Software Agent-Based Cost Model Calculations for Distributed Server Environments

Níels Bjarnason

Faculty of Industrial Engineering, Mechanical Engineering and Computer Science
University of Iceland
2012

Software Agent-Based Cost Model Calculations for Distributed Server Environments

Níels Bjarnason

60 ECTS thesis submitted in partial fulfillment of a
Magister Scientiarum degree in Software Engineering

Advisor(s)
Helmut Wolfram Neukirchen
Snorri Agnarsson

Faculty Representative
Jóhann Pétur Malmquist

Faculty of Industrial Engineering, Mechanical Engineering and Computer Science
School of Engineering and Natural Sciences
University of Iceland
Reykjavik, September 2012

Software Agent-Based Cost Model Calculations for Distributed Server Environments
Agent-Based Cost Model Calculations
60 ECTS thesis submitted in partial fulfillment of a *Magister Scientiarum* degree in Software Engineering

Faculty of Industrial Engineering, Mechanical Engineering and Computer Science
School of Engineering and Natural Sciences
University of Iceland
Hjarðarhaga 2-6
107, Reykjavik
Iceland

Telephone: 525 4000

Bibliographic information:
Níels Bjarnason, 2012, Software Agent-Based Cost Model Calculations for Distributed Server Environments, Master's thesis, Faculty of Industrial Engineering, Mechanical Engineering and Computer Science, University of Iceland.

Printing: Háskólaprent, Fálkagata 2, 107 Reykjavík
Reykjavik, Iceland, September 2012

Abstract

Facing the problem of how to monitor usage of server resources in a distributed server environment with the goal of creating a usable cost model a distributed agent system was created. The agents were based on a framework modeled on the assignment in hand. These agents are able to inhabit servers in the network and monitor their memory and processor usage and also the server's incoming and outgoing connections. The agents perform these tasks by using behaviours implemented on top of the framework. The data collected by the agents is stored to a centralized repository (database) for further processing. By using the data collected by the agents on all servers a cost model was created. The cost model presents the collected data after it has been processed with the necessary methods. With the cost model report the data is made accessible for the user with customizable features. At the end the created cost model is evaluated and the results compared to the existing cost model.

Útdráttur

Standandi frammi fyrir því vandamál að þurfa að fylgjast með notkun á þjónum í dreifðu umhverfi með það fyrir augum að búa til nothæft kostnaðarmódel var dreift kerfi geranda búið til. Gerendurnir voru búnir til útfrá ákveðnum ramma sem var búinn til með væntanlegt verkefni í huga. Þessir gerendur höfðu þann eiginleika að geta legið á þjónum á netinu og fylgst með minnis og örgjörva notkun sem og tengingum til og frá þjóni. Þessi verkefni leystu gerendurnir með því að nota hegðanir sem hefðu verið úfærðar í rammanum. Öllum söfnuðum gögnum skiluðu gerendurnir í miðlægan grunn (gagnagrunn) þar sem unnið var frekar með gögnin. Með því að nota gögnin sem var safnað á öllum þjónum af gerendunum var búið til kostnaðarmódel. Kostnaðarmódelið birti gögnin eftir að búið hefði verið að vinna þau með nauðsynlegum aðgerðum. Með því að búa til skýrslu sem framvísaði kostnaðarmódelinu voru þessi gögn gerð aðgengileg fyrir notandan með þeim möguleika að aðlaga þau að þeirra þörfum. Að endingu var kostnaðarmódelið metið og niðurstöðurnar bornar við það sem gamla kostnaðarmódelið hefði sýnt fram á.

Dedication

I dedicate this thesis to my family.
Thanks for all your patience and support.

Preface

When it came to finding a thesis project I had to think long and hard about what I wanted to do. I knew I wanted to do a project that had something to do with the environment I was working with at my current employers. After looking at all the possible problems I wanted to solve one in particular caught my attention. This was the problem of charging for server usage inside the IT department. I had been interested in learning more about software agents for a long time and especially how they could be used to monitor information about various things so I saw this as a great opportunity. It offered a great environment to test agent management in a distributed server environment. I also had the desire to learn more about Python and programming in Python since I find it a really interesting topic and a fast growing one so it was a welcome opportunity to get to do a thesis project like the one discussed in this thesis. It opened many new doors for me which I welcome freely.

Table of Contents

List of Figures

List of Tables

Abbreviations

ACC	Agent Communication Channel
ACL	Agent Communication Language
AMS	Agent Management System
ETL	Extract, Transform and Load
FIPA	Foundation for Intelligent Physical Agents
HTML	Hypertext Markup Language
IT	Information Technology
JADE	Java Agent Development framework
KQML	Knowledge Query and Manipulation Language
MSSQL	Microsoft SQL Server
ODBC	Open Database Connectivity
SSIS	Microsoft SQL Integration Services
SSRS	Microsoft SQL Reporting Services
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

Acknowledgements

I would like to thank my advisor, Professor Helmut Neukirchen, for his help and guidance during the creation of this master project and thesis. I would also like to thank the secondary supervisor Snorri Agnarsson and the faculty representative Jóhann Pétur Malmquist for their efforts.

I also want to thank my supervisors at work for giving me the opportunity to work on this thesis within the company and all the co-workers that helped with providing the test servers and the necessary tools to be able to accomplish this project.

I also want to thank Einar Sveinsson for the cooperation on this project and the countless hours spent on it.

Thanks,
Níels Bjarnason

1 Introduction

In today's ever growing IT infrastructure it gets harder and harder to have a good overview over the whole server environment of an organization. This is especially a problem in the banking environment which consists of many individual servers which host many different network, application and database services. It can be hard to track traffic of certain software or a service which is used inside the organization and grasp the scope of the software or service. It is also hard to see where the resources of a certain server are going, who is using them, how and etc.

This thesis will address a part of the problem of getting better overview of the growing server environments in IT infrastructures for an Icelandic bank. In the bank each division is operated as a private company, in terms of expenses and earnings. The bank operates a number of servers which are used by the many divisions. Costs for these servers are split between the many divisions that are using them by the division that owns the servers.

This master thesis investigates this subject together with another master thesis by Einar Sveinsson [1]. Both involve the creation of a Python agent framework, agent behaviours and cost model for server usage calculation. This thesis will add a focus on time scheduling of agent behaviours, data post processing and data presentation and cost model calculations while Einar's thesis will focus on the agent lifecycle, communication between agents and data accuracy. Einar's thesis will also consider design changes to the framework and agent behaviours while this thesis will only use the standard design.

1.1 Problem

The main problem with the approach of charging for server usage is that we have multiple services that serve the role of users for the server network which is operated within the bank. This means that we have to divide the cost of each server between multiple number of users. Today users are charged for their usage of servers not by how they are using it or how much but instead all equally, i.e. total cost of server is divided by number of registered users for the server and each user is charged equally. This does not reflect the right cost model for the servers and some users are charged unfairly while others profit from this.

Another problem is that we do not know how the servers are being used and often not who is using them. This means we do not get the right overview of our systems and furthermore we are not charging the right divisions the right amount because of this lack of information. We are not only charging unfairly but sometimes we are not charging departments that are using a server because registration was not done correctly and therefore we do not know that a certain software is using the server.

Yet another problem that we have is that it can be argued how we measure the usage of a server. In our approach we will be using a number of server parameters to measure the total usage for the server. These parameters are random access memory usage, central

processing unit usage and network traffic. This gives us a usable overview of the server usage which can be used in the solution of the problem.

1.2 Approach

The approach for a solution to the problem is to create and utilize a distributed software agent that can collect data about the systems and transmit them to a centralized repository. The agent will collect samples of data about random access memory usage, central processing unit usage and network traffic and these three information categories will be used as cost items. These server parameters will be sampled and collected with appropriate time schedule. The data will be collected by agents that operate on each server and is stored on one control server where all data will be kept for post processing and later representation. Data will be processed using some calculations and then stored with unique time identification in the corresponding tables within an accessible database. Calculations will then be done on all data for each of the three information categories for each time id and with that we will get a cost model for each server for a given time. By doing this we will be able to use the collected data for the cost model so it shows us the usage of individual software for a given server.

With collecting the data, processing it with calculations and representing it according to the desired cost model we will make the job for those who handle accounting much easier. By querying the cost model reports the data will be presented in a useful way so accounting can be done easily and without any extra work. Usage and costs of a single server divided for all users with the right percentage of usage can be accessed fairly easily through the reports and hopefully by that solving the problem of the wrong division of costs between departments.

The decision of using agents was to be able to configure all agents at once instead of having the monitoring process as a running application. By using agents the communication between systems was also easier and it made it easier to create and drop agents if needed. It was decided that for an environment unknown in size it would be easier to start monitoring services on each server by using agents that could easily be started on each new server that needed to be monitored.

1.3 Outline of thesis

After the introduction chapter the thesis will start providing the foundations used in the thesis: methods, principals and tools. It introduces each element and explains its key fundamental structure and usage.

Following the foundation chapter there is chapter three which is about related work. This chapter focuses on work done by others in the field of agent frameworks and server monitoring and how it is related to this thesis.

Chapter four is about the cost model used for calculating the cost for servers. This chapter explains the fundamentals about the model which is a key element in the thesis. It explains the formulas used for these calculations and how the results are presented and how they can be used and customized.

Next comes chapter five which is a about the agent framework used by created agents. This chapter explains the structure of the framework and its key ingredients and operations. The chapter goes into details how the elements of the framework can be used and extended and what they do.

Then comes chapter six which is about the agent itself and its customized behaviours. This chapter explains the usability of the agent and its possibilities. It has a detailed description of the created operation used by the agent and how they work and what they do. It also shows the relationship of its created behaviours.

After chapter six there is a chapter about the post processing (chapter seven). This chapter focuses on the work done on the sampled data. It explains the database structure used for the collected data and about the data post processing and data representation. It also talks about the tools used to post process the data and represent it.

Chapter eight contains an evaluation of the behaviour scheduling. This chapter goes into details about the work done with assessing the time scheduling of executed behaviours. It examines different runtimes of behaviours and introduces and explains the results gathered with each scenario. This chapter explains the effect of these behaviour schedules and what the effect means.

At last is chapter nine where all the conclusions from the thesis will be gathered and talked about. This chapter grasps all that has been done and sums up the key findings and results of the thesis.

2 Foundations

In this chapter the foundations of the concepts and technologies used in this thesis are presented. The distributed software agent that is created is formed out of the JADE framework but implemented in Python following the FIPA guidelines. Data is stored in an SQL database, processed with Microsoft integration server and presented with Microsoft report server.

2.1 Agents

A Software agent is "a software entity which functions continuously and autonomously in a particular environment, often inhabited by other agents and processes" [2]. To say that agents are autonomous means that to some extent they have control over their behaviour and act without interventions of humans and other systems [3]. Agents should be able to carry out activities in a flexible and intelligent manner and respond to changes in the environment without requiring constant human guidance or intervention. An agent that inhabits an environment with other agents and processes should be able to communicate and cooperate with them [2].

Agents can coexist in a multi-agent system, which is a system composed of multiple interacting intelligent agents in which they coordinate their knowledge and activities and reason about the processes of coordination. Multi-agent systems can also distribute problem solving in which the work of solving a particular problem is divided among a number of nodes that divide and share knowledge about the problem and the developing solution. Multi-agent environments provide an infrastructure specifying communication and interaction protocols. The environment is typically open and has no centralized design [3].

For the agent to be able to fulfill the tasks he is intended to do he must be able to communicate with other agents on the internal network. He must also be able to collect performance data for a server which is his fundamental role.

2.1.1 FIPA standards

The Foundation for Intelligent Physical Agents, or FIPA, was formed in 1996. It was formed with the purpose of producing software standards for heterogeneous and interactive agents and agent-based systems. FIPA works on creating open standards which provide specifications that range from architectures to support agent communication, communication language and content language for expressing messages and interaction protocols which expand the scope from a single message to complete transactions [4].

FIPA specifications represent a collection of standards that are intended to promote collaboration of heterogeneous software agents and the services they represent. These specifications cover wide ranges of issues. FIPA specifications do not attempt to describe how developers should implement their agent-based systems nor how they should attempt to specify the internal architectures of agents. Instead they provide the interfaces through

which agents can communicate [4]. The structure of the FIPA specifications can be seen in Figure 2.1.

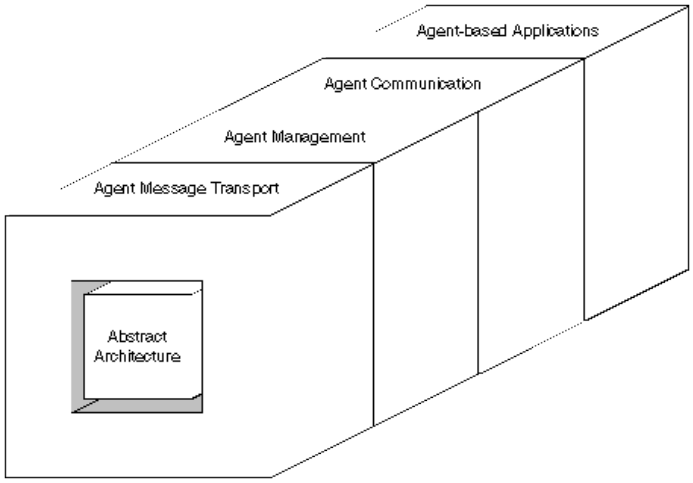


Figure 2.1: Structure of FIPA specifications **[4]**.

The standard specifies the Agent Communication Language (ACL). Agent communication is based on passing messages between each other. Agents communicate by formulating and sending messages to each other. The FIPA ACL specifies a standard message language by providing the encoding, semantics and pragmatics of the message. The syntax of the ACL is very close to the widely used communication language KQML, knowledge query and manipulation language. The standard supports common forms of inter-agent conversations through the specifications of interaction protocols, which are patterns of messages exchanged by two or more agents. Other parts of the FIPA standard specify other aspects. In particular the standard specifies the agent-software integration, agent mobility and security, ontology service and the Human-Agent communication [5]. The FIPA reference model of an agent platform can be seen in Figure 2.2. Agent Platform provides the physical infrastructure where agents can be started. (ACC stands for Agent Communication Channel.)

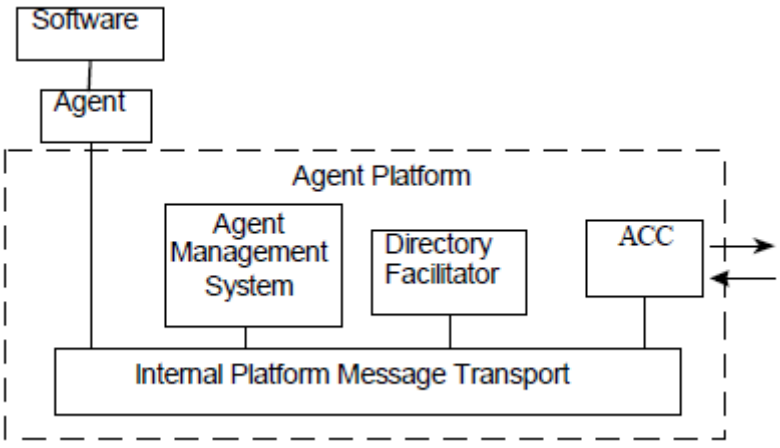


Figure 2.2: FIPA reference model of an Agent Platform **[5]**.

AMS (Agent Management System) is a mandatory component of the agent platform. The AMS controls the access to and use of the agent platform. The AMS maintains a directory of ids for agents registered with the agent platform which contain transport addresses. Each agent must register with the AMS in order to get an id [6].

KQML is a protocol for exchanging information and knowledge. In KQML all information for understanding the content of a message is included in the communication itself. The syntax for KQML is Lisp-like, however the arguments are identified by the keywords preceded [3]. For an example of a KQML message see Figure 2.3.

```
1    :sender :name AMS :address 10.16.46.15:8000
2    :receiver :name agent1 :address 10.16.52.12:8000
3    :content :action ticker :ms 100 :maxTicks 10
```

Figure 2.3: Example of an KQML message

2.2 SQL

SQL is not an acronym for anything but in some cases it is said that it stands for Structured Query Language. SQL is a programming language designed for managing data in database systems. It is a nonprocedural language which means it does not define both the desired results and the mechanism or process by which the results are generated. Instead it defines the desired results but the process by which the results are generated is left to an external agent [7].

```
1    SELECT id, name
2    FROM staff
3    WHERE name like 'Níels%'
```

Figure 2.4: SQL example.

The example in Figure 2.4 shows a simple SQL query that selects the id's and names of all people in table staff that have a name starting with "Níels".

Microsoft SQL Server data solution can be used for data gathering, data processing and data representation. These Microsoft solutions used in this thesis where Microsoft SQL Server 2008R2 and the SQL Server add-ons Microsoft SQL Server 2008R2 Integration Services and Microsoft SQL Server 2008R2 Reporting Services.

2.2.1 Microsoft SQL Server

Microsoft SQL Server 2008R2, also called MSSQL, is a robust and enterprise-reliable database management system (DBMS) that is capable of running anything from a personal database only a few megabytes in size up to a multi-server database system managing terabytes of information. The main component of SQL Server 2008 is the Database Engine. To be able to use the Microsoft SQL Integration Services and the Microsoft SQL Reporting services components a database engine must be installed and configured. The Database Engine is the core application service in the SQL Server package for storing, processing and securing data with SQL Server 2008 [8].

On a single MSSQL server instance you can create multiple databases. In each database you can then create number of database objects used to store and access data. Examples of these objects are:

- Tables: Used for storing the data itself.
- Views: Used to create custom views on top of tables.
- Stored procedures: Used to access data with programmable behaviour.
- Indexes: Used to index data for faster access and lookup on tables.
- Triggers: Used to have action triggers on tables.

2.2.2 Microsoft SQL Integration Services

Microsoft SQL Server 2008R2 Integration Services, also called SSIS, is a "set of utilities, applications, designers, components and services all wrapped up into one powerful software application suite" [9]. SSIS focuses on importing, exporting and transforming data from one or more data sources to one or more data targets. SSIS can be used for a large variety of data transformation purposes and its strength is in direct data access and complex data transformation. Data can be stored in many different formats, contexts, file systems and locations. It can require significant transformation and conversion processing before being represented. The SSIS environment addresses these problems with its tools and abilities [8]. Figure 2.5 shows an example of a data warehouse populated from multiple data sources with SSIS.

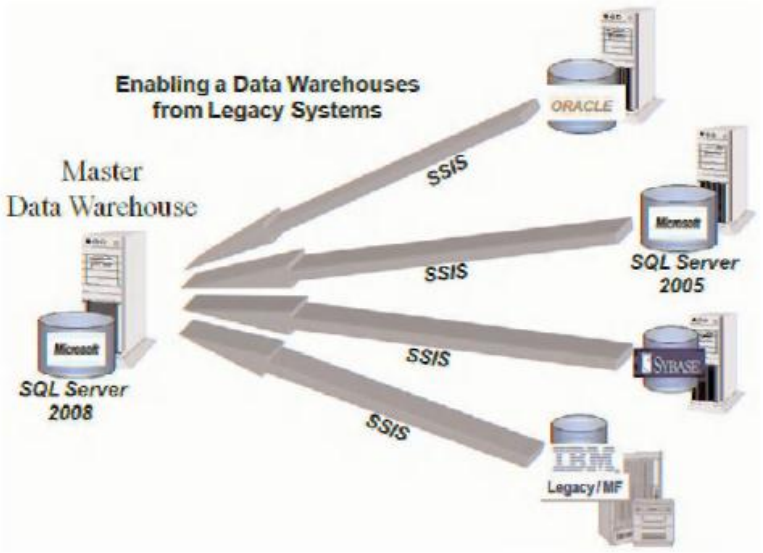


Figure 2.5: Data Warehouse populated from multiple data sources [6].

Integration services offer data import and export wizards, ETL (Extract, Transform and Load) tools, control flow engines and high performance data transformation platforms [9].

- Data import and export wizards makes it easy to move data from source location such as a flat file or database to other destination, flat file or database for example [9].

- ETL tools, or Extract, Transform and Load tools, is the tools mostly used for data warehousing. It is used for extracting data from source transaction systems and transforming, cleaning, duplicating and conforming the data. Finally the data is loaded to the desired location [9].
- Control flow engines are used to move data from location to location and transforming it along the way. This is not restricted to only processing data. The control flow engine can also handle file tasks, table manipulations, rebuilding indexes, performing backups and many other tasks useful for database management [9].
- High performance data transformation platforms can be used to perform complex data transformations on very large datasets. The pipeline concept means that "you can process data from multiple heterogeneous data sources, through multiple parallel sequential transformations, into multiple heterogeneous data destinations". This makes it possible to process data found in different formats and on different media in one common location [9].

2.2.3 Microsoft SQL Reporting Services

Microsoft SQL Server 2008R2 Reporting Services, also called SSRS, is a "server-based, extensible and scalable platform that delivers and presents information based on the data that is collected during operation". The Reporting Services extends traditional paper reports to interactive reports with various forms of delivery for example email and file shares. Report Server is capable of generating reports in different formats for example HTML, XML and Excel [10]. SSRS helps the designer to design reports using a variety of advanced data visualization controls, populate them with data gathered from variety of sources and to deploy secure access schedules for execution of reports [8]. The phases of report development can be seen in Figure 2.6.

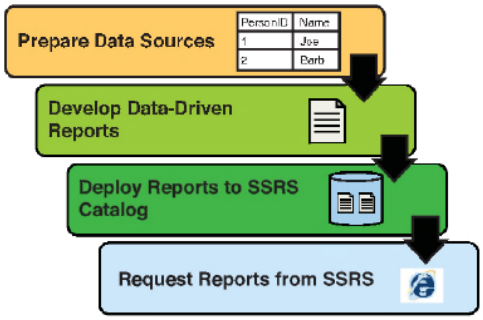


Figure 2.6: Phases of report development **[8]**.

A report is a tool to present data from one or more sources. Reports may display data-bound and non-data bound controls offering static and interactive views on data. It may include all kinds of elements such as header, footer, table of contents, links, images and so forth. To design reports the designer can use various layouts, styles and file formats. It includes sorting, filtering and grouping functionality. Reports can reference embedded or externally stored credentials and data sources for accessing data. Reports also offer input parameters for customization of reports whose values are passed from users or programmatic inputs [8]. Data presentation can be controlled with these parameters to increase customizability for users. These parameters can for example represent a specific

query from a data set, dates or a single value. SSRS can present data with multiple features such as graphs, gauges and tables which helps representing the data in the best possible way. An example of a SSRS report can be seen in Figure 2.7.

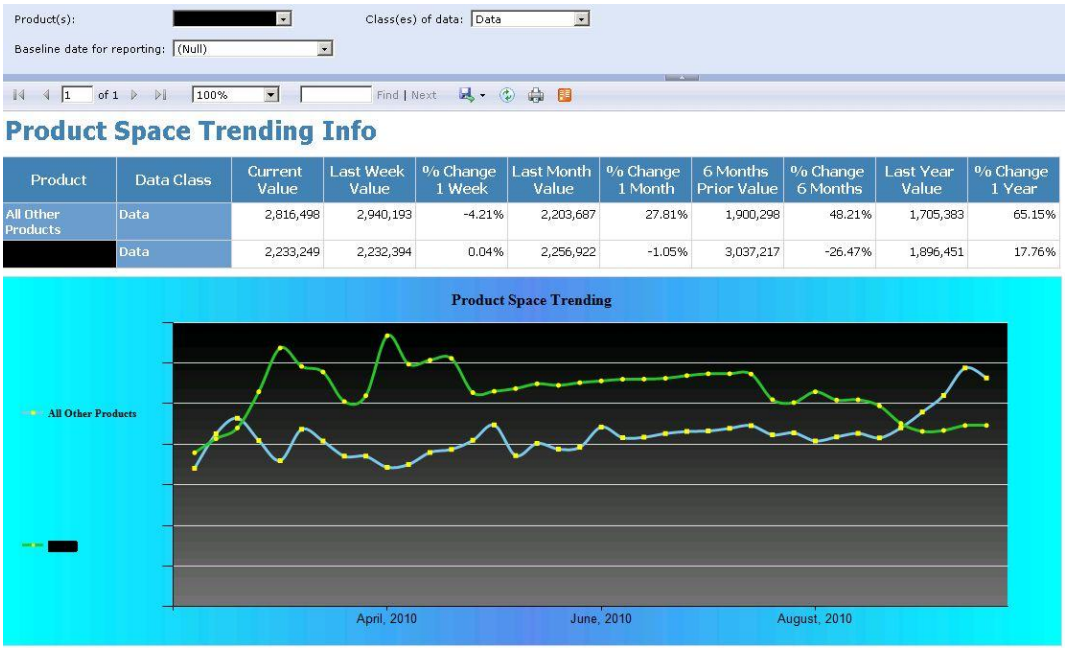


Figure 2.7: Example of a report from SSRS **[11]**.

