        return self.__course

    def set_course(self, value):
        self.__course = value
```

Listing C.8: course.py

```python
__author__ = 'Stian Strøm Andersen'

class Course:

    coursename = ''
    groups = []

    def get_coursename(self):
        return coursename

    def set_coursename(self, value):
        self.__coursename = value

    def get_groups(self):
        return groups

    def set_group_list(self, group):
        self.groups.append(group)
```
Appendix D

Puppet Manifests and Dockerfile

This appendix contains the Puppet manifests, and Dockerfile used to automate the Docker deployment of the worker instances.
Listing D.1: init.pp

```plaintext
class uptime {
  include uptime::install
}
```

Listing D.2: install.pp

```plaintext
class uptime::install {
  include 'docker'
  docker::image ('stianstrom/purserworker':
    image_tag => 'latest'
  )
  docker::run ('webuseworker1':
    image => 'stianstrom/webuseworker',
  )
  docker::run ('webuseworker2':
    image => 'stianstrom/webuseworker',
  )
  docker::run ('purserworker1':
    image => 'stianstrom/purserworker',
  )
  docker::run ('purserworker2':
    image => 'stianstrom/purserworker',
  )
  docker::run ('httperfworker1':
    image => 'stianstrom/httperfworker',
  )
  docker::run ('httperfworker2':
    image => 'stianstrom/httperfworker',
  )
}
```

Listing D.3: nodes.pp

```plaintext
node 'worker30.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker29.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker28.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker27.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker26.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker25.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker24.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker23.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker22.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker21.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker20.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker19.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker18.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker17.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker16.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker15.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker14.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker13.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker12.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker11.openstacklocal' {
  include uptime
}
```

```plaintext
node 'worker10.openstacklocal' {
  include uptime
}
```
Listing D.4: site.pp

import 'nodes.pp'

Listing D.5: Dockerfile

FROM ubuntu:14.04
MAINTAINER Stian Strom Andersen <s177437@hioa.no>
RUN apt-get update && apt-get install -y openssh-server python-pip git-core httpperf libwww-perl
RUN pip install pika==0.9.8
RUN mkdir -p /root/.ssh/
# Add ssh-key
ADD id_rsa /root/.ssh/id_rsa
RUN chmod 700 /root/.ssh/id_rsa
RUN touch /root/.ssh/known_hosts
RUN echo "Host github.com
StrictHostKeyChecking no"
>> /root/.ssh/config
# Close the uptime_challenge repository from Github using ssh-keys
RUN cd /root/ && git clone git@github.com:s177437/uptime_challenge_master.git
RUN sed "s@session \s* required \s* pam_loginuid.so@session optional pam_loginuid.so@g" -i /etc/pam.d/sshd
# Script containing an executable path to the worker-script
ADD ./startup.sh /opt/startup.sh
# Execute the worker
CMD ["/bin/bash", "/opt/startup.sh"]
Appendix E

CouchDB database script

This script was used by the teacher to add accounts, and to reset the balance for the groups each week. This Appendix contains the script and the help-output printed from the script.

Listing E.1: couch_modify.py help output

```python
usage: couch_modify.py [-h] [-a ADDUSERS]
                      [-modifyentry parameters [parameters ...]] [--stop]

EXAMPLE

python couch_modify.py -modifyentry couchdb testaccounts group reportlab
Balance 0

This updates the balance of the user reportlab located in the
testaccounts database on the CouchDB server

optional arguments:
    -h, --help            show this help message and exit
    -a ADDUSERS, --addusers ADDUSERS
                           Add users from a list of usernames
    -modifyentry parameters [parameters ...]
                           Modify an entry in any database, need to be given
                           in the following order: dbserver--hostname
dbname searchkey searchvalue keytoupdate
                           valuetoupdate
    --stop                Exit program
```

187
import argparse
import couchdb
import textwrap
from argparse import RawDescriptionHelpFormatter
import sys

class AddUsers():
    def readFile(self, filename):
        ""
        Read property file containing usernames to be added
        ""
        return [line.rstrip('\n') for line in open(filename)]

    def createUserObject(self, filename):
        ""
        Create a from a file and attach a set of attributes needed for the user
        ""
        couch = couchdb.Server("http://USER:PASSWORD@couchdb:5984/")
db = couch['accounts']
users = self.readFile(filename)
for user in users:
    attributes = {
        "enabled": "1",
        "Balance": 0,
        "bonus": 5,
        "bonus_time_cutoff": 5,
        "course": "infrastructure",
        "file": "transsine.dat",
        "filepath": "/root/uptime_challenge_master/LeeshoreWorker/",
        "group": "user",
        "hourly_rate": 3,
        "ipaddress": "IP",
        "last_check": 1462109486.31,
        "members": ["Tila", "Stine", "Stian"],
        "offset": 25,
        "partial_ok_punishment_decrease": 0.1,
        "semester": "A15",
        "Sentence": "Users:",
        "teacher": "Kyrre")
    }
    print attributes
db.save(attributes)

    def modify_key(self, dbservername, dbname, key, value, keytoupdate, valuetoupdate):
        ""
        This function updates any given key, value in the database. A unique key, value is passed as arguments to the function. This is necessary to get the correct document back from the database. The document is then updated with a given key, value.
        ""
        couch = couchdb.Server("http://USER:PASSWORD@couchdb:5984/")
db = couch[dbname]
map_fun = ''' function (doc) {
    if(doc."{}" == "{}"){
        emit(doc."{}", doc._id);
    }'''
result = db.query(map_fun)
for element in result:
    documentid = element[value]
doc = db[documentid]
doc[keytoupdate] = valuetoupdate
db[documentid] = doc

parser = argparse.ArgumentParser()
    formatter_class=argparse.RawDescriptionHelpFormatter,
    description=textwrap.dedent('''
EXAMPLE
-------------
python couch_modify.py -modifyentry couchdb
testaccounts group reportlab Balance 0

This updates the balance of the user reportlab located in the testaccounts database on the CouchDB server
-------------
'''))

parser.add_argument('-a', '--addusers',
    help='Add users from a list of usernames',
    required=False)

parser.add_argument('-modifyentry',
    help='Modify an entry in any database. need to be given in the following order: dbserver - hostname dbname searchkey searchvalue keytoupdate valuetoupdate',
    nargs='+',
    metavar='parameters',
    required=False)

parser.add_argument('--stop', help='Exit program',
    required=False,
    action='store_true')

args = parser.parse_args()
run = AddUsers()

if args.addusers:
    print(args)
    run.createUserObject(args.addusers)

elif args.modifyentry:
    run.modify_key(args.modifyentry[0],
        args.modifyentry[1],
        args.modifyentry[2],
        args.modifyentry[3],
        args.modifyentry[4],
        int(args.modifyentry[5]))

elif args.stop:
    print('Exiting program')
    sys.exit(0)
Appendix F

Physical infrastructure Overview

This appendix contains the physical infrastructure configured during the project. This provides a detailed view of all instances running the uptime challenge.
Figure F.1: A figure showing the full infrastructure running at Alto Cloud at HiOA, and the OpenStack Cloud running at NTNU Gjøvik. The numbers inside the workers represents Docker-instances.