2.3 Python

Python is an "open source computer programming language optimized for quality, productivity, portability and integration". Python is a popular programming language that reduces the development time and is used in a wide variety of products and roles. Python sports OOP which is a remarkably simple, readable and maintainable syntax. It also sports integration with C components and a vast collection of precoded interfaces and utilities. Its tool makes Python a flexible and agile language. It is ideal for both quick tactical tasks as well as complex application development efforts. Python is often called a scripting language because it makes it easy to utilize and direct other software components. Python makes software development more rapid and enjoyable [12].

2.3.1 PSUTIL

"PSUTIL is a module providing an interface for retrieving information on all running processes and system utilization in a portable way using Python" [13]. PSUTIL is used in the agent for collecting process, memory and processor information about servers. See Figure 2.8 for a simple example of PSUTIL usage.

```
1    import psutil
2    process = psutil.Process(7055)
3    name = process.name
4    percentage = process.get_memory_percent()
```

Figure 2.8: PSUTIL code.

The code in Figure 2.8 gets a reference of the process with id equal to 7055. It then gets the name of the process and stores in variable name and the percentage of memory usage for that process and stores that information in the variable percentage.

2.3.2 PYODBC

ODBC means open database connectivity and is a Microsoft strategic interface for accessing data in heterogeneous environment of relational and non-relational database management systems [14].

PYODBC is a Python module that allows you to use ODBC to connect to almost any database from Windows, Linux, OS/X and more. It implements the "Python database API Specification v2.0", but additional features have been added to simplify database programming even more [15]. See Figure 2.9 for a simple example of usage.

```
1    import pyodbc
2    cnxn = pyodbc.connect('database_server')
3    cursor = cnxn.cursor()
4    num = cursor.execute("SELECT count(*) FROM database.dbo.table")
5    cnxn.commit()
```

Figure 2.9: PYODBC code.

The code in Figure 2.9 connects to the server "database_server" and selects the total number of items from table "database.dbo.table" with an SQL statement and stores it in the variable num.

2.3.3 Twisted

"Twisted is an open source Python framework for building network applications. It gives developers a complete set of tools for communicating across networks and the internet. Twisted includes both high- and low-level tools. Twisted provides a tested well-designed API that makes it possible to rapidly develop powerful network software" [16].

Twisted is written in Python and since Python is a cross-platform language the same Twisted applications can be run on Linux, Windows, Unix and Max OS X. Since Twisted has an event-based, asynchronous framework it is possible to write applications that stay responsive while processing events from multiple network connections without using threads. Twisted includes wide range of functionalities ready for use. These functionalities are for example Mail, web, news, chat, DNS, SSH, Telnet, RPC and database access. Twisted also provides high-level classes for a quick start. With Twisted advanced functionality and customizations of protocols are possible and the same goes for implementations of new protocols. Twisted is free and is released under a liberal license with full source code [16].

Twisted is divided into several packages, each providing different services. Typically, higher-level packages build on lower-level packages, allowing developers to depend only on those packages necessary to their application. One of the main packages of Twisted is the package twisted.internet. The twisted.internet package provides networking event-loop which was chosen over threads as it tends to be more scalable, integrates well with GUI applications and is supported by visually all operating systems. In an event-loop

a single thread responds to network events, deals with the appropriate processing and then continues processing [17].

Implementation of protocols is separated from transport implementation so protocols can run more or less transparently on top of transports of the same kind. For client connection Twisted provides high-level functionality. These functionalities are for example reconnection on failure (or disconnect), connection timeouts and the ability to cancel connections. The event-loop also supports scheduling, thread integration, hooks for non-blocking DNS lookups and other commonly required tasks. It also provides an event driven loop called asynchronous reactor loop [17].

Deferred

The deferred object is a part of the twisted.internet package. Twisted uses the deferred object to manage the callback sequence. The client application attaches series of functions to the deferred to be called in order when the results of the asynchronous request are available. This series of functions is known as series of callbacks or a callback chain. Along with the series of callbacks a series of functions to be called if there is an error in the asynchronous request are attached to the deferred. This series of error handling functions is knows as a series of errbacks or an errback chain. The asynchronous library code calls the first callback when the result is available or the first errback when an error occurs. The deferred object then hands the results of each callback or errback function to the next function in the chain [18]. An example of a callback function and an errback function can be seen in Figure 2.10.

```
1    from twisted.internet import defer
2
3    class Tester:
4         def doOperation(self, number):
5              if x < 0:
6                   self.der.callback(number)
7              else:
8                   self.der.errback(number)
9
10   def finalize(number):
11        print number
12
13   def report_error(number):
14        print 'error: %s' % number
15
16   op = Tester()
17   der = op.doOperation(x)
18   op.addCallback(finalize)
19   op.addErrback(report_error)
```

Figure 2.10: An example of a deferred callback and errback.

In the example in Figure 2.10 we get a single number into the doOperation() function. If this number is larger than zero the number is printed out with the callback function finalize(). If the number is smaller or equal to zero we get the number printed out with an error message with the errback function report_error().

2.4 Agent internals

An agent can have lot of internals it uses for solving the tasks and problems it is given. The key internals used in the programmed agent for this thesis are agent communication, agent data collection and agent lifecycle. These internals describe how the agents communicate, collect data and how there lifespan is controlled. All these internals are described in the context of this thesis and the agents programmed.

2.4.1 Agent communication

"An agent is an active object with the ability to perceive, reason and act". An agent has the ability to communicate and can get and store knowledge for communication in its knowledge store. Communication protocols enable agents to exchange and understand messages. Interaction protocols enable agents to have conversations [3].

2.4.2 Agent data collection

An agent is able to collect data with programmed behaviours. Behaviours are the control mechanism of what the agent can do. With the behaviour the agent fulfills many tasks and one of them is data collection. Behaviour and the process of data collection will be described in more detail later in this thesis.

2.4.3 Agent lifecycle

According to FIPA an agent has only one agent platform lifecycle state at any time and within only one agent platform. The guidelines for FIPA Agent Management system describe the states that an agent can be in at any given time. These states are waiting, active, initiated, suspended and transit state [6]. See Figure 2.11 for the state diagram.

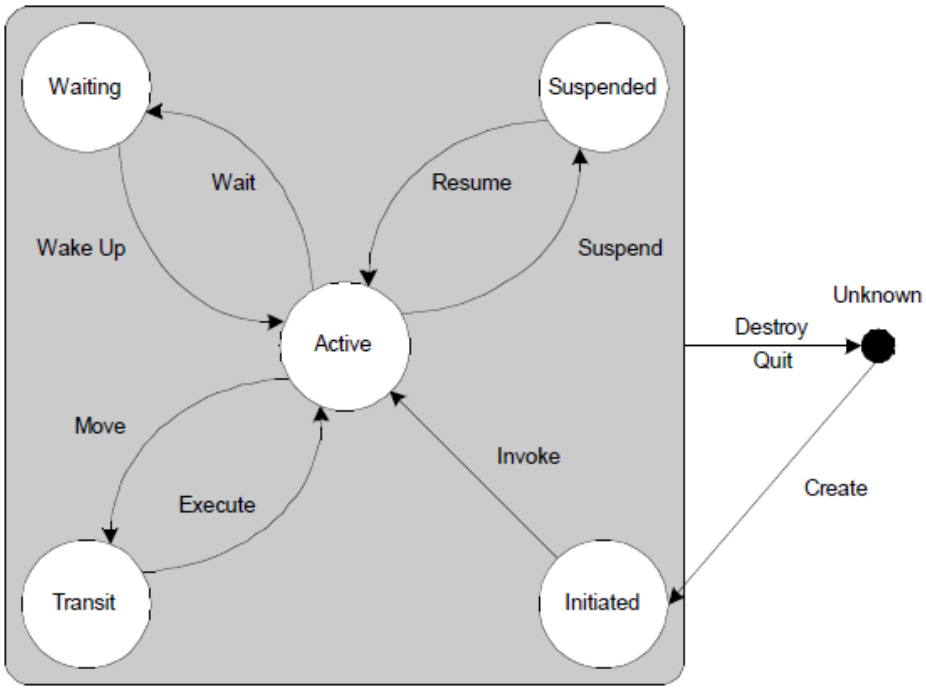


Figure 2.11: Agent lifecycle **[6]**.

3 Related work

Few projects and software we came across were using similar ideas with either using Python framework for multi-agent environment or monitoring processor and memory usage. In this chapter there will be a short introduction of them.

3.1 Monitoring

3.1.1 SCOM

SCOM (System Center Operation Manager) is a product from Microsoft for monitoring Windows operating systems and applications. Operation Manager focuses on the health of applications and components opposed to looking at the status of individual servers [19]. SCOM provides flexible infrastructure monitoring, helps ensure the performance and availability of vital applications and offers monitoring for datacenters and clouds. [20].

Operation Manager helps IT departments to identify when there is a problem, where the problem is and what is causing the problem. Using SCOM makes it easier to monitor multiple computers, devices, services and applications. It provides console to check the health, performance and availability for all monitored objects in the environment. SCOM can be used to monitor the objects and send alerts when problems occur [20]. See Figure 3.1 for an example of the operation manager console.

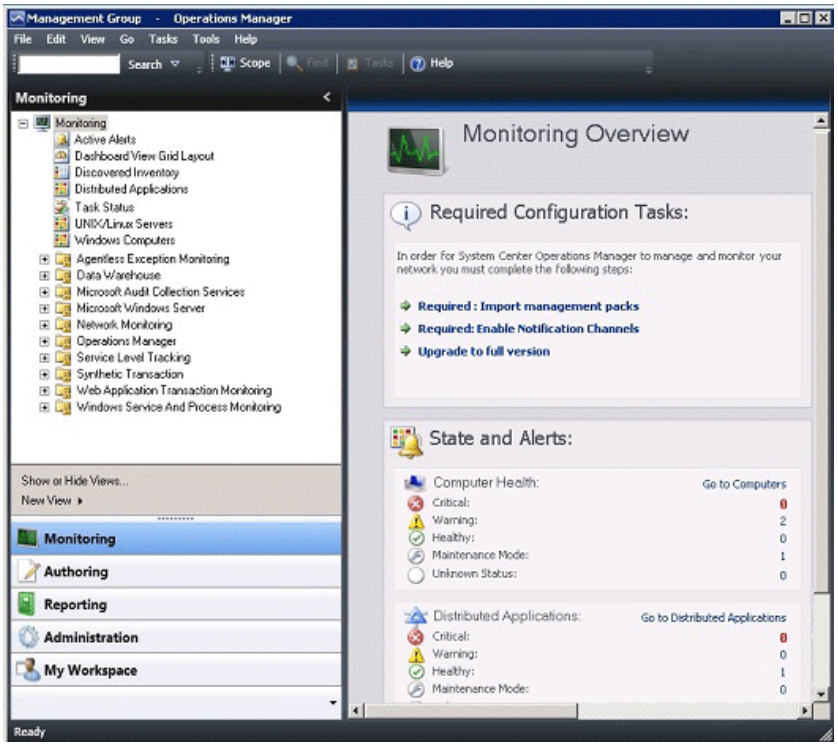


Figure 3.1: The Operation Manager console **[20]**.

SCOM was not used for this project since it is a massive Microsoft solution which is really large and expensive and does not fit the purpose we were trying to get with the creation of our solution.

3.2 Agent frameworks

3.2.1 SPADE2

SPADE2 is the second edition of SPADE (Smart Python multi-Agent Development Environment) which is a multi-agent and organization platform based on XMPP (Extensible Messaging and Presence Protocol), originally named Jabber, technology and written in Python. This technology offers many features by itself and capabilities that help with the construction of multi agent systems, the concepts of users (agents) and servers (platforms) and an extensible communication protocol based on XML [21]. The structure of SPADE2 can be seen in Figure 3.2.

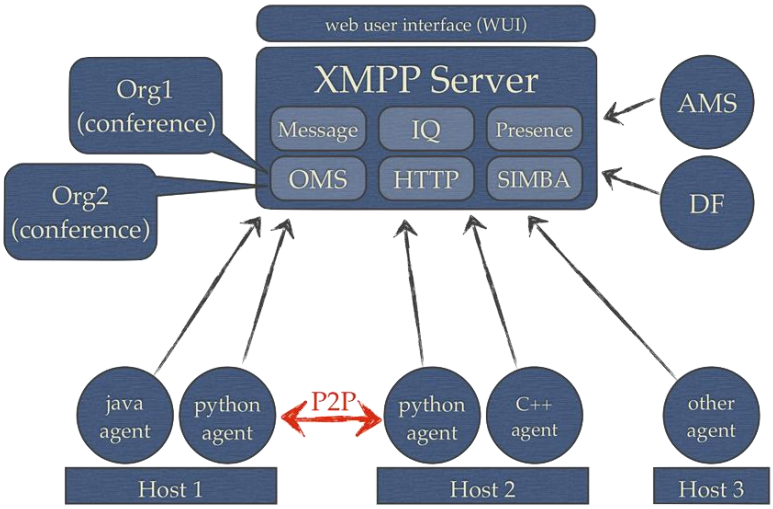


Figure 3.2: Structure of SPADE2 **[21]**.

The main features of SPADE2 are that it is developed using Python, it covers the FIPA standards, implements four media transfer protocols, supports FIPA-SL and resource description framework, SPADE agents reach their goals by running deliberative and reactive tasks, has a web interface to manage platforms, allows its execution in several platforms and operation systems, presence notifications allows the system to determine the current state of the agent in real-time, multi-user conference allows agents to create organizations and groups of agents [21].

SPADE2 offers a SPADE agent library for Python for building agents. It is a collection of classes, functions and tools for creating SPADE agents that can work with the SPADE2 agent platform [21].

SPADE2 was not used for this thesis since the latest release was not yet out when the project started and it was decided that we would learn more from the project by programming a new Python framework.

3.2.2 JADE framework

JADE (Java Agent Development Framework) is a "Java software framework to develop agent applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems" [5]. The goal of JADE is to simplify development while ensuring standard compliance through a comprehensive set of system services and agents. JADE can then be considered an agent middle-ware that implements an Agent Platform and a development framework. It deals with all those aspects that are not specifically of the agent internals and that are independent of the applications. These aspects are for example message transport, encoding and parsing and agent life-cycle [5].

A JADE application is made out of a set of Agents with unique names. Agents execute tasks and communicate by sending messages to each other. Agents live on top of a Platform. From the Platform the Agents get their basic services such as message delivery. Each Platform is made out of one or more Containers that contain zero or more agents. Containers can be executed on one or more servers and by that creating distributed server environment [22]. Figure 3.3 shows the main JADE architecture.

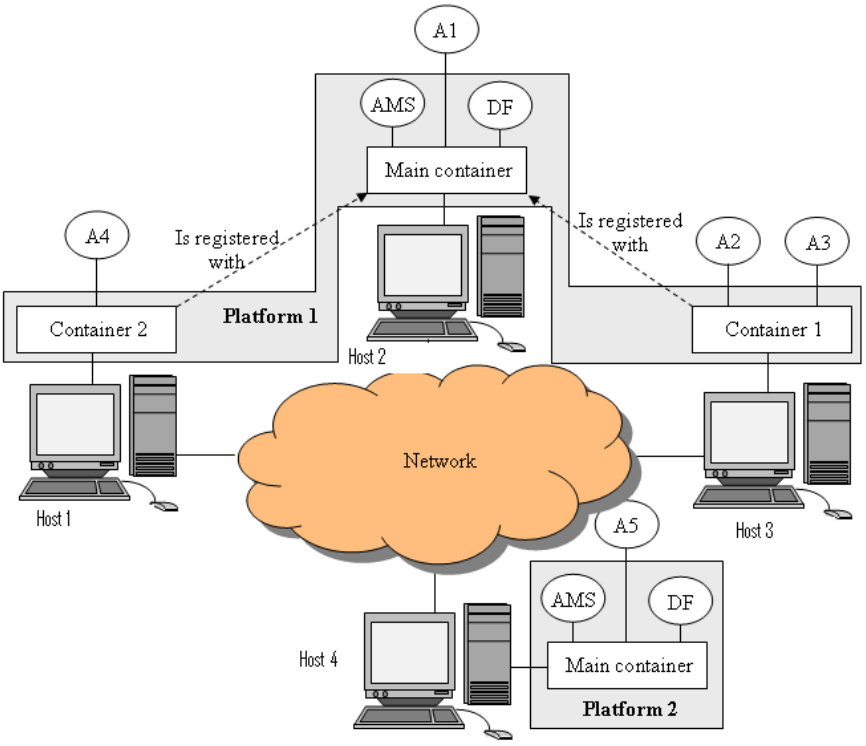


Figure 3.3: The JADE Architecture **[22]**.

It was decided not to use the JADE framework since it is a Java framework and we had decided to use Python because of the tools available for server monitoring. The framework created was though based on the JADE framework architecture.

4 Cost Model

The main reason for creating the agents was to get the data to be used in a cost model for server cost division so charging for servers could be done more accurately. The gathering of data from the servers was done with the created agents and their framework which was structured so that calculations on the gathered data would reflect the right cost model. See chapter 5 for more information about the agent framework and chapter 6 for information about the agents themself.

By using the cost model it is possible to calculate the division of costs for the servers and represent these calculations and their results in a report. The user is then able to access this information in the cost model report.

4.1 Purpose of the Cost Model

By getting the proportion between systems connected to the servers a more accurate view on server usage can be acquired. This is done by creating a cost model that is used for processing the collected data and by that presenting a more accurate division of usage between connected systems. With the cost model more accurate percentage of usage for each connected system to server can be presented in a report. The report is customizable so that the user can change the definition of the cost model, such as time interval, systems that should pay and cost of server. By inputting these simple values the right customized cost model for each server can be presented in a report for the user. This was done to simplify the work for those who handled server accounting. See section 7.3 for more information about the reports and for examples of the reports.

4.2 Basics of the Cost Model

To create the cost model the processed data is used by summing the processor usage and memory usage of all monitored processes for a given time period and storing it with a created timestamp. The calculated proportion of the total usage is then divided by the number of network connections for the same time period and by that calculating the right ratio of usage for a connected server at a given time.

All data is stored with two timestamps to simplify post processing. One timestamp indicates the time of when the data was written to the database and one indicates when it was sampled on the server. By using these timestamps the data can be divided into blocks of collected data by the type of data, for example processor usage and memory usage. Each block consists of one write timestamp which was the time of execution of data processing package. This write timestamp has the average of usage for all sample timestamps for the period between the last write and the current write. So each line which is used for the calculation is the average usage or number of connections for every timestamp between the time that the processing package was executed and the last time the package is executed. See chapter 7 for more information about data post processing. By doing this we are able to make sure that the right numbers were being used for the calculation of each time period.

The formula for the cost model is derived by collecting the elements talked about in the last paragraph. By taking all these boundaries and processes into account we are able to derive a formula which was used for the representation of server usage per process. The formula can be seen in Figure 4.1.

$$ProcessUsage = \frac{\frac{ProcessMemoryUsage}{TotalServerMemoryUsage} + \frac{ProcessCPUUsage}{TotalServerCPUUsage}}{2}$$

Figure 4.1: Formula for server usage by process.

So by using the logic in this formula the calculations of the collected data could be represented in a useful manner. The logic was restructured so it could be used to get the total usage for each connected server or service. By monitoring network traffic the cost of a server is divided between systems. That was done by dividing the process usage between the systems connected to the process. So if a connected system had three out of five connections to a process it would pay 60% of the process costs on the server. It was decided to estimate that memory and processor usage would each represent 50% of server usage. It could be argued that for example that processor is more expensive then memory and should represent more than 50% of the cost but it was decided to keep it equal. The formula above implemented into code can be seen in Figure 4.2. The code uses four external parameters:

- @server: the server to apply the cost model to.
- @date: date to inspect.
- @processes: processes that should pay for server.
- @cost: estimated cost for server.

```
1   ------------------------------------------------------------------
2   /* Get connections */
3
4   SELECT @server AS server_name, [host_process] AS process_name,
5    [count], [connecting_server] AS connectionIP,
6    [connecting_process], [timestamp]
7   INTO [HIverkefni].[temp].[TotalConnections]
8   FROM [HIverkefni].[dbo].[ConnectionsLink]
9   WHERE [timestamp] = @date
10  AND [host_server] = (SELECT [server_ip]
11  FROM [HIverkefni].[dbo].[Servers]
12  WHERE [server_name] = @server)
13  AND [host_process] IN(@processes)
14  ORDER BY [host_process]
15
16  ------------------------------------------------------------------
17  /* Get memory and cpu usage */
18
19  SELECT [server_name], [process_name],
20  CAST([cpu_usage] AS FLOAT) AS cpu_usage,
21  CAST('' AS FLOAT) AS cpu_prop,
22  CAST('' AS FLOAT) AS cpu_total, [timestamp]
23  INTO [HIverkefni].[temp].[tempCPU]
24  FROM [HIverkefni].[dbo].[ProcessCPU]
25  WHERE [process_name] IN (@processes)
26  AND [timestamp] = @date
27  AND [server_name] = @server
```

```
28
29   SELECT [server_name], [process_name],
30   CAST([memory_usage] AS FLOAT) AS memory_usage,
31   CAST('' as FLOAT) AS memory_prop,
32   CAST('' as FLOAT) AS memory_total, [timestamp]
33   INTO [hiverkefni].[temp].[tempMemory]
34   FROM [hiverkefni].[dbo].[ProcessMemory]
35   WHERE [process_name] IN (@processes)
36   AND [timestamp] = @date
37   AND [server_name] = @server
38
39   UPDATE [HIverkefni].[temp].[tempCPU]
40   SET [cpu_total] = (SELECT SUM(CAST([cpu_usage] AS FLOAT))
41   FROM [HIverkefni].[temp].[tempCPU])
42
43   IF((SELECT SUM(cpu_total)
44   FROM [HIverkefni].[temp].[tempCPU]) != 0)
45   BEGIN
46        UPDATE [HIverkefni].[temp].[tempCPU]
47        SET [cpu_prop] = [cpu_usage]/ [cpu_total]
48   END
49
50   UPDATE [HIverkefni].[temp].[tempMemory]
51   SET [memory_total] = (SELECT SUM(CAST([memory_usage] AS FLOAT))
52   FROM [HIverkefni].[temp].[tempMemory])
53
54   IF((SELECT SUM([memory_total])
55   FROM [HIverkefni].[temp].[tempMemory]) != 0)
56   BEGIN
57        UPDATE [HIverkefni].[temp].[tempMemory]
58        SET [memory_prop] = [memory_usage]/[memory_total]
59   END
60
61   CREATE TABLE [HIverkefni].[temp].[tempTotal](
62        [server_name] VARCHAR(100),
63        [process_name] VARCHAR(100),
64        [cpu_usage] FLOAT,
65        [memory_usage] FLOAT,
66        [res_usage] FLOAT,
67        [timestamp] DATETIME)
68
69   INSERT INTO [HIverkefni].[temp].[tempTotal](
70        [server_name],
71        [process_name],
72        [timestamp])
73   SELECT [server_name], [process_name], [timestamp]
74   FROM [HIverkefni].[temp].[tempCPU]
75
76   INSERT INTO [HIverkefni].[temp].[tempTotal](
77        [server_name],
78        [process_name],
79        [timestamp])
80   SELECT [server_name], [process_name], [timestamp]
81   FROM [HIverkefni].[temp].[tempMemory]
82
83   UPDATE [HIverkefni].[temp].[tempTotal]
84   SET [memory_usage] = a.[memory_prop]
85   FROM [HIverkefni].[temp].[tempTotal] b
86   JOIN [HIverkefni].[temp].[tempMemory] a
87   ON a.[process_name] = b.[process_name]
```

```
88
89   DROP TABLE [HIverkefni].[temp].[tempMemory]
90
91   UPDATE [HIverkefni].[temp].[tempTotal]
92   SET [cpu_usage] = a. [cpu_prop]
93   FROM [HIverkefni].[temp].[tempTotal] b
94   JOIN [HIverkefni].[temp].[tempCPU] a
95   ON a.[process_name] = b.[process_name]
96
97   DROP TABLE [HIverkefni].[temp].[tempCPU]
98
99   IF(((SELECT SUM([memory_usage])
100  from [HIverkefni].[temp].[tempTotal]) != 0)
101  AND ((SELECT SUM([cpu_usage])
102  FROM [HIverkefni].[temp].[tempTotal]) != 0))
103  BEGIN
104       UPDATE [HIverkefni].[temp].[tempTotal]
105       SET [res_usage] = ([cpu_usage] + [memory_usage])/2
106  END
107  ELSE
108  BEGIN
109       UPDATE [HIverkefni].[temp].[tempTotal]
110       SET [res_usage] = ([cpu_usage] + [memory_usage])/1
111  END
112
113  SELECT DISTINCT [server_name], [process_name],
114   [res_usage], [timestamp]
115  INTO [HIverkefni].[temp].[tempTotalRes]
116  FROM [HIverkefni].[temp].[tempTotal]
117
118  DROP TABLE [HIverkefni].[temp].[tempTotal]
119
120  SELECT * INTO [HIverkefni].[temp].[TotalResources]
121  FROM [HIverkefni].[temp].[tempTotalRes]
122
123  DROP TABLE [HIverkefni].[temp].[tempTotalRes]
124
125  ----------------------------------------------------------------
126  /* Calulate cost model */
127
128  SELECT DISTINCT [process_name], SUM([count]) AS charge,
129  CAST('' AS FLOAT) AS res_usage
130  INTO [HIverkefni].[temp].[tempCharge]
131  FROM [HIverkefni].[temp].[TotalConnections]
132  GROUP BY [process_name]
133
134  INSERT INTO [HIverkefni].[temp].[tempCharge]
135  SELECT DISTINCT [process_name], '1' AS charge,
136  CAST('' AS FLOAT) AS res_usage
137  FROM [HIverkefni].[temp].[TotalResources]
138  WHERE [process_name] NOT IN (SELECT [process_name]
139  FROM [HIverkefni].[temp].[tempCharge])
140  group by [process_name]
141
142  UPDATE [HIverkefni].[temp].[tempCharge]
143  SET [res_usage] = b.[res_usage]
144  FROM [HIverkefni].[temp].[TotalResources] b
145  JOIN [HIverkefni].[temp].[tempCharge] a
146  ON a.[process_name] = b.[process_name]
147
```

```
148  SELECT [server_name],[process_name],[count],
149  CAST('' AS FLOAT) AS charge,CAST('' AS FLOAT) AS res_usage,
150  CAST('' AS FLOAT) AS amount, [connectionIP],
151   [connecting_process], [timestamp]
152  INTO [HIverkefni].[temp].[tempRukkun]
153  FROM [HIverkefni].[temp].[TotalConnections]
154
155  DROP TABLE [HIverkefni].[temp].[TotalConnections]
156
157  INSERT INTO [HIverkefni].[temp].[tempRukkun]
158  SELECT [server_name], [process_name], '1',
159  CAST('' AS FLOAT) AS charge, CAST('' AS FLOAT) AS res_usage,
160  CAST('' AS FLOAT) AS amount,
161   (SELECT [server_ip] FROM [HIverkefni].[dbo].[servers]
162  WHERE [server_name] = @server),
163   [process_name], [timestamp]
164  FROM [HIverkefni].[temp].[TotalResources]
165  WHERE [process_name] NOT IN(SELECT [process_name]
166  FROM [HIverkefni].[temp].[tempRukkun])
167
168  drop table [hiverkefni].[temp].[TotalResources]
169
170  UPDATE [HIverkefni].[temp].[tempRukkun]
171  SET [charge] = b.[charge], [res_usage] = b.[res_usage]
172  FROM [HIverkefni].[temp].[tempCharge] b
173  JOIN [HIverkefni].[temp].[tempRukkun] a
174  ON a.[process_name] = b.[process_name]
175
176  DROP TABLE [HIverkefni].[temp].[tempCharge]
177
178  UPDATE [HIverkefni].[temp].[tempRukkun]
179  SET [amount] = ([res_usage] * [count]) / [charge]
180
181  SELECT [server_name], [connectionIP],
182   [connecting_process],
183  SUM([amount]) AS [total_amount], [timestamp]
184  FROM [HIverkefni]. [temp]. [tempRukkun]
185  GROUP BY [connectionIP], [server_name],
186   [connecting_process], [timestamp]
187  ORDER BY [connectionIP]
188
189  DROP TABLE [HIverkefni]. [temp]. [tempRukkun]
```

Figure 4.2: SQL report code for cost model.

The first part of the code in Figure 4.2 (“Get connections”, lines 1-15) is the code that counts and processes number of connections to each process for the selected server. The second part (“Get memory and cpu usage”, lines 16-124) is the part that processes and returns the average of memory and processor usage for each process monitored on the server. The final part of the code (“Calculate cost model”, lines 125-189) takes the results from the previous two parts and joins them together. By doing that we get the amount of which each connected system should pay for the use of the server.

For more information about handling of data, post processing of data and data representation see chapter 7.

5 Framework

The structure of the created agent framework can be divided into three main parts. These parts are container, behaviour and agents and will they be described in the following sections. By describing these main parts the implementation of the framework and how the framework operates will be explained.

The framework created for this thesis was inspired by the Java framework Jade which was described in section 3.2.2. This was done to simplify the architecture work and the creation of the framework.

5.1 Container

The container is the primary part of the framework. It hosts the agents and supplies them with the tools they need. These tools are supporting modules the agent needs to communicate and schedule, so each container gets two modules for each operation. These modules are time dispatcher and message manager; they will be discussed in detail in sections 5.1.2 and 5.1.3.

Multi-agent systems can consist of one or more agents located in one container on a single server or it can consist of multiple agents on multiple distributed servers in multiple containers. See Figure 5.1 for an example of an agent framework on multiple servers.

The container name concept was adapted from the Jade framework but the implementation built in Python and used in this thesis is different from what is implemented in Jade.

5.1.1 Types

The containers come in two types: agent container and domain container. The difference lies in the AMS (Agent Management System) as described in the subsections below.

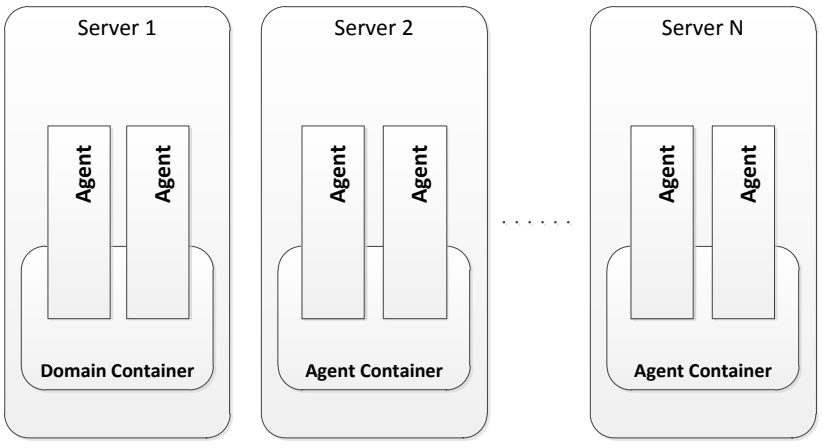


Figure 5.1: Agent framework on N servers.

Agent Container

The Agent container is the standard container. An example of an agent container can be seen in Figure 5.2. An agent container contains the tools message manager (see section 5.1.2) and time dispatcher (see section 5.1.3). When an agent container is created it initializes these tools so agents hosted in the container can use them. The container starts a new thread for each hosted agent but before agents can be hosted the support tools must be initialized. When a new agent is hosted the container registers it both within the hosting container and to the AMS which is located in the domain container for the agent network. The container creates a message with the necessary information for registration and sends it to the AMS. The necessary information is the agent location and agent name. The reason the container registers the agent within itself along with the AMS is to keep track of all agents it is hosting. There is no limit of how many agents can be created within a container as long as the container can start a new thread for the agent.

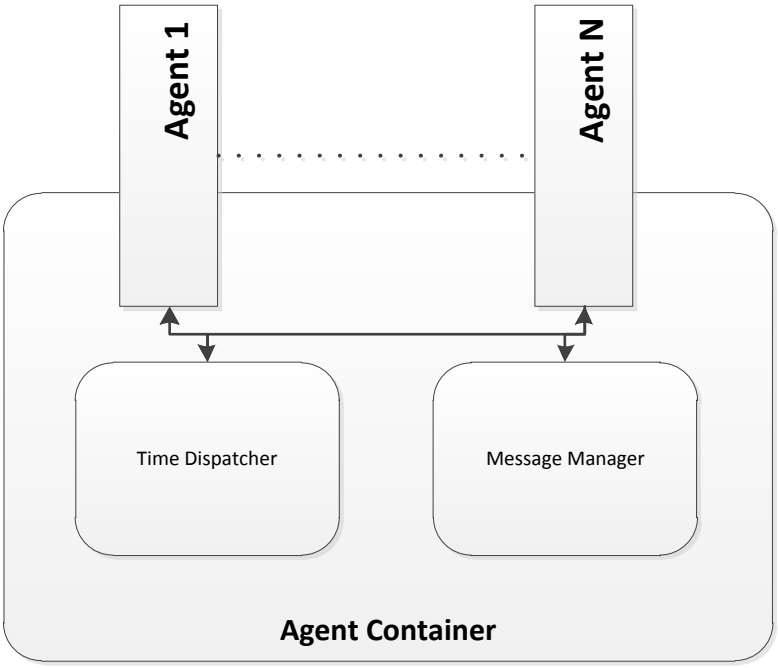


Figure 5.2: Agent container.

Domain Container

Within every multi-agent system which consists of one or more agents there must always be at least one domain container. The domain container holds the AMS which is crucial for the registration and tracking of agents. The domain container also includes the supporting tools message manager and time dispatcher like the agent container. An example of a domain container can be seen in Figure 5.3. When the domain container is started one of the first tasks is to initialize the AMS. Like for the agent container the tools must be initialized before the container can start hosting agents. So the only difference between the domain container and the agent container is the ability to initialize the AMS tool. In all other parts the domain container is the same as the agent container.

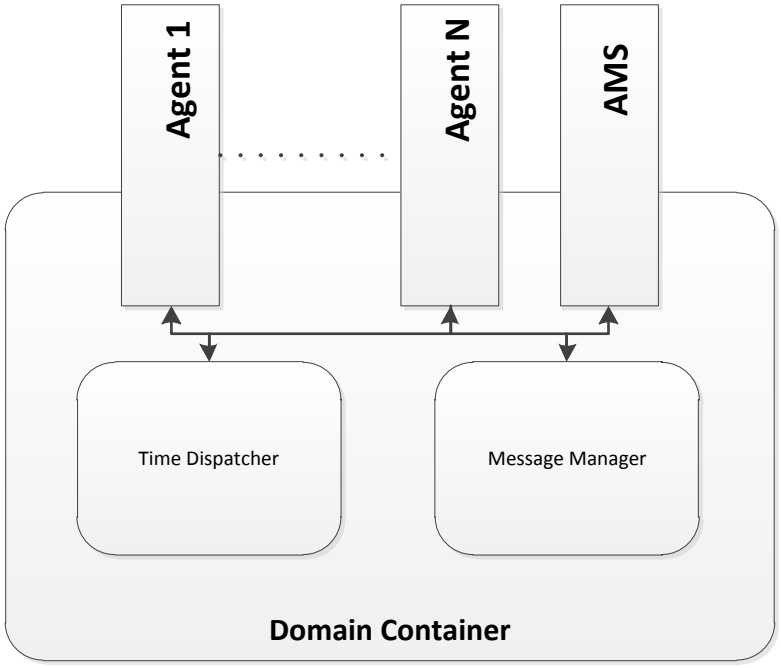


Figure 5.3: Domain container.

5.1.2 Message manager

When a container is created it comes with the tool message manager which is initialized as a separate thread with the creation of the container. The purpose of the message manager tool is to provide a communication module for all the agents within the container so they can communicate with other agents within the container or agents in other containers located in the multi-agent system. Communication for the agents created is done with the Twisted framework. It uses the asynchronous reactor loop included in Twisted to route messages to and from the message module.

Listener and sender

When the message manager is initialized it creates a listener and a sender. Since there was only a prototype of the agents created as described in this thesis the only listener available in the framework is a TCP Listener. The TCP Listener opens a specified TCP port which is used to handle all incoming traffic to the container. The same thing goes for the sender. The sender handles all outgoing traffic from the container. Since Twisted was used by the framework it would be fairly simple to implement other protocols, for example UDP.

Inbox and outbox

When a message is sent to a container the message is handled by the TCPListenerProtocol which encodes the message received and inserts it to the container inbox. From the container inbox the message is routed to the corresponding agent local inbox where the agent picks up the message and processes it. The message is routed with deferred which means that if a message is received the routing function is called as a callback in the outbox module. See section 2.3.3 for more information about deferred.

When a message is sent from the container it receives a message to the container outbox where the message is decoded. The message is then handled by the TCPSenderProtocol which sends it to the agent in the multi-agent system.

An example of an encoded message can be seen in Figure 5.4. When the message is decoded by the TCPListenerProtocol the corresponding information is stored in an object so information about sender, receiver and content can be easily read.

Messaging

The agent has the ability to receive requests for executing behaviours and acting on them. The agent communicates with a modified form of the knowledge query and manipulation language (KQML), see section 2.1.1 for more information about KQML. The changes are that ";;" is used to identify start of line and ";" is used to identify information instead of ":". This was done to be able to use ":" to identify the port for IP addresses. See Figure 5.4 for an example of the modified message.

```
1    ;;sender ;name AMS ;address 10.16.46.15:8000
2    ;;receiver ;name agent1 ;address 10.16.52.12:8000
3    ;;content ;action ticker ;ms 100 ;maxTicks 10
```

Figure 5.4: KQML like message used for agent communication.

The message seen in Figure 5.4 tells the agent the name and address of the sender, the name and address of himself and the action he should take. So the sender is AMS with IP 10.16.46.15:8000, the receiver is agent1 with IP 10.16.52.12:8000 and the action is that he should perform the action ticker with the parameters 100 and 10.

5.1.3 Time dispatcher

When a container is created it comes with the tool time dispatcher which is initialized as a separate thread with the creation of the container. The time dispatcher services all agents within the container. The purpose of the time dispatcher is to make available for the agent the option of executing an action after a specific time with a predefined interval. An example of this is when a behaviour should be executed every five minutes. The action then gets the interval five minutes which means that the behaviour is moved to a blocked queue after execution where it waits for five minutes and is then added to the ready queue for execution. This course of events is repeated every five minutes. To be able to set up this kind of action scheduling the agent creates a timer object which specifies the fundamental values so this scheduling can be done. The timer object includes the end time which indicates when the action should be executed and the action that should be done which in this case is to remove the behaviour from blocked queue and add it to the ready queue. The action needed for the timer object is specified by the name of the function and the attributes that need to be executed with the specified function. For an example of a timer created using Python see Figure 5.5.

```
1    timer = Timer(self._agentScheduler)
2    timer.setEndTime(timeout)
3    timer.addAction('restart', behaviour = _currentBehaviour)
4
5    self._timeDispatcher.addTimer(timer)
```

Figure 5.5: Example of a created timer.

The example in Figure 5.5 shows a timer created with an instance of the scheduler. The specifying of the scheduler is done so the time dispatcher will know the location of the function to be executed when the time of execution comes. The timeout variable in line two indicates the desired timeout, this variable is in milliseconds and is specified in the initialization of the behaviour. In line three the action is added which is specified with the function "restart" and with the attribute behaviour which contains a reference to the behaviour that should be restarted. In the last line or line five the timer is added to the time dispatcher list. There the timer will lay until the time comes for the timer to be processed.

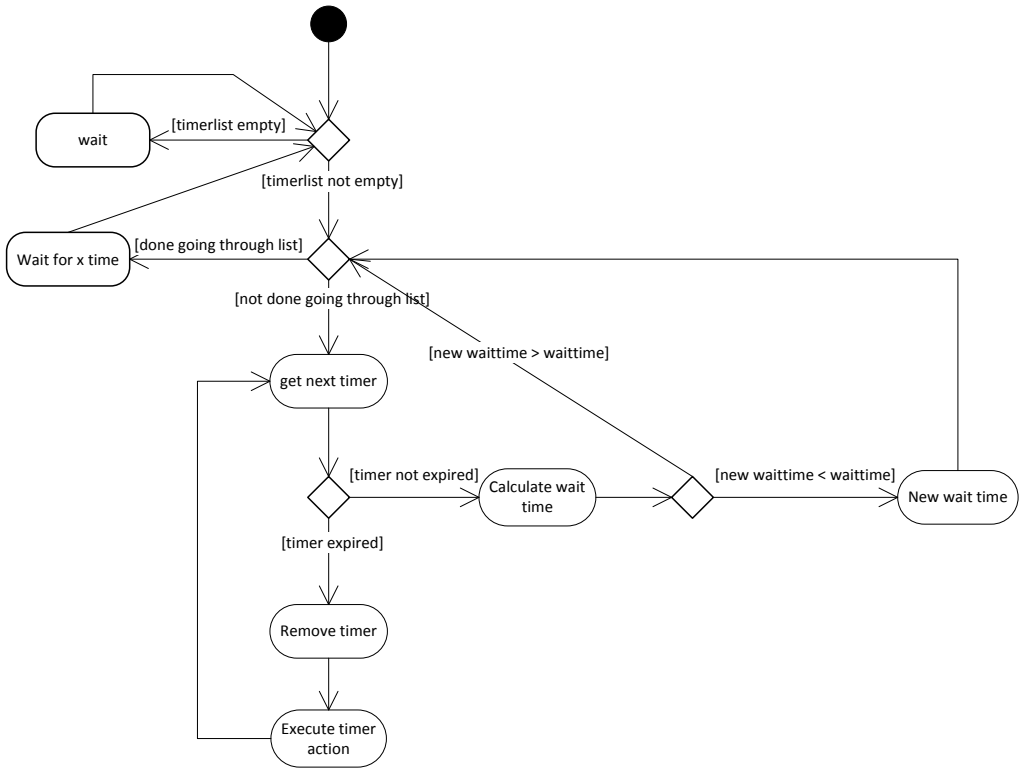


Figure 5.6: Activity diagram for TimeDispatcher.

The activity diagram for the TimeDispatcher can be seen in Figure 5.6. The time dispatcher contains a list of all active timers that have been created. If the timer list contains living timers the time dispatcher goes down the list and checks if the timers are expired. If the timer is expired the time dispatcher removes it and the timer action is executed. If the timer is not expired the time dispatcher calculates the wait time and makes note of it. It then keeps going through the list until it has gone through all the timers. When the time dispatcher is done going through the whole list it sleeps for as long as the shortest wait time on a timer and then starts again. If the timer list is empty the time dispatcher sleeps until a new timer is added to the timer list.

5.2 Behaviour

Behaviours are the operations that an agent which is created from the framework can use or extend. These behaviours make up the core functionality of the agent. The framework behaviours are divided into few types which each have specific execution criteria and action.

5.2.1 Types

The types of behaviours available in the framework are SimpleBehaviour, OneshotBehaviour, CyclicBehaviour and TickerBehaviour. These methods are all derived from the top class Behaviour. The class diagram can be seen in Figure 5.7

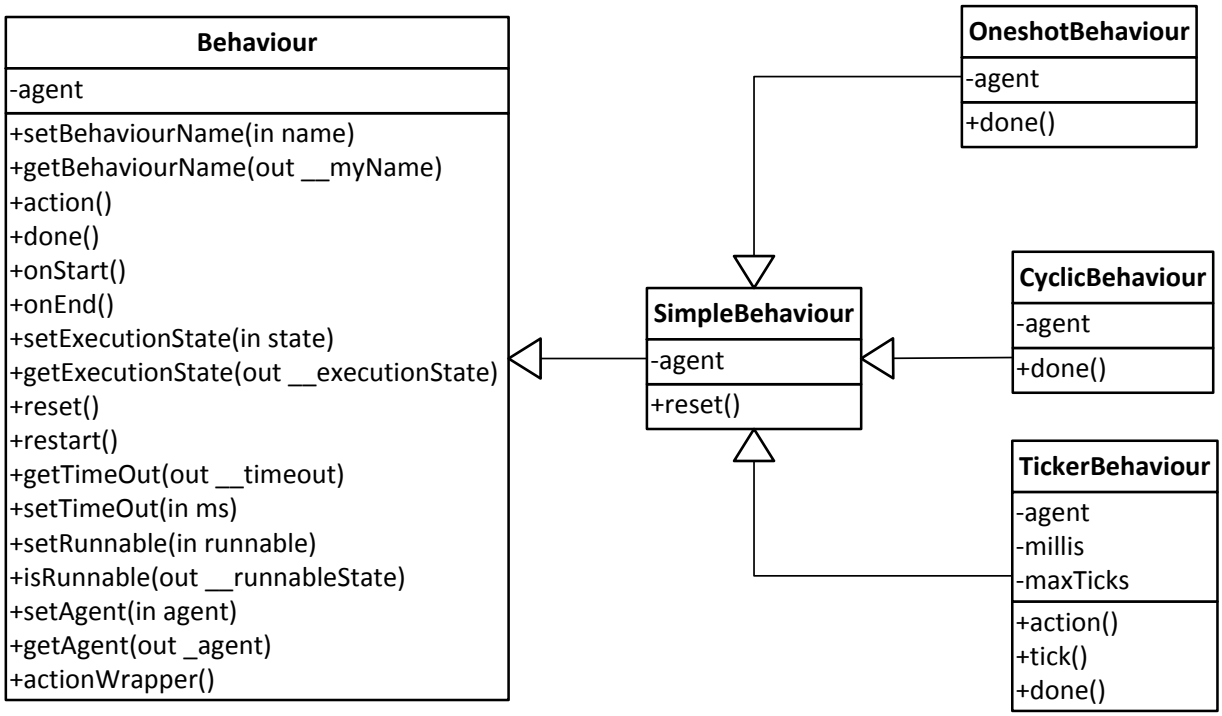


Figure 5.7: Class diagram for the behaviour classes in framework.

Behaviour

This is the top class that implements the basic behaviour methods. It holds the definition of the basic behaviour methods. It also holds the methods that are used to define the running state of the behaviour. See Figure 5.7 for available methods in the Behaviour class.

Simple

The SimpleBehaviour class extends the Behaviour class. It calls the Behaviour class with default values. SimpleBehaviour class is used for making a simple behaviour since it is a shell for the Behaviour class with default values. See Figure 5.7 for methods of this class.

Oneshot

The OneshotBehaviour class extends the SimpleBehaviour class. It calls the SimpleBehaviour class with the default values and also with TRUE value which means the behaviour is only executed ones. OneshotBehaviour class is used when an operation is needed that should only be executed ones. See Figure 5.7 for methods.

Cyclic

The CyclicBehaviour class extends the SimpleBehaviour class. It calls the SimpleBehaviour class with the default values and also with FALSE value which means the behaviour is executed in a cycle until the class is called with a TRUE value. CyclicBehaviour class is called when an operation should be executed in an infinite loop. See Figure 5.7 for methods.

Ticker

The TickerBehaviour class extends the SimpleBehaviour class. It calls the SimpleBehaviour class with the default values except overriding the action method to implement TickerBehaviour. When calling the TickerBehaviour class time between executions and number of executions are defined. The action implements a waiting system for the behaviour so it will only be executed when expected. TickerBehaviour class is called when an operation should be executed number of times with a static interval. See Figure 5.7 for TickerBehavior class methods.

5.2.2 System behaviours

With the other behaviours the framework also implements two system behaviours which are used to divert messages to and from the agent.

Sender behaviour

SenderBehaviour class extends the OneshotBehaviour class. It calls the OneshotBehaviour class by overriding the action method. In the action method it uses a method from a proxy which sends the message that was included in the call to the class.

Receiver behaviour

ReceiverBehaviour class extends the OneshotBehaviour class. It calls the OneshotBehaviour class by overriding the action method. Within the action method it calls a method from the agent class which is used to receive messages. It also gets the message attribute content so message can be translated.

5.3 Agents provided by the framework

5.3.1 Types

The framework provides two kinds of agents out of the box. These agent are the standard agent and the AMS agent. Both these types of agents will be described in the following sections.

Standard agent

The standard agent is the common super class for all agents built using the framework. The standard agent provides all components for the agent to be able to operate. It also provides methods so the agent can perform the basic tasks. The methods and components are for example:

- *Scheduler:* Used to handle behaviour execution. See section 5.3.3 for more information about the scheduler.
- *Time dispatcher:* It is located in the container with the agent when it is created. The agent has a reference to the time dispatcher. The agent uses the time dispatcher to create schedules for behaviour execution. See section 5.1.3 for more details about the time dispatcher.
- *Data store:* Is used for storing information and staging data gathered by the agent. See section 5.3.4 for more information about the data store.
- *Knowledge store:* Is used for storing knowledge which the agent needs to operate. See section 5.3.4 for more information about the knowledge store.
- *Internal message queue*: The queue is used to handle and receiving incoming messages received by the agent.

The agent executes all operations through behaviours. The execution of behaviours takes place in the agent lifecycle which controls the schedule and execution of the behaviours. See section 5.3.2 for more information about the lifecycle.

AMS

The AMS agent is built on the standard agent and provides registering services for running agents. The AMS is a concept taken from FIPA as described in section 2.1.1. The AMS is located in the domain container, for more information about the domain container see section 5.1.1. A new agent who is created must register its name and location to the AMS. The AMS also provides a lookup services for the agent which works like a phonebook services for the agent. The agents use the lookup services to locate another agent to be able to open a communication channel with the other agent it wants to communicate with.

5.3.2 Lifecycle

An agent can only be in a single lifecycle state at any time and within only one agent platform. The lifecycle used in the framework is adapted from FIPA, see section 2.4.3 for more information. Some states from the FIPA standard are not used in the framework to simplify things, the framework only implements the waiting, active and initiated state. Figure 5.8 shows the possible states and the transition between states. The states that were not implemented according to FIPA are suspended state and transit state. They were not

implemented because the agent created did not need to be set to the states suspended or to be moved between agent platforms.

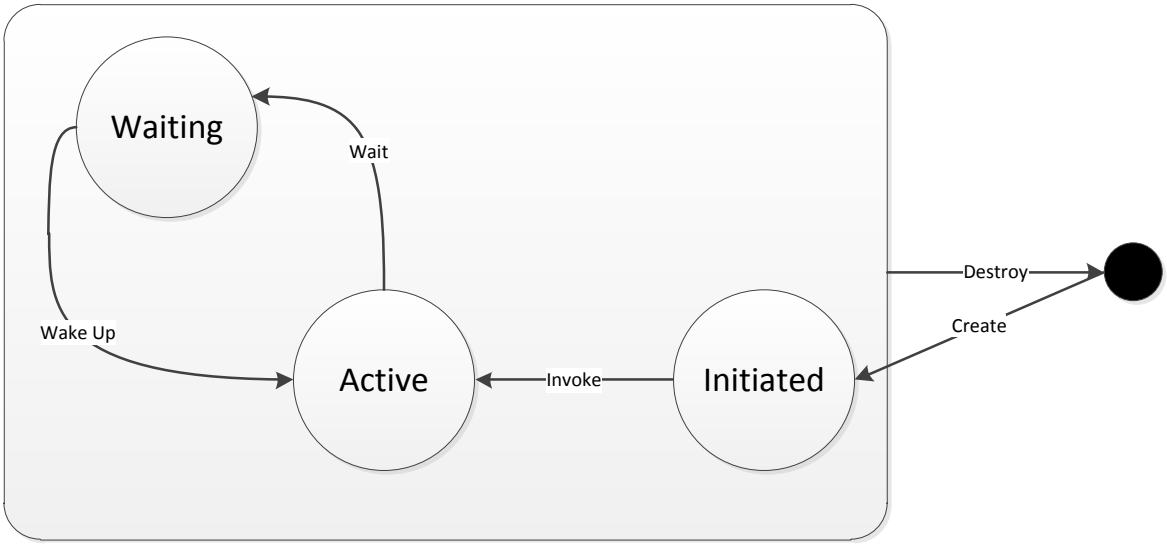


Figure 5.8: Agent lifecycle.

A created agent always starts in the initiated state. An agent is automatically invoked when it has been initiated and added to the active lifecycle where it is fully functionally and able to run all available behaviours. The active lifecycle is responsible for getting the next behaviour in line from the scheduler and execute it. It also makes sure that an executed behaviour has finished running and checking if it should be removed from the ready queue, executed again or terminated. If the behaviour should be executed again the active lifecycle either adds the behaviour to the ready queue by calling the Scheduler or creates a timer for the behaviour if it should be executed after some time interval and adds the timer to the TimeDispatcher. See section 5.3.3 for more information about the Scheduler and section 5.1.3 for more information about the TimeDispatcher.

When there are no available behaviours in the ready queue the Scheduler tells the agent to go to waiting state. When a new behaviour arrives in the ready queue the agent is put back into active state. See Figure 5.9 for a sequence diagram of the agent lifecycle process described in the paragraphs above.

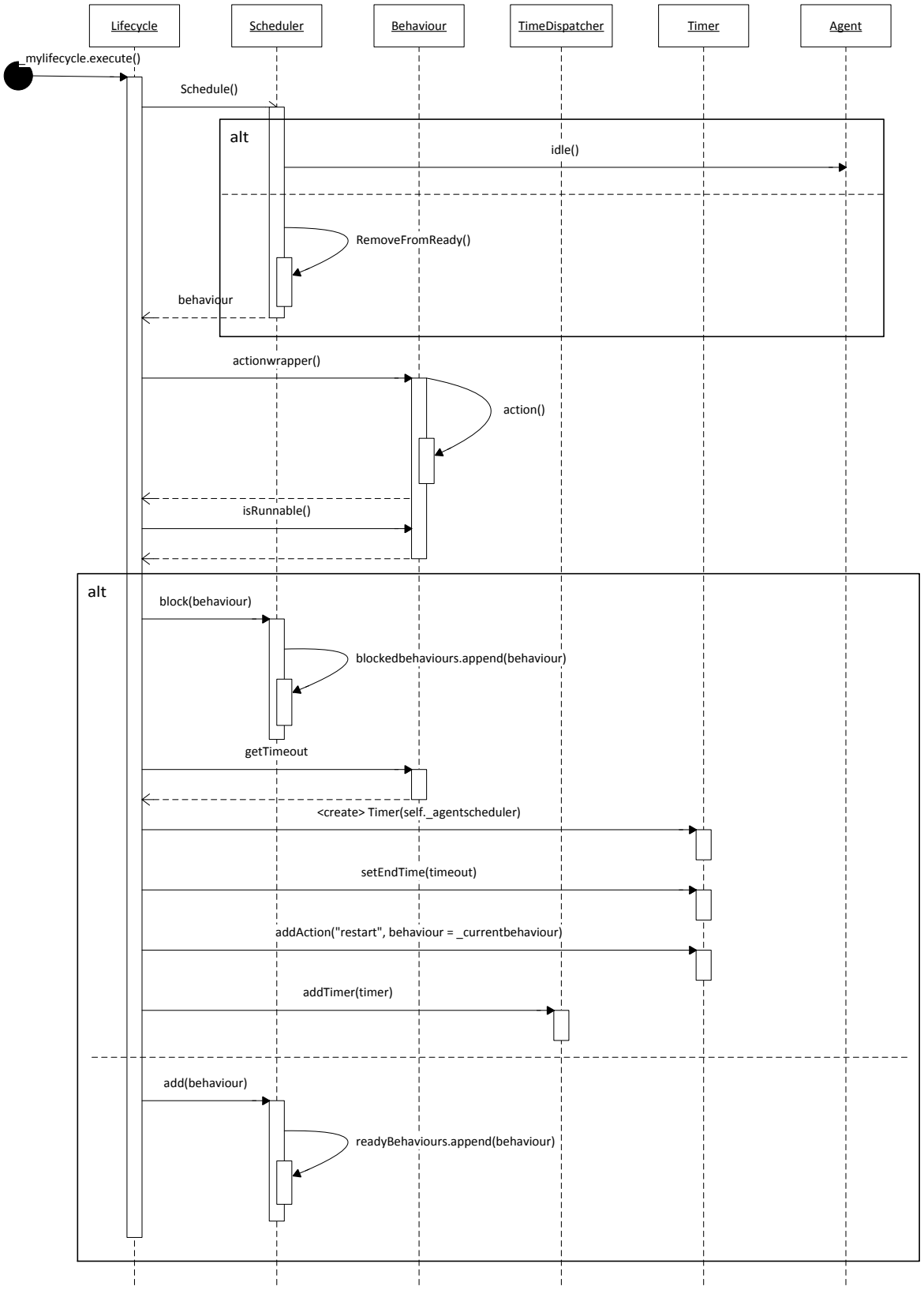


Figure 5.9: Sequence diagram for agent lifecycle process.

5.3.3 Scheduler

The scheduler controls the execution of behaviours. When a request is received from the active-lifecycle for a new behaviour the scheduler gets the behaviour that is next in line from the ready queue. The scheduler removes the behaviour from the ready queue so if the behaviour should be run again it is added back at the end of the ready queue as the last item. If a request is received from the active-lifecycle for a new behaviour and the ready queue is empty the scheduler notifies the agent to go into idle state. This means that the agent will sleep until it gets the message that something has happened, such as a new behaviour is added to the ready queue or a message is received in the agent inbox.

5.3.4 Data storage

Data storage is divided into two separate categories which are represented by modules as described below. Each module in Python can contain many classes. The module datastore.py is used for staging data from the agent and the other data module knowledge.py is used for knowledge data for the agent.

Staging data

The module datastore.py is used to store data that is collected, processed and used by the agents. This data is mostly staging data that is collected information about the server usage. This data is stored in lists which are stored in a hash table which is indexed with unique keys. The identifying key represents the type of staging data, for example memory. The data in the lists are managed by common Python list operators which have been integrated into corresponding methods with descripting names. For example addData, deleteLine and readFirst. The class diagram for DataStore can be seen in Figure 5.10.

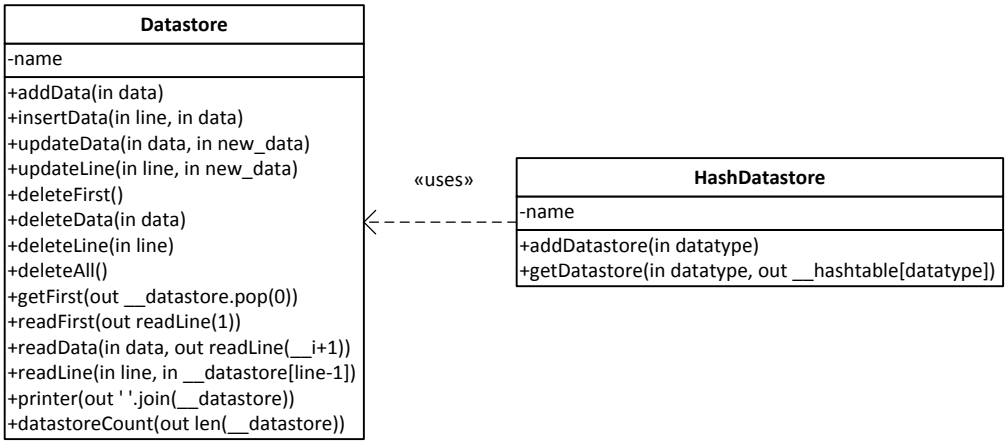


Figure 5.10: Class diagram for datastore.

Knowledge

The module knowledge.py is used to hold data that the agent has to have to be able to process their data correctly and to do their tasks correctly. This knowledge is for example which processes the agents should not monitor. In the knowledge module the agent also stores critical knowledge about for example where other agents are and where the SQL server is and so forth. The knowledge is managed with lists or a single element which is stored in hash table which is indexed by unique keys. The identifying key represents the type of knowledge, for example blocked. An example of knowledge stored as a list is a list of processes that should not be monitored. Knowledge stored as a single element is for example the location of an SQL server. The knowledge in the lists is managed by common Python list operators which have been integrated into corresponding methods with descripting names. For example addKnowledge, deleteKnowledge and readKnowledge. Knowledge stored as a single element is managed by calling the identifying key and then using Python hash table operators which have been integrated into corresponding methods with descripting names. For example addKnowledgeItem, getKnowledgeItem and removeKnowledgeItem. The class diagram for Knowledge can be seen in Figure 5.11.

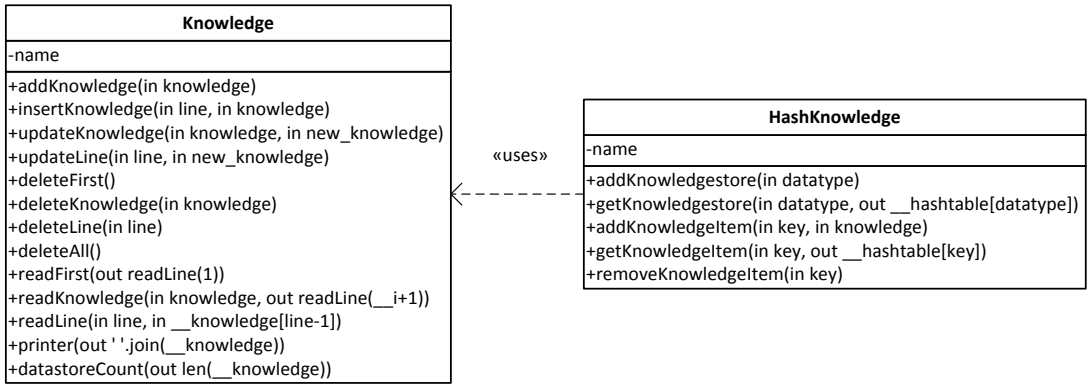


Figure 5.11: Class diagram for knowledge.

6 Agents

Specialized agent was built on top of the created framework. The agents are able to run in multiple different instances with many possible behaviours. Each programmed behaviour handles an ability which is a certain activity the agent can carry out.

6.1 Functionality

The agents main functionalities is to monitor server usage and connections to servers and distribute the collected data to a repository (database). The agent uses behaviours to fulfill these functionalities. Part of the behaviours collect the necessary information needed to prepare the monitoring of the servers or information that it must have to handle the collected data. Other behaviours take care of the monitoring itself. To see more information about these behaviours see section 6.2.

6.2 Behaviour

On top of the behaviour framework six special behaviours were made. Each behaviour is intended to carry out a special task so the agent can fulfill his purpose. These specialized behaviours available for the agent are visible in Figure 6.1 which is an extended version of Figure 5.7. In addition of the behaviour classes in the framework (showed with dotted lines in Figure 6.1) it shows the customized behaviours for the agents.

The following classes are the classes from the framework (CyclicBehaviour is not shown since it is not needed here):

- Behaviour
- SimpleBehaviour
- TickerBehaviour
- OneshotBehaviour

These are the classes that have been added and make use of the behaviours implemented in the framework:

- SQLBehaviour
- InitialProcessBehaviour
- ProcessBehaviour
- SniffingBehaviour
- ConnectionMonitorBehaviour
- Processor Behaviour
- MemoryBehaviour

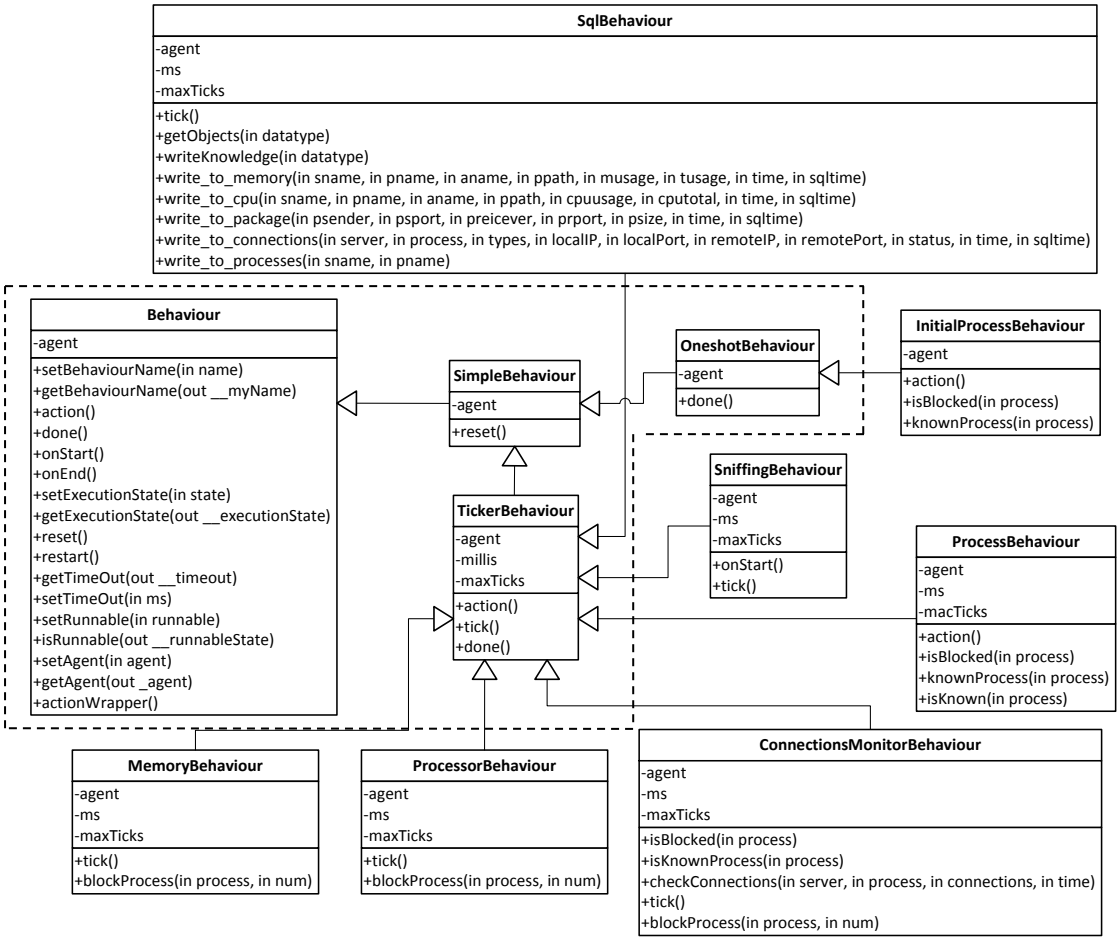


Figure 6.1: Class diagram for the agents behaviour classes.

6.2.1 Process

The ProcessBehaviour class collects information about processes running on server as parameters and stores as objects. These parameters are process name and process id. It can be called in two separate ways as two classes. These two classes are InitialProcessBehaviour class which extends the OneshotBehaviour class and the ProcessBehaviour class which extends the TickerBehaviour class. The InitialProcessBehaviour class can only be called if no ProcessBehaviour classes have been called before. The InitialProcessBehaviour class is executed each time an agent is created. The ProcessBehaviour class can be called at any time.

The InitialProcessBehaviour class extends the OneshotBehaviour class by overriding the action method. It implements new methods that are used for checking if the process is on the blocked list or not and a method that adds the process to a list of known processes. Methods can be seen in Figure 6.1. Within the action method it lists up all processes running on the server and checks if each process should be blocked or not. If the process should be blocked the process is added to the knowledge store with known processes that should not be monitored. Initial process behaviour class is called at start of agent execution to collect all processes that should not be monitored.

The ProcessBehaviour class extends the TickerBehaviour class by overriding the tick method. It implements new methods that are used for checking if process is on the blocked list or not, a method that check if process is known or not and a method that adds the process to a list of known processes. Methods can be seen in Figure 6.1. Within the tick method it lists up all processes running on the server and checks if each process should be blocked or not. If the process should be blocked it checks if it is already blocked and if not the process is added to the knowledge store with known processes that should not be monitored. ProcessBehaviour class is called to update the list of processes that should not be monitored.

6.2.2 Memory

The MemoryBehaviour class collects information about memory usage of chosen processes. It extends the TickerBehaviour class by overriding the tick method. The MemoryBehaviour class goes through all known processes from the process knowledge store and creates an object with the known memory parameters about the process. These paramaters are server name, process name, process description, process executable path, process memory usage, total memory usage of server and timestamp. This object is then added to the memory data store. The MemoryBehaviour uses PSUTIL to get the information about the memory usage, see section 2.3.1 for more information about PSUTIL. If the process is non-existing it is deleted from the process knowledge store. If no access can be gained to the process it is permanently blocked. Methods for the behaviour can be seen in Figure 6.1.

6.2.3 Processor

The ProcessorBehaviour class collects data about processor usage for chosen process. It extends the TickerBehaviour class by overriding the tick method. The ProcessBehaviour class goes through all known process from the process knowledge store and creates an object with the known processor parameters about the process. These paramaters are server name, process name, process description, process executable path, process processor usage, total processor usage of server and timestamp. This object is then added to the cpu data store. The ProcessBehaviour uses PSUTIL to get the information about the processor usage, see section 2.3.1 for more information about PSUTIL. If the process is non-existing it is deleted from the process knowledge store. If no access can be gained to the process it is permanently blocked. Methods for the behaviour can be seen in Figure 6.1.

6.2.4 SQL

The SQLBehaviour class is used to write data or information to an SQL database. It extends the TickerBehaviour class by overriding the tick() method. It implements methods that are used to get objects from data store and knowledge store and writing them to the corresponding SQL table. See Figure 6.1 for the class diagram of the behaviour. The behaviour gets items from the data store or knowledge store and then calls the corresponding write_to method which writes the data to the correct SQL table at a selected location. It writes all selected parameters for the given data store or knowledge store item along with a write timestamp. The write timestamp is the time when the behaviour started looping through either data store or knowledge store, so all items in the data store are given the same write timestamp so the behaviour execution can be identified in the database. The

SQLBehaviour uses PYODBC to connect to the database server, see section 2.3.2 for more information about PYODBC.

6.2.5 Connection Monitor

The ConnectionMonitorBehaviour monitors all active connections to processes for all processes that the agents should be monitoring. The behaviour extends the TickerBehaviour class by overriding the tick() method. The ConnectionMonitorBehaviour uses PSUTIL (see section 2.3.1) for monitoring of connections. The behaviour checks all processes listed in the running processes list in the agent knowledge store and checks if it is still running on the server. If the process is not active when the behaviour checks it removes the process from the list. If the process is active when the behaviour checks then the behaviour uses a tool from the PSUTIL to list up all active connections behind the process, see section 2.3.1 for more information about PSUTIL. By doing this the agent is able to check if any new connections have emerged since the process was last checked. All connections that are discovered are stored in the agents knowledge store for further use.

6.2.6 Sniffing

The SniffingBehaviour monitors incoming network to the servers for the process that should be monitored on the servers that the agents are running on. It extends the TickerBehaviour class by overriding the tick() method. The behaviour creates a raw socket which it binds to the public network interface. It sniffs all network traffic to the server for a predefined time which is declared when the behaviour is called. The behaviour extracts the needed information from the network packages and stores in an object. This information is the source IP address and port and the destination IP address and port. The object is then stored in the datastore. This is done for all messages that are received during the behaviour execution.

7 Post processing

As described in chapter 6 the programmed behaviours use PSUTIL to collect the suitable data and then use PYODBC for storing the data in the corresponding tables in an SQL database on the appropriate server. The data is stored in memory heaps on the local server before it is sent to the SQL database where it is permanently stored. All data is collected to an internal data store as is described in chapter 5.3.4. The data from the data store is then written to an SQL database with the SQLBehaviour (see section 6.2.4 for more information about SQLBehaviour and sections 6.2.2 and 6.2.3 for more information about MemoryBehaviour and ProcessorBehaviour).

7.1 Database structure

The data is divided into three schemas dbo, import and history. The import schema holds the tables used for staging data from the agents. The following tables are in the import schema:

- *import.rawConnections:* Holds staging data for the connection behaviour, see section 6.2.5.
- *import.rawPackages*: Holds staging data for the sniffing behaviour, see section 6.2.6.
- *import.rawProcessCPU:* Holds staging data for the processor behaviour, see section 6.2.3.
- *import.rawProcessMemory:* Holds staging data for the memory behaviour, see section 6.2.2

The history schema holds archived staging data from the raw import tables mentioned above along with the timestamp of archiving time.

- *history.Connections:* Holds archived data from table import.rawConnections.
- *history.Packages*: Holds archived data from table import.rawPackages.
- *history.ProcessCPU:* Holds archived data from table import.rawProcessCPU.
- *history.ProcessMemory:* archived data from table import.rawProcessMemory.

The dbo schema holds tables that are used for creating the cost model calculations. These tables contain the processed data from the staging tables. In the dbo schema are also a table that contains the blocked processes that should not be monitored by the agents and a table of known servers. The following are the tables in the dbo schema:

- *dbo.BlockedProcesses:* Holds the processes that agents should not monitor. Used in the process behaviour, see section 6.2.1.
- *dbo.Connections:* Holds processed data for the collected server connection information with information about connecting servers.
- *dbo.ConnectionsLink:* Holds processed data for the collected server connection information with information about the connecting process.

- *dbo.Packages:* Holds processed data for the collected package sending between server information.
- *dbo.ProcessCPU:* Holds processed data for the processor information on servers.
- *dbo.ProcessMemory:* Holds processed data for the memory information on servers.
- *dbo.Servers:* Holds all known servers and their IP addresses.

7.2 Data processing

Data was imported to staging tables from the agents data stores. To clean up the data and prepare it for use in cost model calculations it was transformed with SQL server integration services. The data transformation process is pictured in Figure 7.1 and described in the following.

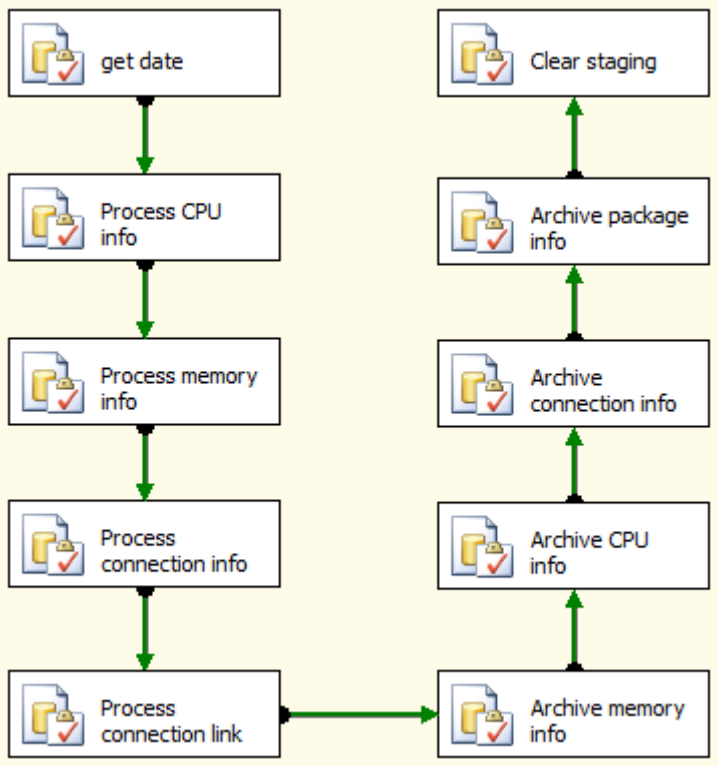


Figure 7.1: SSIS data transformation process.

So the transformation process was divided into these ten steps. The main tasks are getting execution date, process data, archive staged data and then clear staging tables.

1. *Get date:* Stores the current date as a variable for the following data steps.
2. *Process CPU info:* Gets data from ProcessCPU staging table and joins all data before the point the step runs into one field of data. All data is summed together and then the average calculated and stored in the ProcessCPU table with the timestamp created in step 1.
3. *Process memory info:* Gets data from ProcessMemory staging table and joins all data before the point the step runs into one field of data. All data is summed together and then the average calculated and stored in the ProcessMemory table with the timestamp created in step 1.

4. *Process connection info:* All connections over the period used for calculation is summed together.
5. *Process connection link:* All connection over the period linked with the process on connecting server which opened the connection is summed together.
6. *Archive memory info:* Copy data from memory staging table to archive table.
7. *Archive CPU info:* Copy data from CPU staging table to archive table.
8. *Archive connection info:* Copy data from connection staging table to archive table.
9. *Archive package info:* Copy data from package staging table to archive table.
10. *Clear staging:* Clears all data from staging tables that was used in the steps 2, 3, 4 and 5. So all data with the timestamp smaller than the one created in step 1 is deleted.

After these ten steps the data is ready to be used in cost model calculations and presented in a report.

7.3 Data presentation

To present the outcome of the cost model a report was built using SQL Server Reporting Services. The report was built to take transformed data after it had been processed and run it through the calculations for the cost model, as described in chapter 0. After the data has been processed with the calculations it gets presented by the report. The report shows the processed usage of given server for a given time chosen by the report user. The report is structured according to the cost model described in chapter 0. It uses memory, processor and network information to estimate the server usage. The user selects the server (s)he wants to calculate usage off, the time (s)he wants to monitor, the processes that should be charged and the calculated cost of the server. Two reports are created: One that shows the cost model for a given time and one that shows the cost model for a time period. For the report for a given time the user selects a single time id. The report shows the usage percentage of what each connecting process on the connecting server has on the monitored server. The report also shows the usage percentage of each connecting server. See Figure 7.2 for an example of the cost model report for a given time with the cost of ten million. The user can input values into and modify values in the input fields at the top of the report. The server chosen is the server the cost model should be calculated for, the time is what sampled data should be used, the processes entry determines which processes monitored should pay for the server and cost is the total chargeable amount of the overall operational costs of the server.

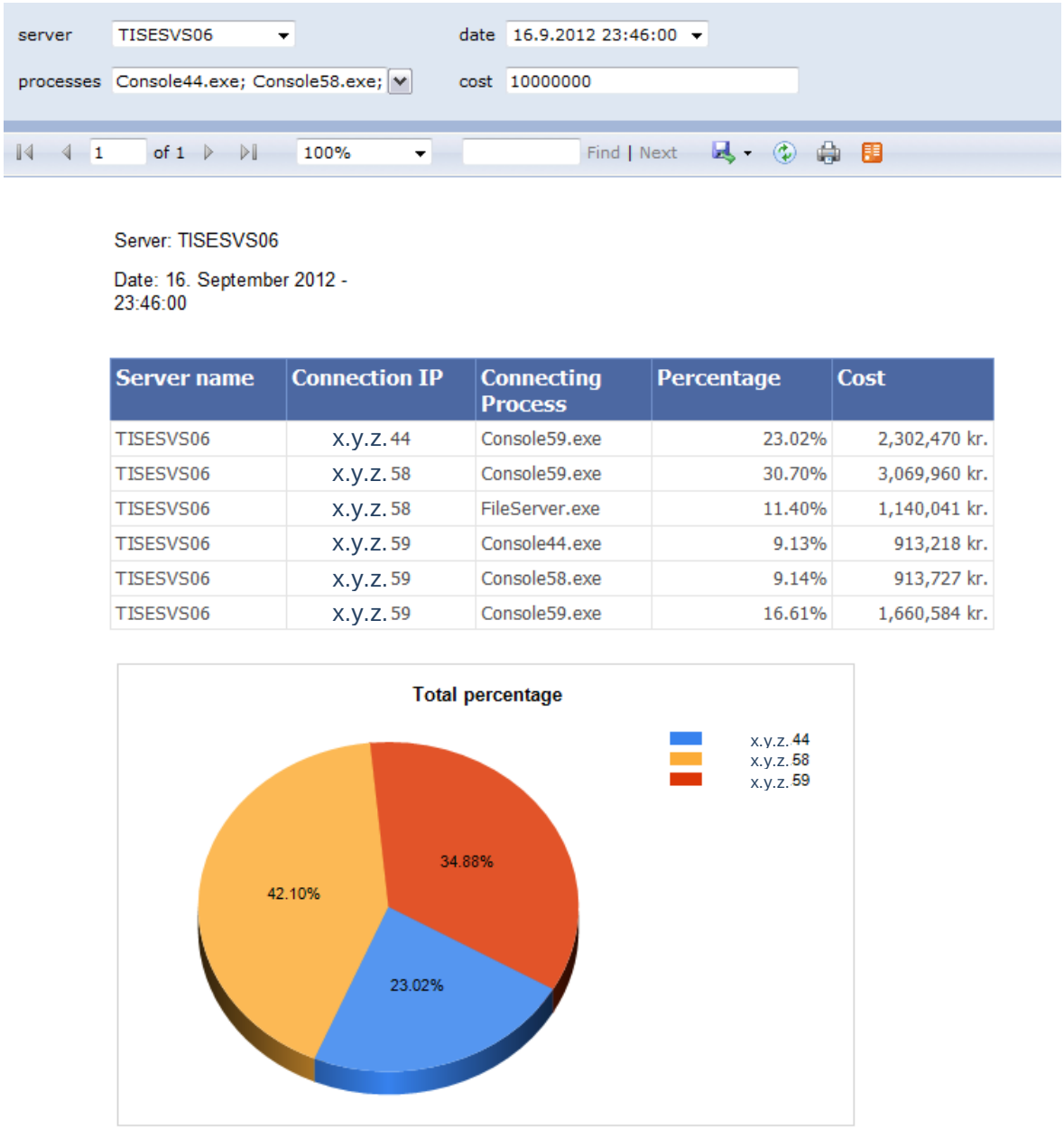


Figure 7.2: Cost Model report for a given time id.

For the report for time periods the user both selects to and from date. See Figure 7.3 for an example of the cost model report for a given time period with cost ten million.

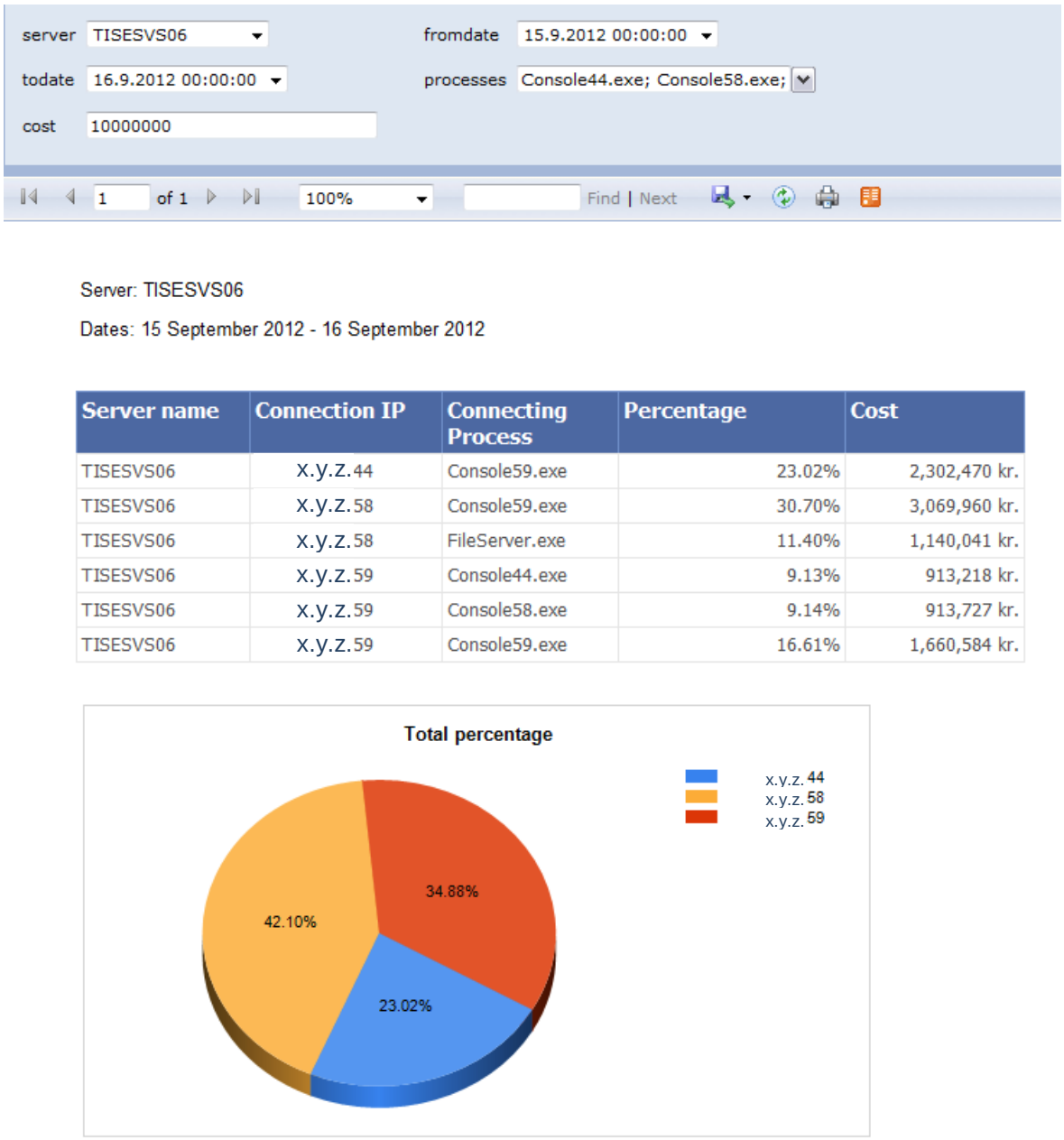


Figure 7.3: Cost model report for a time period.

8 Evaluation

When sampling data from servers the corresponding agent behaviours were executed in an arbitrary but pre-decided order. By doing that the agents only gather pieces of data flowing from and to servers. To see what affect the order of executed behaviours and the timing of executed behaviour would have a special study of their execution was done. This gives us a better vision on how much influence the order of executed behaviours has on the cost model.

For the evaluation of the agent behaviours it was presumed that the data gathered by the agents was correct. For evaluation of the quality of gathered data please refer to Einar Sveinsson's thesis [1].

8.1 Environment

To test the created agents a multi-server environment was created which consisted of three servers:

- TISESVS02 with IP x.y.z.44
- TISESVS04 with IP x.y.z.58
- TISESVS06 with IP x.y.z.59

On each server the created agent system was initialized with both domain container and agent container. To be able to exclude the data that should be monitored a specific number of processes were defined and all other processes running on the server were added to the list of blocked processes stored on the database server and imported to the knowledge store during execution. The processes were:

- *Console44.exe:* Client process that connects to the FileServer.exe process on server TISESVS02 (x.y.z.44).
- *Console58.exe:* Client process that connects to the FileServer.exe process on server TISESVS04 (x.y.z.58).
- *Console59.exe:* Client process that connects to the FileServer.exe process on server TISESVS06 (x.y.z.59).
- *FileServer.exe:* File server that gets a request from a console client to create a file. It stores the file in folder c:/temp. Each file contains previously decided number of random strings which the server gets from StringServer.exe that runs on a different server.
- *StringServer.exe:* String server that created a random string with specific number of characters and returns to the connected client, in this case FileServer.exe running on a different server.

These processes were the same on all servers. The FileServer.exe and StringServer.exe were configured in a previously defined order:

- FileServer.exe on TISESVS02 (44) connected to StringServer.exe on TISESVS04 (58).
- FileServer.exe on TISESVS04 (58) connected to StringServer.exe on TISESVS06 (59).
- FileServer.exe on TISESVS06 (59) connected to StringServer.exe on TISESVS02 (44).

The structure of the test environment logic can be seen in Figure 8.1. The console client in the figure can be any of the three console clients.

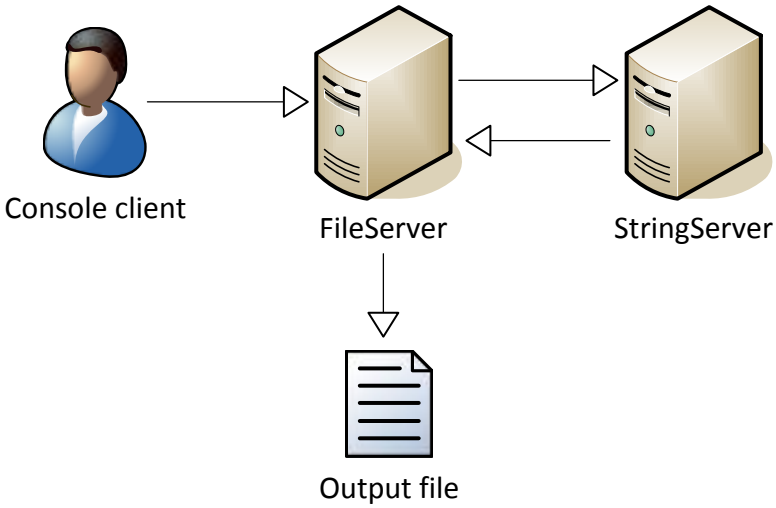


Figure 8.1: Structure of the test environment.

The internals of the test environment seen in Figure 8.1 was structured in the way that a console client sent a message to a file server on any server with a predefined number of lines, length of each line, number of files, wait in seconds between files and a tick whether the wait should be a random. If the wait should be random the process for example waits for 0 to 10 seconds if the input wait between files was 10 seconds. The file server then sends a message to the string server on a predefined server which created a random string with a predefined number of characters and returns it to the file server. The file server then writes the string to a file. This is repeated between the file server and string server until the number of lines and files is fulfilled. A screen shot of the user console client can be seen in Figure 8.2.

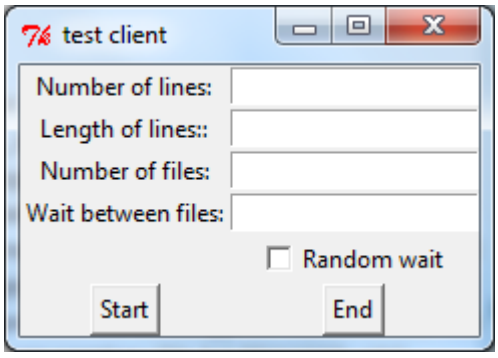


Figure 8.2: The console client user interface.

8.2 Scenarios

To see the effect of the time schedules of behaviours during execution on the cost model five different scenarios for data gathering where implemented and executed. Each scenario was executed both with standard time and random time resulting in ten test cases. With input time of 10 second standard time means 10 seconds between files while random means 0 to 10 seconds between files. The same configurations of the consoles were executed for each scenario. These configurations can be seen in Figure 8.3 for standard time and Figure 8.4 for random time.

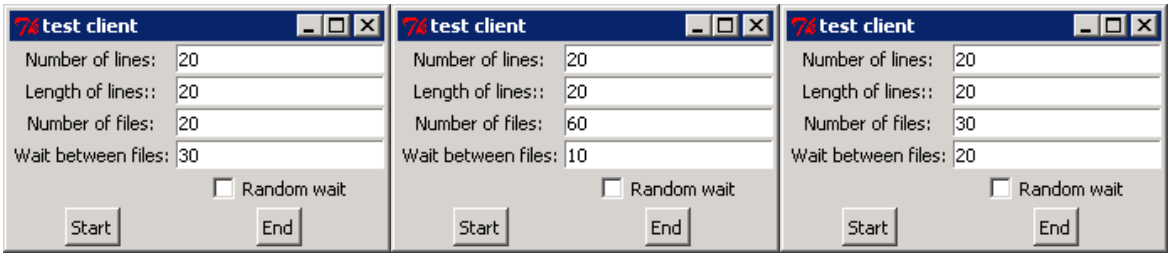


Figure 8.3: Configuration for standard time.

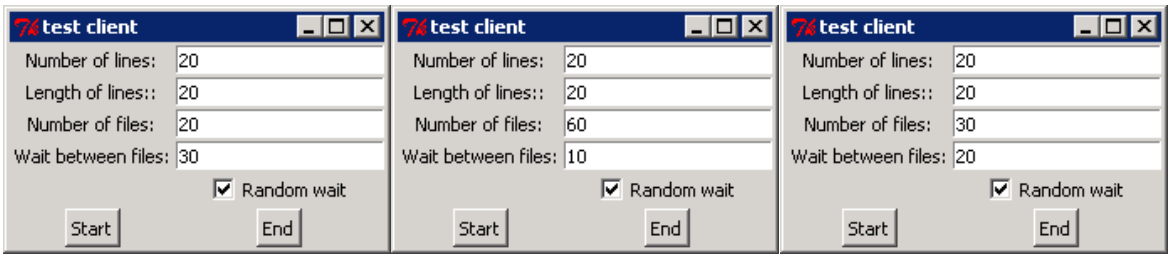


Figure 8.4: Configuration for random time.

These configurations were divided between servers so each server would get the same number of files, so the same configuration of clients was not executed on all servers. The same configuration was though used in all scenarios. The configurations of each server where the following (number of lines; length of lines; number of files; wait between files):

- *TISESVS02 (44):* Console44: (20; 20; 60; 10), Console58 (20; 20; 30; 20), Console59 (20; 20; 20; 30).
- *TISESVS04 (58):* Console44: (20; 20; 30; 20), Console58 (20; 20; 20; 30), Console59 (20; 20; 60; 10).
- *TISESVS06 (59):* Console44: (20; 20; 20; 30), Console58 (20; 20; 60; 10), Console59 (20; 20; 30; 20).

All the clients were executed on the same servers as the services. This was done to avoid using computers or servers outside the environment. To have as little interference of agent behaviours as possible during execution only the agent behaviours needed for the cost model were executed. These behaviours were:

- InitialProcessBehaviour
- ConnectionsMonitorBehaviour
- MemoryBehaviour

- ProcessorBehaviour
- SqlBehaviour

InitialProcessBehaviour is only executed at the start of execution so no schedule modification is possible. SqlBehaviour does not sample any data it only reads from datastore and writes to an external database so scheduling changes of that behaviour does not have any effect on data. The effect of the sampling behaviours ConnectionsMonitorBehaviour, MemoryBehaviour and ProcessorBehaviour are the only ones that have an effect on data so changes on schedules for these behaviours were evaluated. I will briefly explain the scenarios here but more details about each can be found in the correspondent sections below.

- The first scenario was to execute all the behaviours with the decided base schedule, see section 8.2.1 for more details.
- The second scenario was to execute all the behaviours with the MemoryBehaviour and ProcessorBehaviour time between increased, see section 8.2.2 for more details.
- The third scenario was to execute all behaviours with the MemoryBehaviour and ProcessorBehaviour time between decreased, see section 8.2.3 for more details.
- The fourth scenario was to execute all the behaviours with the ConnectionsMonitorBehaviour time between increased, see section 8.2.4 for more details.
- The fifth scenario was to execute all behaviours with the ConnectionsMonitorBehaviour time between decreased, see section 8.2.5 for more details.

For all scenarios the number of ticks (or runs) the agent behaviour should fulfill was set to 100000 so that the execution of agent behaviours would not end before all files had been created.

8.2.1 First scenario

The first scenario executed was executing all the behaviours with pre decided base schedules. This base configuration was chosen randomly and was done so comparison of scenarios could be done to the same base scenario. By doing this it was possible to compare the changes in usage between the scenarios by referring this base scenario. This scenario was executed both for the configurations in Figure 8.3 and Figure 8.4. The schedule used was:

- InitialProcessBehaviour (standard)
- ConnectionsMonitorBehaviour (time between: 2000 ms, number of runs: 100000)
- MemoryBehaviour (time between: 5000 ms, number of runs: 100000)
- ProcessorBehaviour (time between: 5000 ms, number of runs: 100000)
- SqlBehaviour (time between: 60000 ms, number of runs: 100000)

8.2.2 Second scenario

The second scenario executed was to increase the time between executions for behaviours MemoryBehaviour and ProcessorBehaviour to 10000 milliseconds. This was done to see if

by getting less sample data about memory and processor usage would affect the cost model output. The schedule used was:

- InitialProcessBehaviour (standard)
- ConnectionsMonitorBehaviour (time between: 2000 ms, number of runs: 100000)
- MemoryBehaviour (time between: 10000 ms, number of runs: 100000)
- ProcessorBehaviour (time between: 10000 ms, number of runs: 100000)
- SqlBehaviour (time between: 60000 ms, number of runs: 100000)

8.2.3 Third scenario

The third scenario executed was to decrease the time between executions for behaviours MemoryBehaviour and ProcessorBehaviour to 1000 milliseconds. This was done to see if by getting more sample data about memory and processor usage would affect the cost model output. The schedule used was:

- InitialProcessBehaviour (standard)
- ConnectionsMonitorBehaviour (time between: 2000 ms, number of runs: 100000)
- MemoryBehaviour (time between: 1000 ms, number of runs: 100000)
- ProcessorBehaviour (time between: 1000 ms, number of runs: 100000)
- SqlBehaviour (time between: 60000 ms, number of runs: 100000)

8.2.4 Fourth scenario

The fourth scenario executed was to increase the time between executions for behaviour ConnectionsMonitorBehaviour to 10000 milliseconds. This was done to see if by getting less sample data about connecting systems would affect the cost model output. The schedule used was:

- InitialProcessBehaviour (standard)
- ConnectionsMonitorBehaviour (time between: 10000 ms, number of runs: 100000)
- MemoryBehaviour (time between: 5000 ms, number of runs: 100000)
- ProcessorBehaviour (time between: 5000 ms, number of runs: 100000)
- SqlBehaviour (time between: 60000 ms, number of runs: 100000)

8.2.5 Fifth scenario

The fifth scenario executed was to decrease the time between executions for behaviour ConnectionsMonitorBehaviour to 500 milliseconds. This was done to see if by getting more sample data about connecting systems would affect the cost model output. The schedule used was:

- InitialProcessBehaviour (standard)
- ConnectionsMonitorBehaviour (time between: 500 ms, number of runs: 100000)
- MemoryBehaviour (time between: 5000 ms, number of runs: 100000)
- ProcessorBehaviour (time between: 5000 ms, number of runs: 100000)
- SqlBehaviour (time between: 60000 ms, number of runs: 100000)

8.3 Obtained data

For each test execution the cost model report was executed and the results stored alongside the corresponding data. For the results see Appendix A. In Appendix A subsections show the result of each test scenario and their explanation. As said earlier each scenario consists of two runs, one with standard timing and one with random timing. The results will be analyzed in section 8.4.

8.4 Comparison of scenarios

To get a better view of the evaluation the results from section 8.3 and Appendix A respectively were united for each server. These results are presented in two tables. One table that shows the percentage usage for each process of a connecting server and one table that shows percentage change in usage for each scenario in regards of the base scenario. The later table also shows the minimum change, the maximum change and the average change for each process through all scenarios.

To understand the results better result tables were created in the following subsections which show the percentage change of usage between the corresponding scenario to the base scenario or scenario 1. Decreasing change in these tables is shown with minus sign in front of the number while increase is shown with no sign. The tables also show the minimum change, the maximum change and the average change to help with analysis of the result data. An example of these tables is Table 8.4.2.

8.4.1 Results for TISESVS02

In the following subsections the two result tables for server TISESVS02 (44) will be presented and discussed. There are two subsections one for standard timing and one for random timing.

Standard timing

Table 8.4.1 shows the percentage usage of server TISESVS02 (44) for each connecting process for all scenarios with standard timing.

Usage of TISESVS02 (44), standard timing						
Connecting Process	Connection IP	Scenario1	Scenario2	Scenario3	Scenario4	Scenario5
FileServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.44	16,62%	14,27%	19,27%	13,68%	26,39%
Console58.exe	x.y.z.44	9,07%	11,15%	9,02%	9,01%	18,02%
Console59.exe	x.y.z.44	9,09%	11,41%	9,04%	9,03%	18,05%
FileServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.58	7,33%	5,16%	8,54%	3,75%	8,24%
Console58.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%
Console59.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%
FileServer.exe	x.y.z.59	50,47%	52,81%	43,91%	61,40%	22,81%
StringServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.59	7,42%	5,19%	10,21%	3,12%	6,49%
Console58.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%
Console59.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%

Table 8.4.1: Results for TISESVS02, standard.

Table 8.4.2 shows the percentage change of usage between the corresponding scenario to the base scenario or scenario 1.

TISESVS02 (44): Percentage changes from Scenario 1								
Connecting Process	Connection IP	Scenario 2	Scenario 3	Scenario 4	Scenario 5	MIN	MAX	AVG
FileServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.44	-2,35%	2,65%	-2,94%	9,77%	2,35%	9,77%	4,43%
Console58.exe	x.y.z.44	2,08%	-0,05%	-0,06%	8,95%	0,05%	8,95%	2,79%
Console59.exe	x.y.z.44	2,32%	-0,05%	-0,06%	8,96%	0,05%	8,96%	2,85%
FileServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.58	-2,17%	1,21%	-3,58%	0,91%	0,91%	3,58%	1,97%
Console58.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console59.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
FileServer.exe	x.y.z.59	2,34%	-6,56%	10,93%	-27,66%	2,34%	27,66%	11,87%
StringServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.59	-2,23%	2,79%	-4,30%	-0,93%	0,93%	4,30%	2,56%
Console58.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console59.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%

Table 8.4.2: Percentage changes for TISESVS02, standard.

As can be seen in Table 8.4.2 the greatest difference in change of usage is for the FileServer.exe on x.y.z.59 or 11,87% average change. This high average can be traced to the results of FileServer.exe in scenario 5. All other average changes are within 5% which could indicate that changes on behaviour scheduling is not affecting the outcome that much. The result for FileServer.exe will though have to be taken into an account and kept in mind for the remaining results analysis.

Random timing

Table 8.4.3 shows the percentage usage of server TISESVS02 (44) for each connecting process for all scenarios with random timing.

Usage of TISESVS02 (44), random timing						
Connecting Process	Connection IP	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
FileServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.44	18,02%	16,96%	19,89%	19,15%	25,30%
Console58.exe	x.y.z.44	9,05%	9,04%	9,01%	9,03%	18,01%
Console59.exe	x.y.z.44	9,06%	10,68%	9,03%	9,05%	18,05%
FileServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.58	8,84%	6,41%	10,44%	5,68%	6,94%
Console58.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%
Console59.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%
FileServer.exe	x.y.z.59	46,43%	48,16%	42,34%	52,50%	22,82%
StringServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.59	8,60%	8,76%	9,28%	4,58%	8,88%
Console58.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%
Console59.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%

Table 8.4.3: Results for TISESVS04, random.

Table 8.4.4 shows the percentage change of usage between the corresponding scenario to the base scenario or scenario 1.

TISESVS02 (44): Percentage changes from Scenario 1								
Connecting Process	Connection IP	Scenario 2	Scenario 3	Scenario 4	Scenario 5	MIN	MAX	AVG
FileServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.44	-1,06%	1,87%	1,13%	7,28%	1,06%	7,28%	2,84%
Console58.exe	x.y.z.44	-0,01%	-0,04%	-0,02%	8,96%	0,01%	8,96%	2,26%
Console59.exe	x.y.z.44	1,62%	-0,03%	-0,01%	8,99%	0,01%	8,99%	2,66%
FileServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.58	-2,43%	1,60%	-3,16%	-1,90%	1,60%	3,16%	2,27%
Console58.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console59.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
FileServer.exe	x.y.z.59	1,73%	-4,09%	6,07%	-23,61%	1,73%	23,61%	8,88%
StringServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.59	0,16%	0,68%	-4,02%	0,28%	0,16%	4,02%	1,29%
Console58.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console59.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%

Table 8.4.4: Percentage changes for TISESVS02, random.

By looking at Table 8.4.4 it can be seen that only one average of usage change is over 5%. That is for the FileServer.exe on server x.y.z.59 which has the average of 8,88%. This is the same element that caused the high average for standard timing so it could be concluded that the result is not faulted but instead the short time between behaviour execution is affecting the outcome for the process. The FileServer.exe process which is causing this result is the process connecting to the local StringServer.exe process on the server.

8.4.2 Results for TISESVS04

In the following subsections the two result tables for server TISESVS04 (58) will be presented and discussed. There are two subsections one for standard timing and one for random timing.

Standard timing

Table 8.4.5 shows the percentage usage of server TISESVS04 (58) for each connecting process for all scenarios with standard timing.

Usage of TISESVS04 (58), standard timing						
Connecting Process	Connection IP	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
FileServer.exe	x.y.z.44	48,46%	39,33%	44,31%	55,88%	22,85%
StringServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%
Console58.exe	x.y.z.44	6,04%	9,62%	7,37%	4,01%	5,88%
Console59.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%
FileServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.58	9,19%	10,49%	9,12%	12,88%	18,29%
Console58.exe	x.y.z.58	18,00%	21,56%	18,23%	13,88%	27,24%
Console59.exe	x.y.z.58	9,07%	9,18%	9,68%	9,02%	18,06%
FileServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%
Console58.exe	x.y.z.59	9,24%	9,82%	11,29%	4,33%	7,70%
Console59.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%

Table 8.4.5: Results for TISESVS04, standard.

Table 8.4.6 shows the percentage change of usage between the corresponding scenario to the base scenario or scenario 1.

TISESVS04 (58): Percentage changes from Scenario 1								
Connecting Process	Connection IP	Scenario 2	Scenario 3	Scenario 4	Scenario 5	MIN	MAX	AVG
FileServer.exe	x.y.z.44	-9,13%	-4,15%	7,42%	-25,61%	4,15%	25,61%	11,58%
StringServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console58.exe	x.y.z.44	3,58%	1,33%	-2,03%	-0,16%	0,16%	3,58%	1,78%
Console59.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
FileServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.58	1,30%	-0,07%	3,69%	9,10%	0,07%	9,10%	3,54%
Console58.exe	x.y.z.58	3,56%	0,23%	-4,12%	9,24%	0,23%	9,24%	4,29%
Console59.exe	x.y.z.58	0,11%	0,61%	-0,05%	8,99%	0,05%	8,99%	2,44%
FileServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console58.exe	x.y.z.59	0,58%	2,05%	-4,91%	-1,54%	0,58%	4,91%	2,27%
Console59.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%

Table 8.4.6: Percentage changes for TISESVS04, standard.

Once again all average of usage change are within 5% except for process FileServer.exe on server x.y.z.44 which has the average of 11,58%. This high average is caused by the result of scenario 5 for the FileServer.exe process. This supports the hypothesis that having a short time between executions of the connection monitoring behaviour affects the FileServer.exe connection to the local StringServer.exe on the server.

Random timing

Table 8.4.7 shows the percentage usage of server TISESVS04 (58) for each connecting process for all scenarios with random timing.

Usage of TISESVS04 (58), random timing						
Connecting Process	Connection IP	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
FileServer.exe	x.y.z.44	48,89%	44,87%	49,54%	47,41%	40,43%
StringServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%
Console58.exe	x.y.z.44	6,13%	5,64%	4,84%	9,83%	7,01%
Console59.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%
FileServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.58	9,15%	9,15%	9,15%	9,15%	9,12%
Console58.exe	x.y.z.58	19,31%	18,85%	18,10%	16,36%	20,23%
Console59.exe	x.y.z.58	9,04%	11,76%	10,10%	9,03%	9,01%
FileServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%
Console58.exe	x.y.z.59	7,48%	9,72%	8,27%	8,23%	14,20%
Console59.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%

Table 8.4.7: Results for TISESVS04, random.

Table 8.4.8 shows the percentage change of usage between the corresponding scenario to the base scenario or scenario 1.

TISESVS04 (58): Percentage changes from Scenario 1								
Connecting Process	Connection IP	Scenario 2	Scenario 3	Scenario 4	Scenario 5	MIN	MAX	AVG
FileServer.exe	x.y.z.44	-4,02%	0,65%	-1,48%	-8,46%	0,65%	8,46%	3,65%
StringServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console58.exe	x.y.z.44	-0,49%	-1,29%	3,70%	0,88%	0,49%	3,70%	1,59%
Console59.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
FileServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.58	0,00%	0,00%	0,00%	-0,03%	0,00%	0,03%	0,01%
Console58.exe	x.y.z.58	-0,46%	-1,21%	-2,95%	0,92%	0,46%	2,95%	1,39%
Console59.exe	x.y.z.58	2,72%	1,06%	-0,01%	-0,03%	0,01%	2,72%	0,96%
FileServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console58.exe	x.y.z.59	2,24%	0,79%	0,75%	6,72%	0,75%	6,72%	2,63%
Console59.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%

Table 8.4.8: Percentage changes for TISESVS04, random.

This time all results are within 5%. In reference to previous tests the average of FileServer.exe on server x.y.z.44 should have been over the 5% limit. This could be caused by a faulted outcome or by the randomization of the timing between files from the clients.

8.4.3 Results for TISESVS06

In the following subsections the two result tables for server TISESVS06 (59) will be presented and discussed. There are two subsections one for standard timing and one for random timing.

Standard timing

Table 8.4.9 shows the percentage usage of server TISESVS06 (59) for each connecting process for all scenarios with standard timing.

Usage of TISESVS06 (59), standard timing						
Connecting Process	Connection IP	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
FileServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%
Console58.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%
Console59.exe	x.y.z.44	4,58%	10,51%	11,51%	4,57%	8,32%
FileServer.exe	x.y.z.58	56,57%	39,89%	40,08%	51,67%	49,87%
StringServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%
Console58.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%
Console59.exe	x.y.z.58	4,63%	10,67%	9,85%	3,78%	8,23%
FileServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.59	9,74%	9,12%	9,10%	9,09%	9,08%
Console58.exe	x.y.z.59	10,25%	9,04%	8,97%	13,87%	8,96%
Console59.exe	x.y.z.59	14,24%	20,77%	20,49%	17,01%	15,54%

Table 8.4.9: Results for TISESVS06, standard.

Table 8.4.10 shows the percentage change of usage between the corresponding scenario to the base scenario or scenario 1.

TISESVS06 (59): Percentage changes from Scenario 1								
Connecting Process	Connection IP	Scenario 2	Scenario 3	Scenario 4	Scenario 5	MIN	MAX	AVG
FileServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console58.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console59.exe	x.y.z.44	5,93%	6,93%	-0,01%	3,74%	0,01%	6,93%	4,15%
FileServer.exe	x.y.z.58	-16,68%	-16,49%	-4,90%	-6,70%	4,90%	16,68%	11,19%
StringServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console58.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console59.exe	x.y.z.58	6,04%	5,22%	-0,85%	3,60%	0,85%	6,04%	3,93%
FileServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.59	-0,62%	-0,64%	-0,65%	-0,66%	0,62%	0,66%	0,64%
Console58.exe	x.y.z.59	-1,21%	-1,28%	3,62%	-1,29%	1,21%	3,62%	1,85%
Console59.exe	x.y.z.59	6,53%	6,25%	2,77%	1,30%	1,30%	6,53%	4,21%

Table 8.4.10: Percentage changes for TISESVS06, standard.

For the standard results of TISESVS06 (59) the server FileServer.exe on x.y.z.58 is the only process which goes over the 5% change limit. This time it is not only caused by scenario 5. Instead it is mostly influenced by the results of scenarios 2 and 3. This could mean that changes on behaviour scheduling are not seriously affecting except for the connecting FileServer.exe process which is connecting to the local StringServer.exe process. This process seems to be the most vulnerable for changes.

Random timing

Table 8.4.11 shows the percentage usage of server TISESVS06 (59) for each connecting process for all scenarios with random timing.

Usage of TISESVS06 (59), random timing						
Connecting Process	Connection IP	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
FileServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%
Console58.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%
Console59.exe	x.y.z.44	10,38%	8,58%	9,22%	11,92%	7,16%
FileServer.exe	x.y.z.58	44,35%	44,97%	51,04%	44,25%	22,89%
StringServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%
Console58.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%
Console59.exe	x.y.z.58	8,67%	9,18%	6,19%	7,65%	6,98%
FileServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.59	9,11%	9,11%	9,11%	10,04%	18,24%
Console58.exe	x.y.z.59	8,99%	8,98%	8,98%	10,86%	17,99%
Console59.exe	x.y.z.59	18,51%	19,17%	15,46%	15,28%	26,74%

Table 8.4.11: Results for TISESVS06, random.

Table 8.4.12 shows the percentage change of usage between the corresponding scenario to the base scenario or scenario 1.

TISESVS06 (59): Percentage changes from Scenario 1								
Connecting Process	Connection IP	Scenario 2	Scenario 3	Scenario 4	Scenario 5	MIN	MAX	AVG
FileServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console58.exe	x.y.z.44	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console59.exe	x.y.z.44	-1,80%	-1,16%	1,54%	-3,22%	1,16%	3,22%	1,93%
FileServer.exe	x.y.z.58	0,62%	6,69%	-0,10%	-21,46%	0,10%	21,46%	7,22%
StringServer.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console58.exe	x.y.z.58	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console59.exe	x.y.z.58	0,51%	-2,48%	-1,02%	-1,69%	0,51%	2,48%	1,43%
FileServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
StringServer.exe	x.y.z.59	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Console44.exe	x.y.z.59	0,00%	0,00%	0,93%	9,13%	0,00%	9,13%	2,52%
Console58.exe	x.y.z.59	-0,01%	-0,01%	1,87%	9,00%	0,01%	9,00%	2,72%
Console59.exe	x.y.z.59	0,66%	-3,05%	-3,23%	8,23%	0,66%	8,23%	3,79%

Table 8.4.12: Percentage changes for TISESVS06, random.

Again the only process exceeding the average of 5% change is the FileServer.exe on server x.y.z.58. This is the process connecting to the local StringServer.exe process. Now scenario 5 has the effect that causes this spike in usage change. It could therefore be concluded that the most vulnerable process to schedule changes is the connecting FileServer.exe process which connects to the local StringServer.exe process. It is vulnerable to all schedule changes but mostly to really short wait time between executions of connection monitor behaviour.

8.4.4 Comparing first to second scenario

To sum up the results of the second scenario the data of the changes in usage percentages for both standard timing and random timing in regards of scenario one will be presented graphically. The second scenario had the time between executions of MemoryBehaviour and ProcessorBehaviour increased.

Standard timing

The percentage of change in usage from scenario 1 to scenario 2 for standard timing is presented in Figure 8.5. The figure shows the change for all servers TISESVS02 (44), TISESVS04 (58) and TISESVS06 (59) along with the average change.

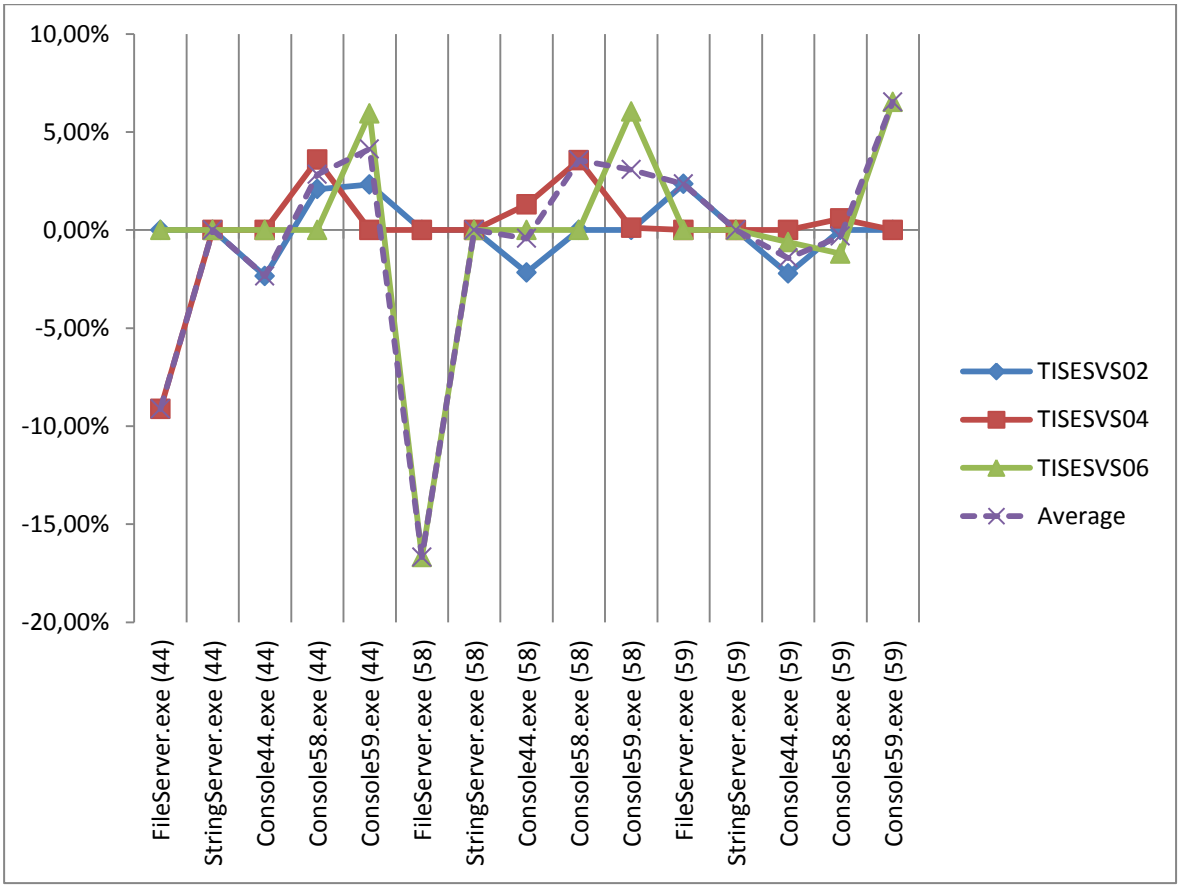


Figure 8.5: Difference in usage for scenario 2 to 1, standard.

The average in Figure 8.5 does not include the zeroes of processes not running on the server. This results shows that five processes go over the 5% change limit when time between executions of memory and processor behaviour is increased. Those are the

Console59.exe processes which are connecting to FileServer.exe on server TISESVS06 (59). This could be a consequence of strain on the server since this does not happen to other clients. The other connecting processes are FileServer.exe which is running on server TISESVS02 (44) and FileServer.exe running on server TISESVS04 (58). These two processes go well over the 5% limit and as can be seen FileServer.exe (58) has the largest change with 16,68%. This support the hypothesis that the FileServer.exe process is must vulnerable for behaviour changes. This could be an indication that changes in behaviour scheduling could affect the outcome for this type of processes.

Random timing

The percentage of change in usage from scenario 1 to scenario 2 for random timing will be represented in Figure 8.6. The figure shows the change for all servers TISESVS02 (44), TISESVS04 (58) and TISESVS06 (59) along with the average change.

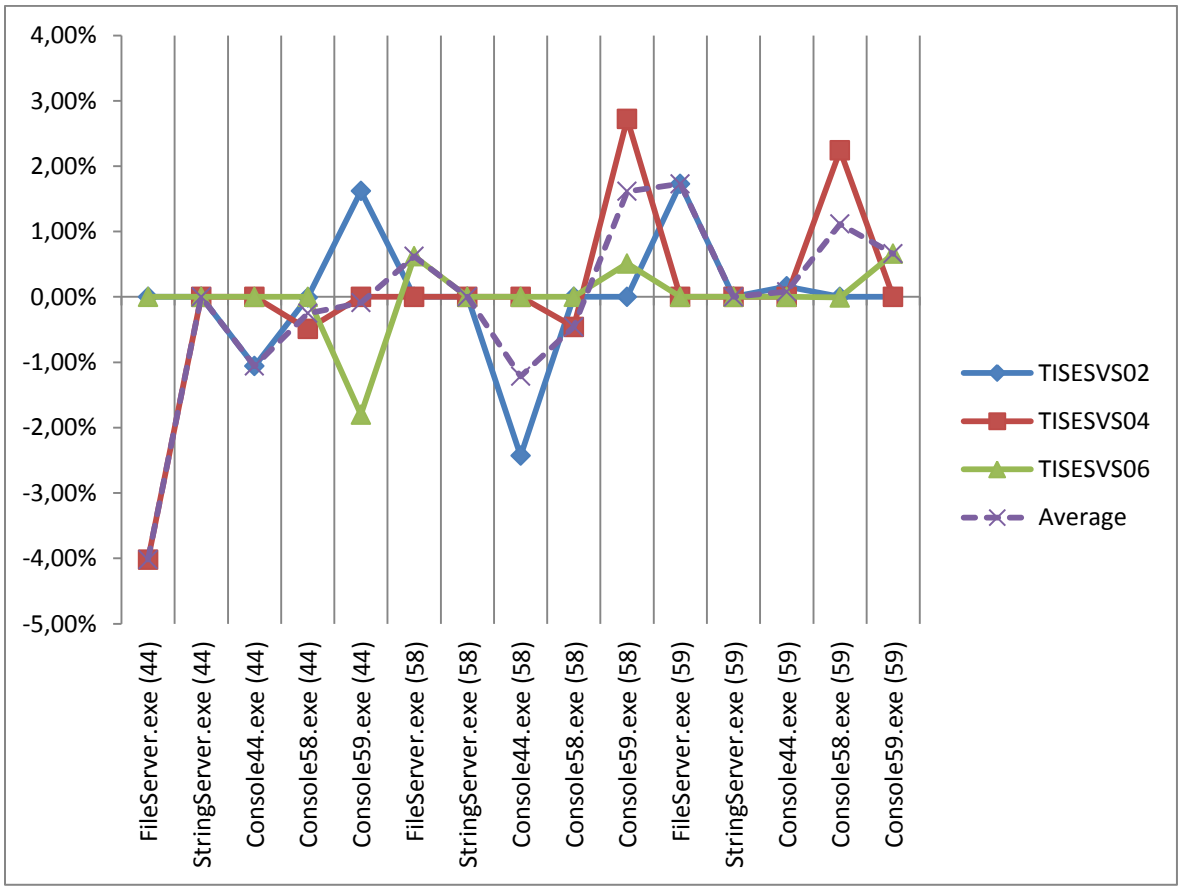


Figure 8.6: Difference in usage for scenario 2 to 1, random.

The average in Figure 8.6 does not include the zeroes of processes not running on the server. For random timing no process goes over the 5% limit. This could be an indication that the behaviours are not as vulnerable for requests not coming with predefined intervals.

8.4.5 Comparing first to third scenario

To sum up the results of the third scenario the data of the changes in usage percentages for both standard timing and random timing in regards of scenario one will be presented graphically. The third scenario had the time between executions of MemoryBehaviour and ProcessorBehaviour decreased.

Standard timing

The percentage of change in usage from scenario 1 to scenario 3 for standard timing will be represented in Figure 8.7. The figure shows the change for all servers TISESVS02 (44), TISESVS04 (58) and TISESVS06 (59) along with the average change.

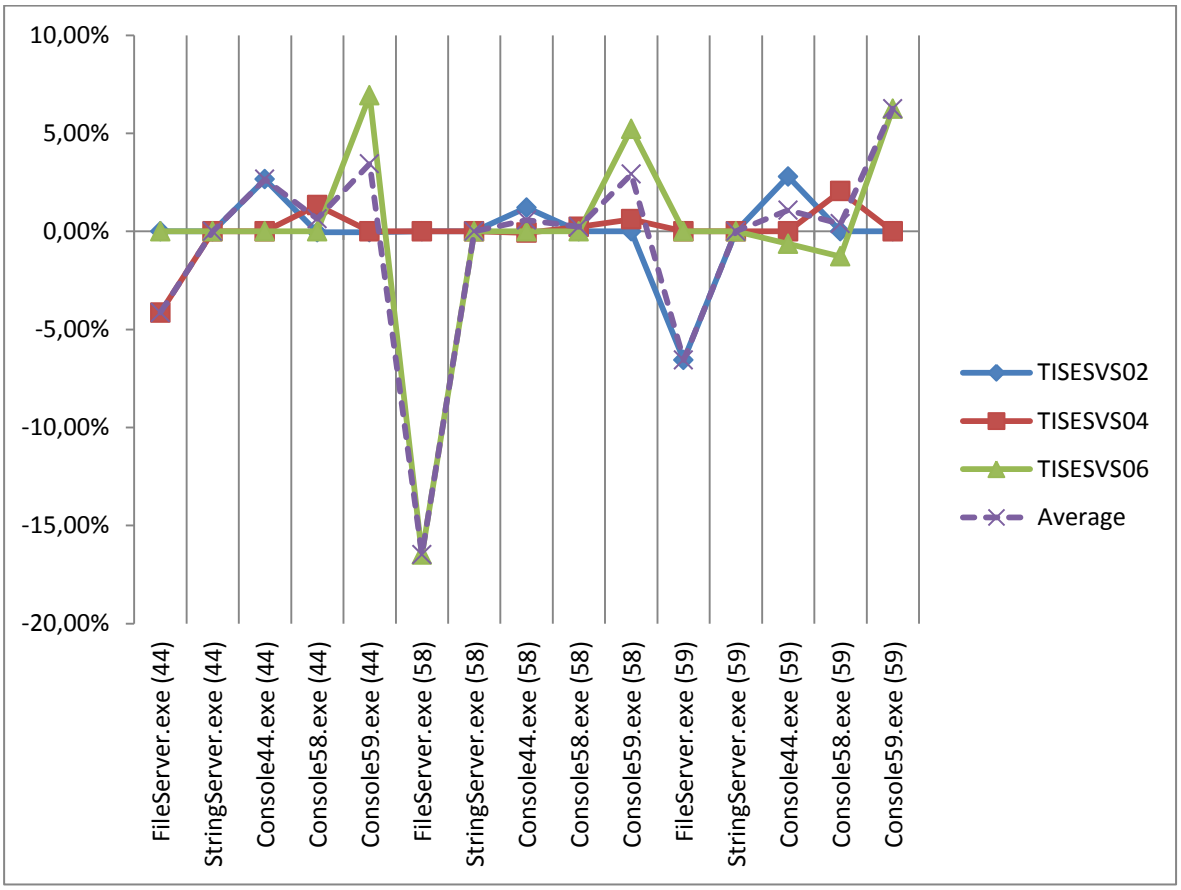


Figure 8.7: Difference in usage for scenario 3 to 1, standard.

The average in Figure 8.7 does not include the zeroes of processes not running on the server. For standard timing in scenario 3, five processes are again over the limit. These are again the three Console59.exe processes. This could mean that changes in memory and processor behaviour are affecting these clients since in this scenario the time between executions was shortened. This does though not affect the other clients which could indicate strain on server TISESVS06 (59). The main thing is that still are the FileServer.exe processes going over the 5% limit in this case the processes from TISESVS04 (58) and TISESVS06 (59).

Random timing

The percentage of change in usage from scenario 1 to scenario 3 for random timing will be represented in Figure 8.8. The figure shows the change for all servers TISESVS02 (44), TISESVS04 (58) and TISESVS06 (59) along with the average change.

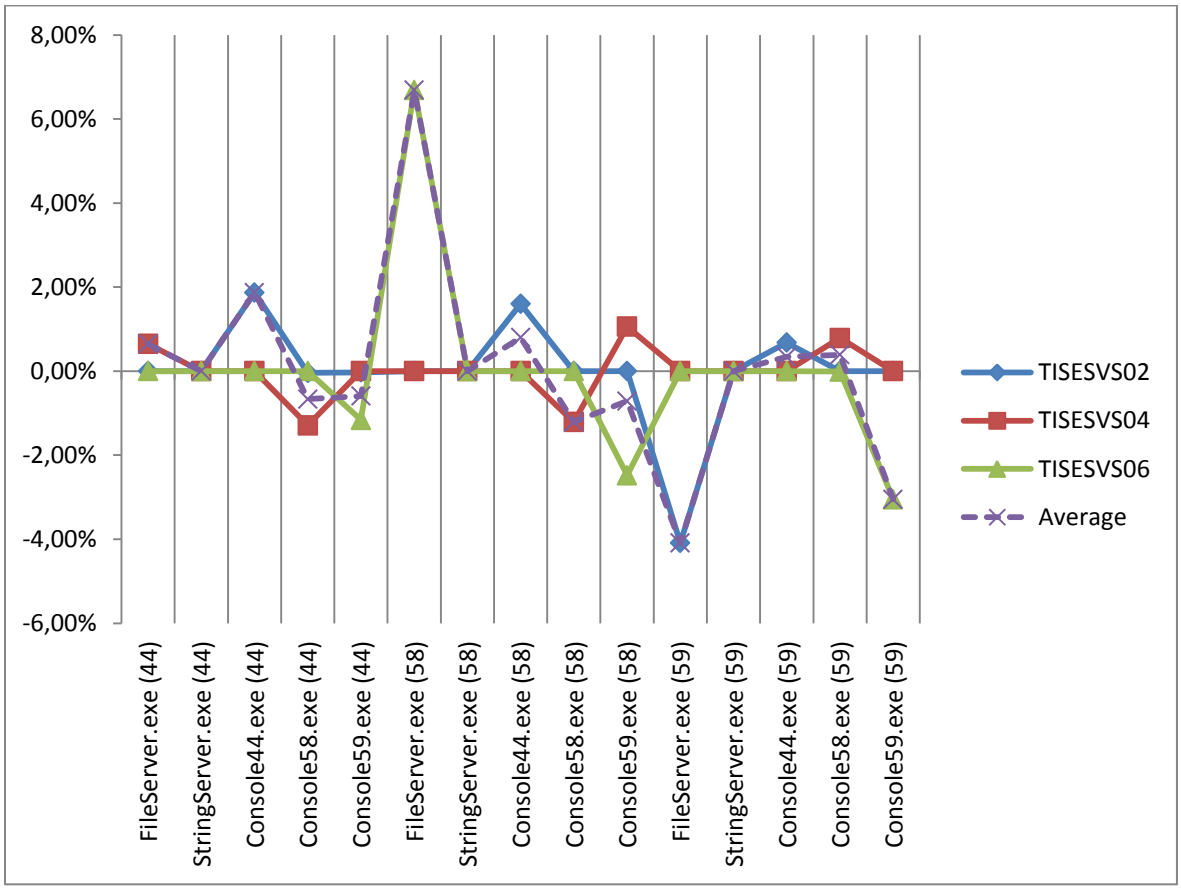


Figure 8.8: Difference in usage for scenario 3 to 1, random.

The average in Figure 8.8 does not include the zeroes of processes not running on the server. Again fever process are going over the 5% limit when the scenario is executed with random timing. Still there is one process going over which is FileServer.exe on server TISESVS04 (58).

8.4.6 Comparing first to fourth scenario

To sum up the results of the fourth scenario the data of the changes in usage percentages for both standard timing and random timing in regards of scenario one will be presented graphically. The fourth scenario had the time between executions of ConnectionsMonitorBehaviour increased.

Standard timing

The percentage of change in usage from scenario 1 to scenario 4 for standard timing will be represented in Figure 8.9. The figure shows the change for all servers TISESVS02 (44), TISESVS04 (58) and TISESVS06 (59) along with the average change.

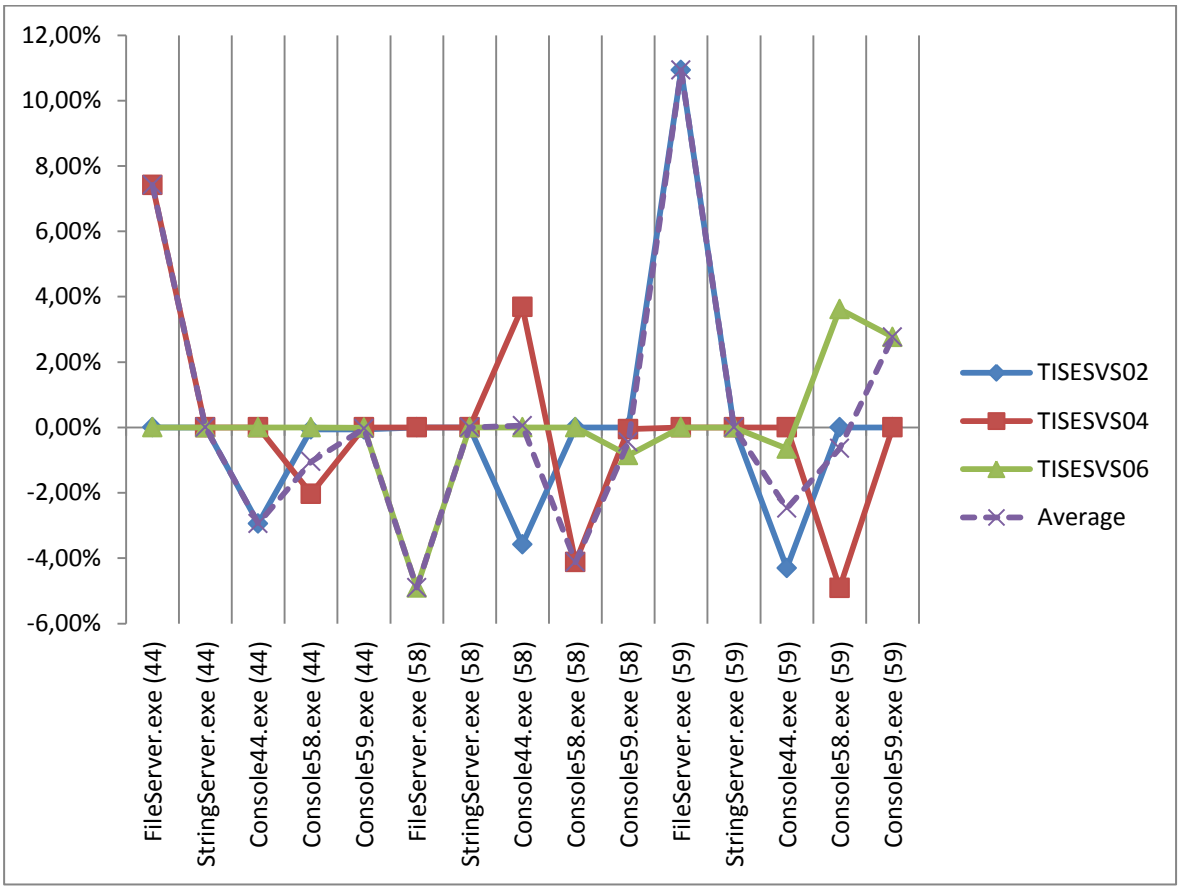


Figure 8.9: Difference in usage for scenario 4 to 1, standard.

The average in Figure 8.9 does not include the zeroes of processes not running on the server. For scenario 4 which is increase in time between execution of connection monitor behaviour two processes go over the 5% limit. These processes are FileServer.exe on TISESVS02 (44) and FileServer.exe on TISESVS06 (59). These are the processes connecting to the StringServer.exe process on remote server.

Random timing

The percentage of change in usage from scenario 1 to scenario 4 for random timing will be represented in Figure 8.10. The figure shows the change for all servers TISESVS02 (44), TISESVS04 (58) and TISESVS06 (59) along with the average change.

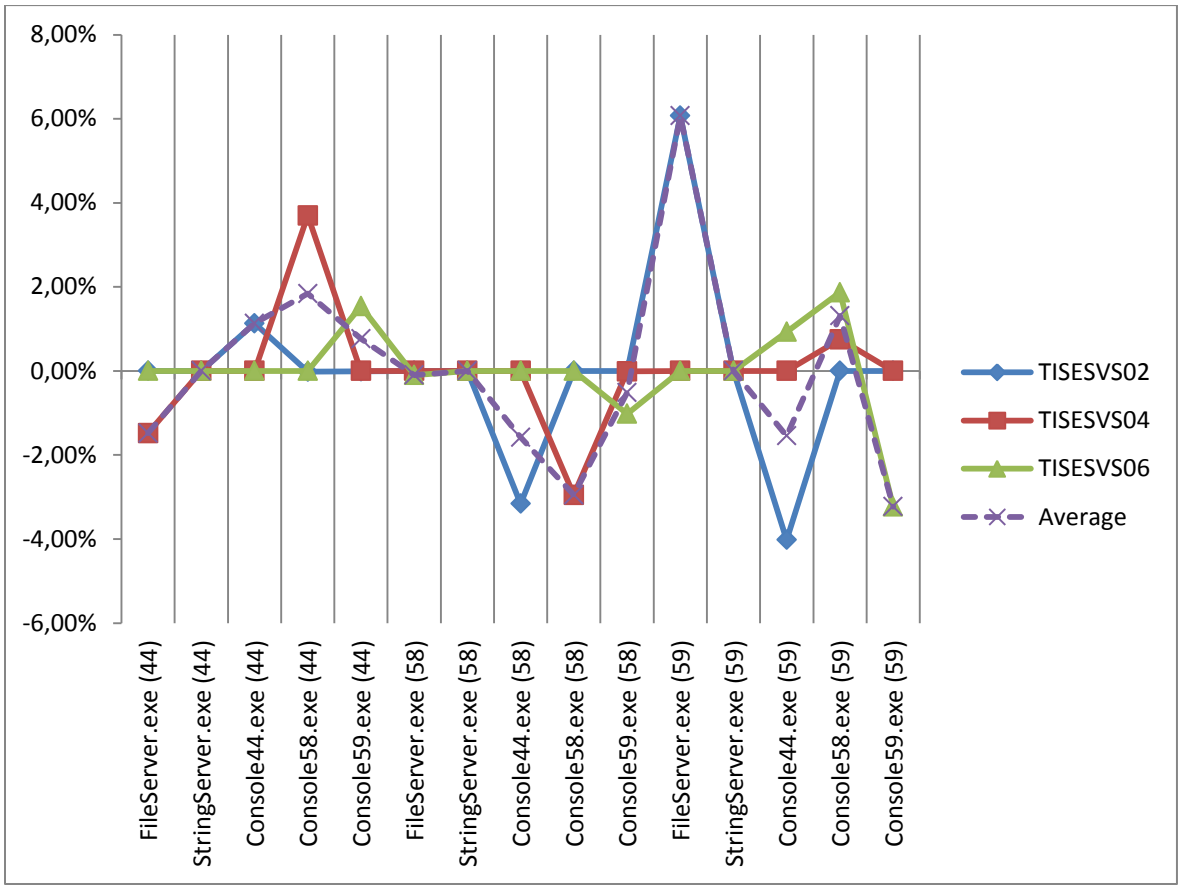


Figure 8.10: Difference in usage for scenario 4 to 1, random.

The average in Figure 8.10 does not include the zeroes of processes not running on the server. Again only one process goes over the 5% limit when files are created with random time between files. This process is the FileServer.exe on server TISESVS06 (59) which is connecting to StringServer.exe on TISESVS02 (44). This supports the hypothesis that processes connecting to multi task remote processes are most vulnerable to schedule chances.

8.4.7 Comparing first to fifth scenario

To sum up the results of the fifth scenario the data of the changes in usage percentages for both standard timing and random timing in regards of scenario one will be presented graphically. The fifth scenario had the time between executions of ConnectionsMonitorBehaviour decreased.

Standard timing

The percentage of change in usage from scenario 1 to scenario 5 for standard timing will be represented in Figure 8.11. The figure shows the change for all servers TISESVS02 (44), TISESVS04 (58) and TISESVS06 (59) along with the average change.

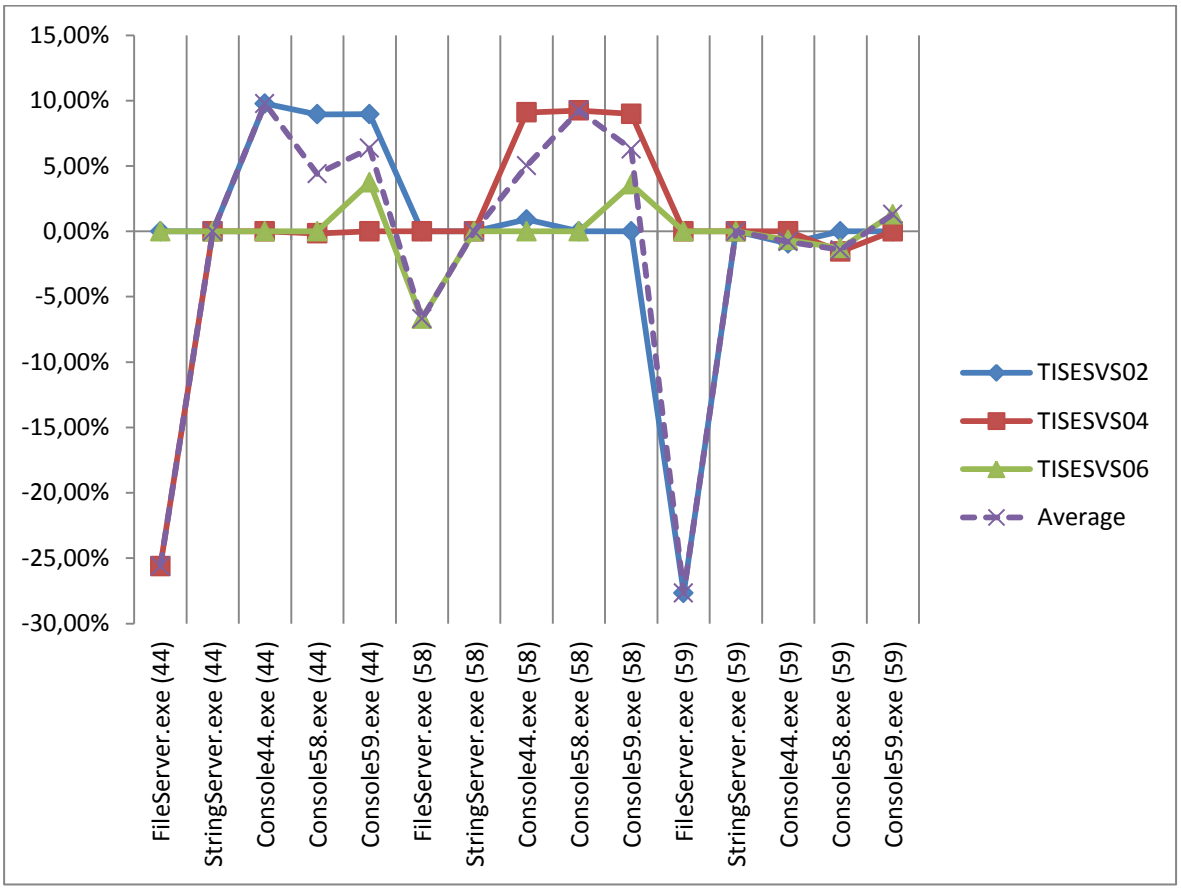


Figure 8.11: Difference in usage for scenario 5 to 1, standard.

The average in Figure 8.11 does not include the zeroes of processes not running on the server. When time between execution of connection monitoring behaviour is decreased to 500 ms almost all processes go over the 5% limit or nine processes. That shows that the agents are most vulnerable to decreased time between executions of the connection monitoring behaviour when standard wait time is between executions of the connecting processes.

Random timing

The percentage of change in usage from scenario 1 to scenario 5 for random timing will be represented in Figure 8.12. The figure shows the change for all servers TISESVS02 (44), TISESVS04 (58) and TISESVS06 (59) along with the average change.

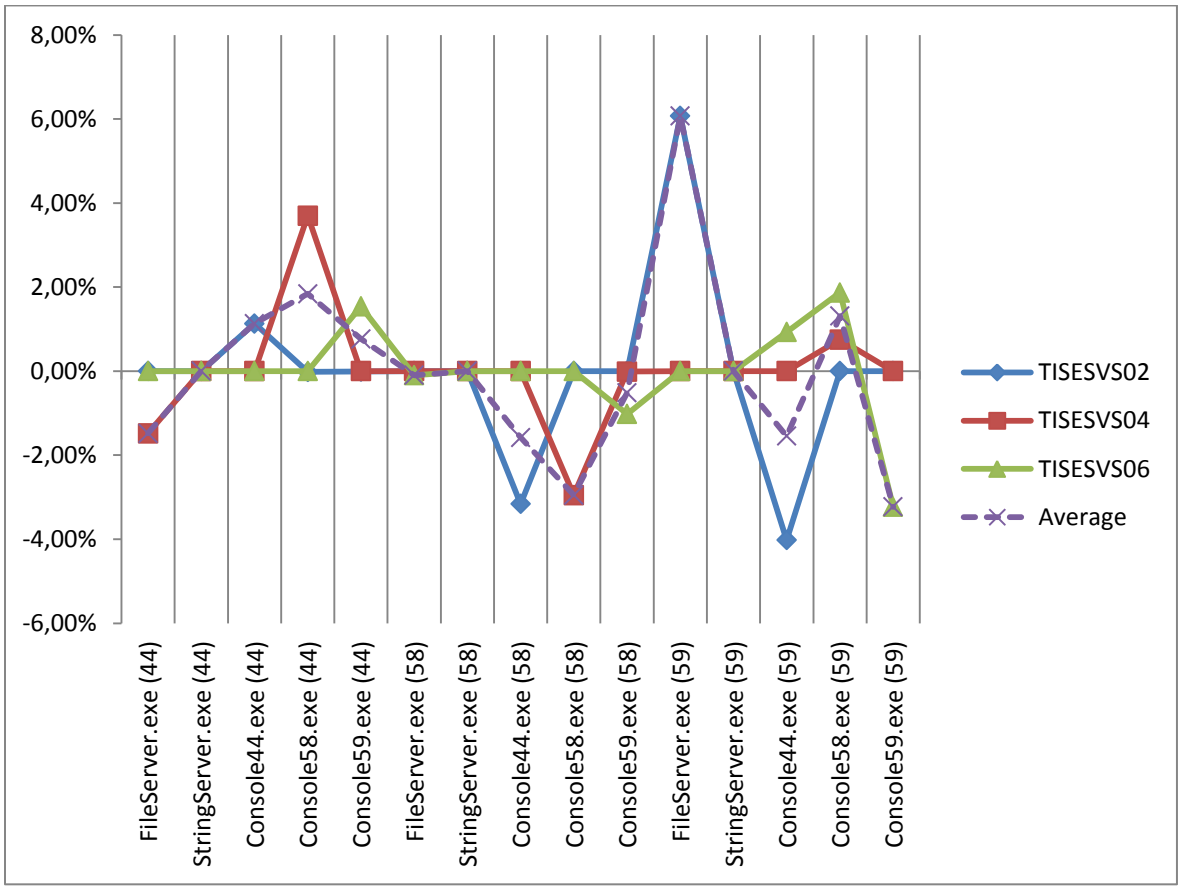


Figure 8.12: Difference in usage for scenario 5 to 1, random.

The average in Figure 8.12 does not include the zeroes of processes not running on the server. The same effect as for standard timing cannot be seen for random timing. This time only one process goes over the 5% limit and that is once again the FileServer.exe process on TISESVS06 (59).

8.4.8 Comparison to the old cost model

To compare the new cost model to the old cost model (as described in section 1.1) the results of scenario 1 in Table 8.4.1 will be configured to show the results if the usage had been calculated with the old cost model compared to the new cost model. These results can be seen in Table 8.4.13.

Usage of TISESVS02 (44), standard timing			
Connecting Process	Connection IP	New model	Old model
FileServer.exe	x.y.z.44	0,00%	0,00%
StringServer.exe	x.y.z.44	0,00%	0,00%
Console44.exe	x.y.z.44	16,62%	16,67%
Console58.exe	x.y.z.44	9,07%	16,67%
Console59.exe	x.y.z.44	9,09%	16,67%
FileServer.exe	x.y.z.58	0,00%	0,00%
StringServer.exe	x.y.z.58	0,00%	0,00%
Console44.exe	x.y.z.58	7,33%	16,67%
Console58.exe	x.y.z.58	0,00%	0,00%
Console59.exe	x.y.z.58	0,00%	0,00%
FileServer.exe	x.y.z.59	50,47%	16,67%
StringServer.exe	x.y.z.59	0,00%	0,00%
Console44.exe	x.y.z.59	7,42%	16,67%
Console58.exe	x.y.z.59	0,00%	0,00%
Console59.exe	x.y.z.59	0,00%	0,00%

Table 8.4.13: New cost model versus old model.

As can be seen the new cost model is much more accurate than the old one. It cannot be said that the new cost model is correct but it can be stated that it divides usage more fairly between connected processes. By implementing the new cost model costs for server usage could be collected with more accuracy.

8.5 Evaluation conclusions

When all the results from section 8.4 are taken into an account it can be concluded that changes in behaviour scheduling does affect the outcome of the monitored data by the agents. The most vulnerable processes to monitor are processes that connect to local processes that perform multi task assignments that demand lot of work by the server the process connects to. It can also be concluded that the affect is bigger when the same time is always used between requests of executions because of the sampling methods used (running behaviour with predefined intervals). The agents are most vulnerable to changes in scheduling of the connection monitoring behaviour though effects of changes on memory and processor scheduling can be measured.

So the conclusion of this evaluation is that changes in behaviour scheduling can affect the sampling data and by that affect the results of the cost model. It can though be concluded that the cost model created is presenting more precise divisions then the old cost model used, see section 8.4.8.

9 Conclusion

The goal we wanted to achieve with this project was to create agents that could monitor memory and processor usage for each process and associate that usage to incoming connections. By doing this we wanted to be able to create a cost model that could be used by the user to divide the calculated cost of a server in use. The most important results of the thesis are in summary:

- A Python agent framework was created to be used for the creation of agents running on a distributed system.
- Agent behaviours were created for the agents so they could fulfill their tasks at hand.
- A cost model was created to show the division of cost between connecting systems to the monitored server.
- The created cost model was presented with an SSRS report which is customizable by the user.
- An evaluation was done on the created cost model and the effect of change of time scheduling of behaviours on the cost model.
- Finally was the new cost model compared to the old cost model to show if any gain had been achieved by the creation of the cost model.

It has been shown in this thesis that it is possible to monitor server usage by creating distributed agents that operate on each server which should be monitored. The data these agents collect by sampling the server usage can show a better picture of the whole environment. The scheduling of behaviours can though affect the outcome and to perfect these methods more research on behaviour scheduling should be done. To improve the methods used in this thesis more research on connection monitoring should be done to get a better vision of the connecting system and by doing that increase the quality of the cost model. It can though be concluded that the created cost model is much more efficient than the model discussed in the beginning of this thesis used to divide costs of servers. As was shown in section 8.4.8 the new cost model is an improvement over the old one and it provides a more accurate result for the division of cost per server.

The total size of the project implementation created in this thesis was 4000 lines of Python code and 500 lines of SQL code. Only a proof of concept was created in this thesis to show that the methods discussed in this thesis could be used to create a more accurate cost model for server environments. As has been shown in this thesis it is possible. With more time and resources to investigate this matter in more detail a good cost model could be created with methods discussed in this thesis. The cost model could for example be improved by taking into account how much load on the processor each individual request creates. Currently a short request is charged just like a long request. The improved cost model should provide a good tool for users interesting in getting the right cost division of their server environment.

This thesis focused on the creation of the cost model, agent behaviours and evaluation of the created cost model. To get more information about agent lifecycles, communication between agents and data accuracy please study the other thesis on the topic done by Einar Sveinsson [1].

References

[1] E. Sveinsson, "Design decision alternatives for agent-based monitoring of distributed server environments," Faculty of Industrial Engineering, Mechanical Engineering and Computer Science, University of Iceland, 2012.

[2] J. M. Bradshaw, An Introduction to Software Agents, Menlo Park, CA: American Association for Artificial Intellegence, 1997.

[3] G. Weiss, Multiagent Systems, London: The MIT Press, 1999.

[4] J. Dale and E. Mamdani, "FIPA.org," [Online]. Available: http://www.fipa.org/docs/input/f-in-00023/f-in-00023.html. [Accessed April 2012].

[5] F. Bellifemine, A. Poggi and G. Rimassa, "JADE – A FIPA-compliant agent framework," 1999. [Online]. Available: http://www.dia.fi.upm.es/~phernan/AgentesInteligentes/referencias/bellifemine99.pdf. [Accessed July 2012].

[6] "FIPA Agent Management Specification," 18 March 2004. [Online]. Available: http://www.fipa.org/specs/fipa00023/SC00023K.pdf. [Accessed August 2012].

[7] A. Beaulieu, Learning SQL, Second Edition, Sebastopol, CA: O'Reilly Media, Inc., 2009.

[8] R. Rankins, P. Bertucci, C. Gallelli og A. T. Silverstein, Microsoft SQL Server 2008 R2 Unleased, USA: Pearson Education, Inc., 2011.

[9] K. Haselden, Microsoft SQL Server 2008 Integration Services Unleashed, USA: Pearson Education, Inc., 2009.

[10] M. Lisin, J. Joseph and A. Goyal, Microsft SQL Server 2008 Reporting Services Unleashed, USA: Sams Publishing, 2010.

[11] S. Bowden, "SQLServerCentral.com," Simple Talk Publishing, 29 November 2011. [Online]. Available: http://www.sqlservercentral.com/articles/Management+Data+Warehouse/71491/. [Accessed July 2012].

[12] M. Lutz, Programming Python, Sebastopel, CA: O'Reilly Media, Inc., 2006.

[13] "PSUTIL," [Online]. Available: http://code.google.com/p/psutil/. [Accessed April 2012].

[14] "Microsoft support," Microsoft, July 2010. [Online]. Available: http://support.microsoft.com/kb/110093. [Accessed April 2012].

[15] "PYODBC," [Online]. Available: http://code.google.com/p/pyodbc/. [Accessed April 2012].

[16] A. Fetting, Twisted Network Programming Essentials, Sebastopel, CA: O'Reilly Media, Inc., 2006.

[17] I. Shtull-Trauring, "An Introduction to the Twisted Networking Framework," O'Reilly, 15 Januart 2004. [Online]. Available: http://onlamp.com/pub/a/python/2004/01/15/twisted_intro.html. [Accessed August 2012].

[18] "Twisted Matrix Labs," [Online]. Available: http://twistedmatrix.com/trac/wiki/Documentation. [Accessed August 2012].

[19] K. Meyler, C. Fuller, J. Joyner and A. Dominey, System Center Operation Manager 2007 R2 Unleashed, USA: Pearson Education, Inc., 2010.

[20] "Microsoft System Center," 10 September 2012. [Online]. Available: http://technet.microsoft.com/en-us/library/hh205987.aspx. [Accessed September 2012].

[21] "Spade2," [Online]. Available: http://code.google.com/p/spade2/. [Accessed June 2012].

[22] "Tutorial 1: JADE Architecture Overview," [Online]. Available: http://jade.tilab.com/doc/tutorials/JADEAdmin/jadeArchitecture.html. [Accessed September 2012].

Appendix A

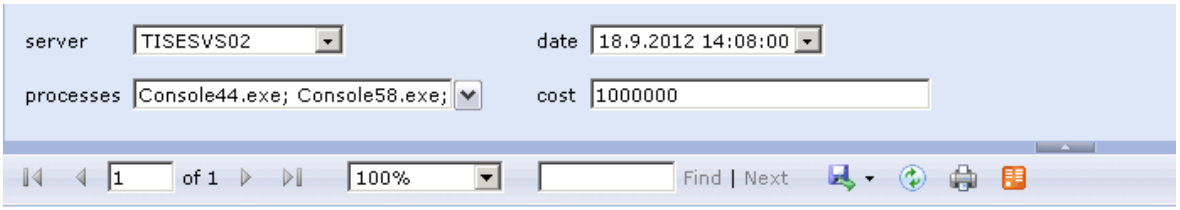
This appendix provides the results of each test scenario and their explanation. As said earlier each scenario consists of two runs, one with standard timing and one with random timing.

First scenario

The first scenario had the pre-defined base schedules.

Standard timing

After execution of the planned scenario the cost model for each server was executed. See Figure A.1 for the corresponding cost model report of TISESVS02 (44), Figure A.2 for cost model report of TISESVS04 (58) and Figure A.3 for cost model report of TISESVS06 (59).



Server: TISESVS02

Date: 18. September 2012 - 14:08:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS02	x.y.z.44	Console44.exe	16.62%	166,158 kr.
TISESVS02	x.y.z.44	Console58.exe	9.07%	90,719 kr.
TISESVS02	x.y.z.44	Console59.exe	9.09%	90,891 kr.
TISESVS02	x.y.z.58	Console44.exe	7.33%	73,329 kr.
TISESVS02	x.y.z.59	Console44.exe	7.42%	74,171 kr.
TISESVS02	x.y.z.59	FileServer.exe	50.47%	504,732 kr.

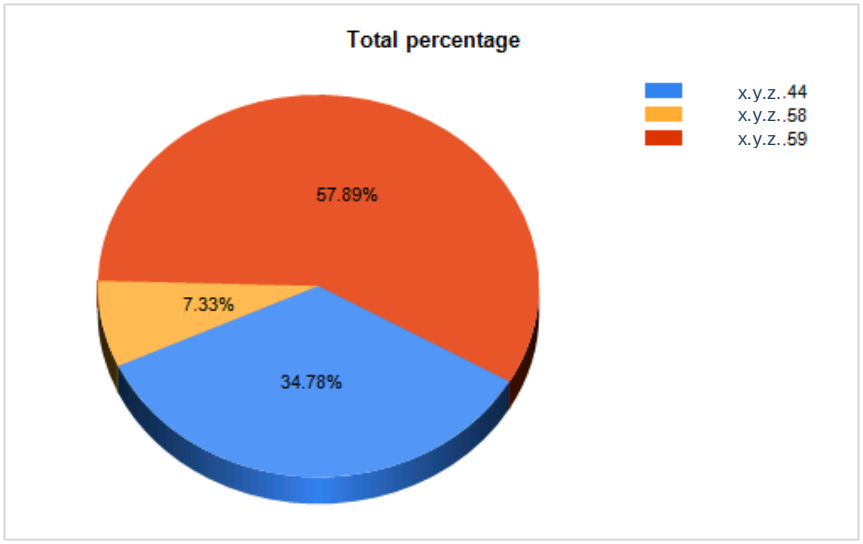
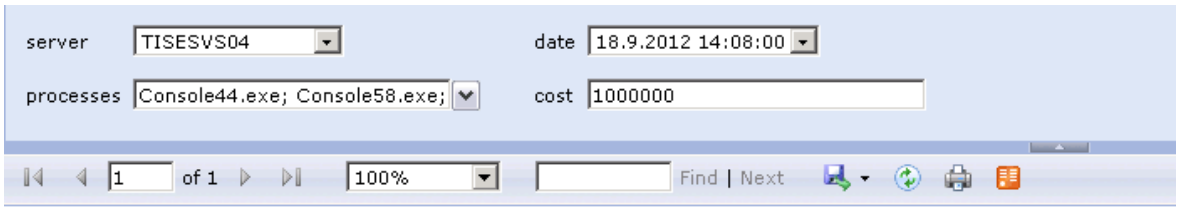


Figure A.1: Scenario one for TISESVS02, standard.



Server: TISESVS04

Date: 18. September 2012 - 14:08:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS04	x.y.z. 44	Console58.exe	6.04%	60,420 kr.
TISESVS04	x.y.z. 44	FileServer.exe	48.46%	484,592 kr.
TISESVS04	x.y.z. 58	Console44.exe	9.19%	91,869 kr.
TISESVS04	x.y.z. 58	Console58.exe	18.00%	179,971 kr.
TISESVS04	x.y.z. 58	Console59.exe	9.07%	90,709 kr.
TISESVS04	x.y.z. 59	Console58.exe	9.24%	92,438 kr.

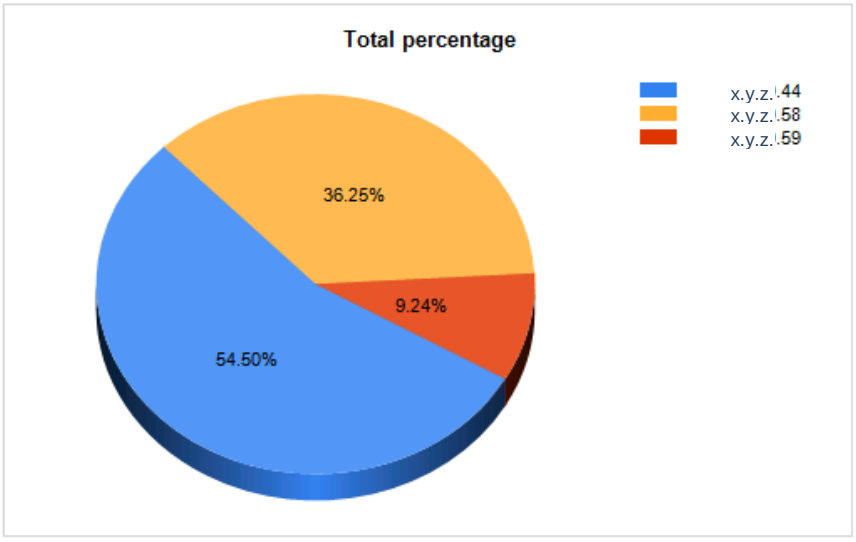


Figure A.2: Scenario one for TISESVS04, standard.

Server: TISESVS06

Date: 18. September 2012 - 14:08:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS06	x.y.z..44	Console59.exe	4.58%	45,759 kr.
TISESVS06	x.y.z..58	Console59.exe	4.63%	46,282 kr.
TISESVS06	x.y.z..58	FileServer.exe	56.57%	565,653 kr.
TISESVS06	x.y.z..59	Console44.exe	9.74%	97,369 kr.
TISESVS06	x.y.z..59	Console58.exe	10.25%	102,494 kr.
TISESVS06	x.y.z..59	Console59.exe	14.24%	142,443 kr.

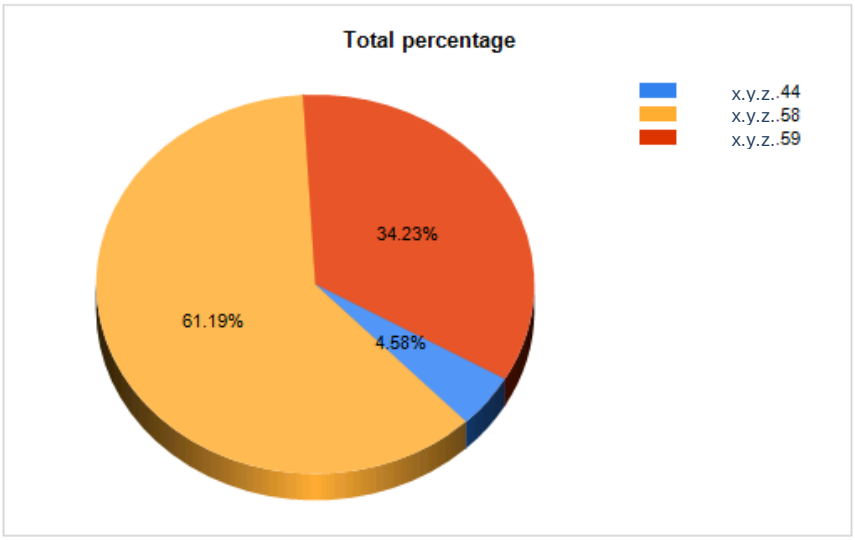
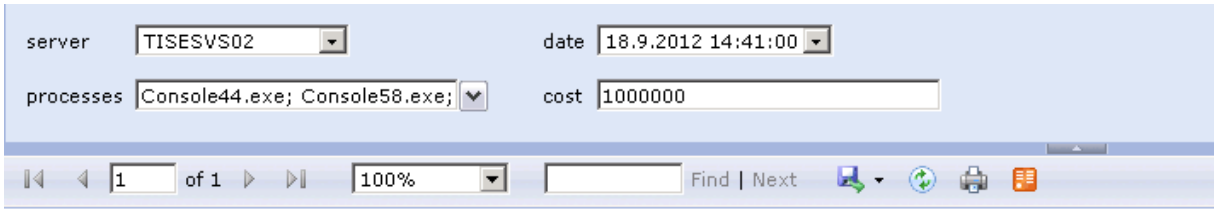


Figure A.3: Scenario one for TISESVS06, standard.

Random timing

After execution of the planned scenario the cost model for each server was executed. See Figure A.4 for the corresponding cost model report of TISESVS02 (44), Figure A.5 for cost model report of TISESVS04 (58) and Figure A.6 for cost model report of TISESVS06 (59).



Server: TISESVS02

Date: 18. September 2012 - 14:41:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS02	x.y.z..44	Console44.exe	18.02%	180,225 kr.
TISESVS02	x.y.z..44	Console58.exe	9.05%	90,470 kr.
TISESVS02	x.y.z..44	Console59.exe	9.06%	90,645 kr.
TISESVS02	x.y.z..58	Console44.exe	8.84%	88,375 kr.
TISESVS02	x.y.z..59	Console44.exe	8.60%	85,965 kr.
TISESVS02	x.y.z..59	FileServer.exe	46.43%	464,320 kr.

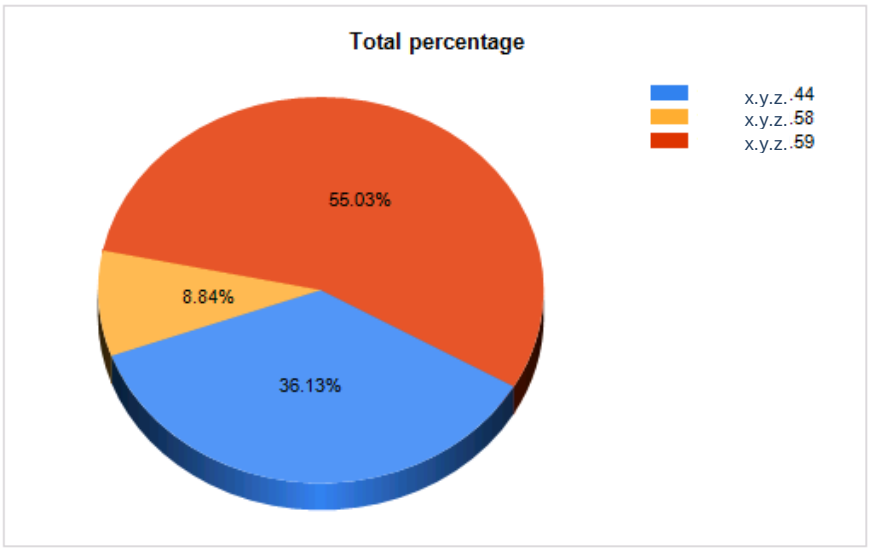


Figure A.4: Scenario one for TISESVS02, random.

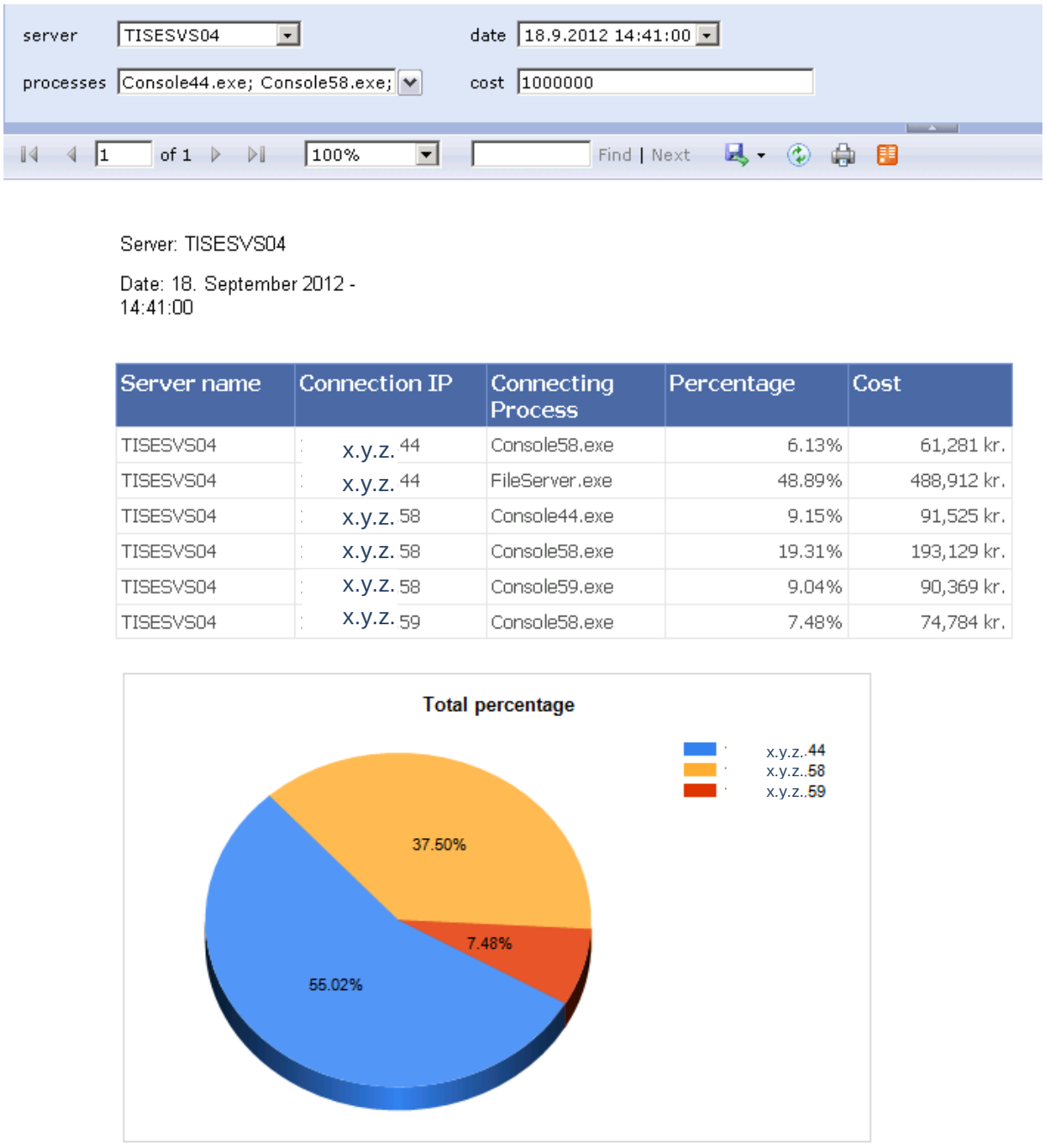
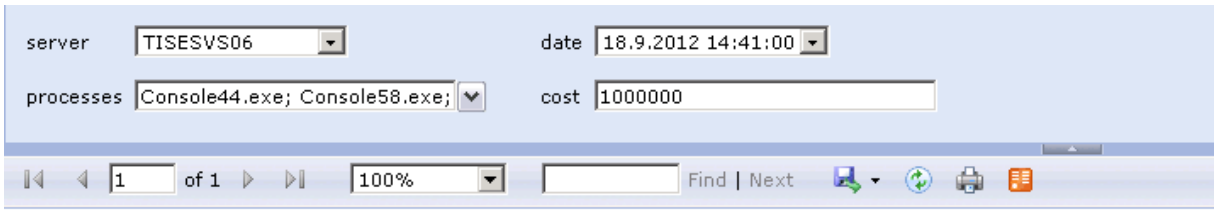


Figure A.5: Scenario one for TISESVS04, random.



Server: TISESVS06

Date: 18. September 2012 - 14:41:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS06	x.y.z.44	Console59.exe	10.37%	103,712 kr.
TISESVS06	x.y.z.58	Console59.exe	8.67%	86,741 kr.
TISESVS06	x.y.z.58	FileServer.exe	44.35%	443,480 kr.
TISESVS06	x.y.z.59	Console44.exe	9.11%	91,091 kr.
TISESVS06	x.y.z.59	Console58.exe	8.99%	89,862 kr.
TISESVS06	x.y.z.59	Console59.exe	18.51%	185,114 kr.

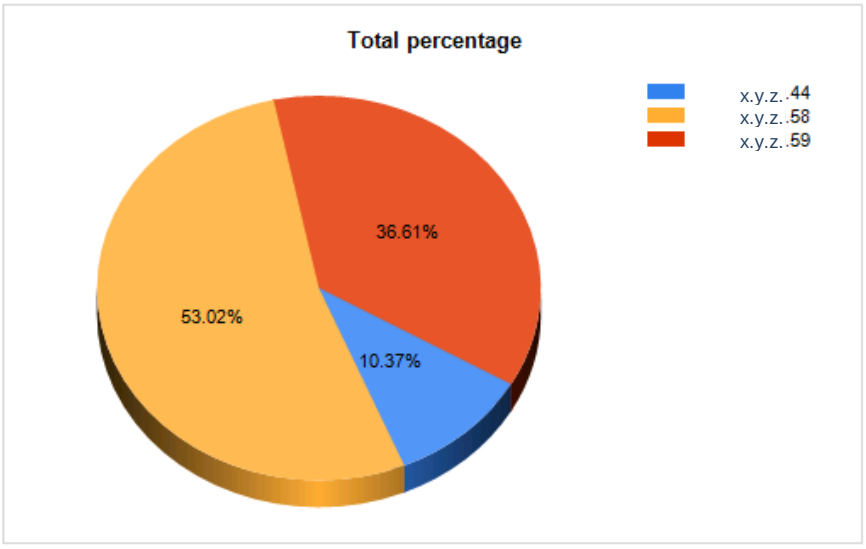


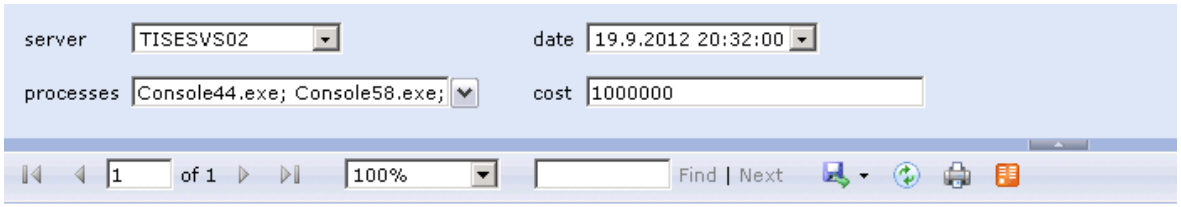
Figure A.6: Scenario one for TISESVS06, random.

Second scenario

The second scenario had the time between memory and processor sampling behaviours increased.

Standard timing

After execution of the planned scenario the cost model for each server was executed. See Figure A.7 for the corresponding cost model report of TISESVS02 (44), Figure A.8 for cost model report of TISESVS04 (58) and Figure A.9 for cost model report of TISESVS06 (59).



Server: TISESVS02

Date: 19. September 2012 - 20:32:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS02	x.y.z.44	Console44.exe	14.27%	142,701 kr.
TISESVS02	x.y.z.44	Console58.exe	11.15%	111,524 kr.
TISESVS02	x.y.z.44	Console59.exe	11.41%	114,105 kr.
TISESVS02	x.y.z.58	Console44.exe	5.16%	51,627 kr.
TISESVS02	x.y.z.59	Console44.exe	5.19%	51,895 kr.
TISESVS02	x.y.z.59	FileServer.exe	52.81%	528,148 kr.

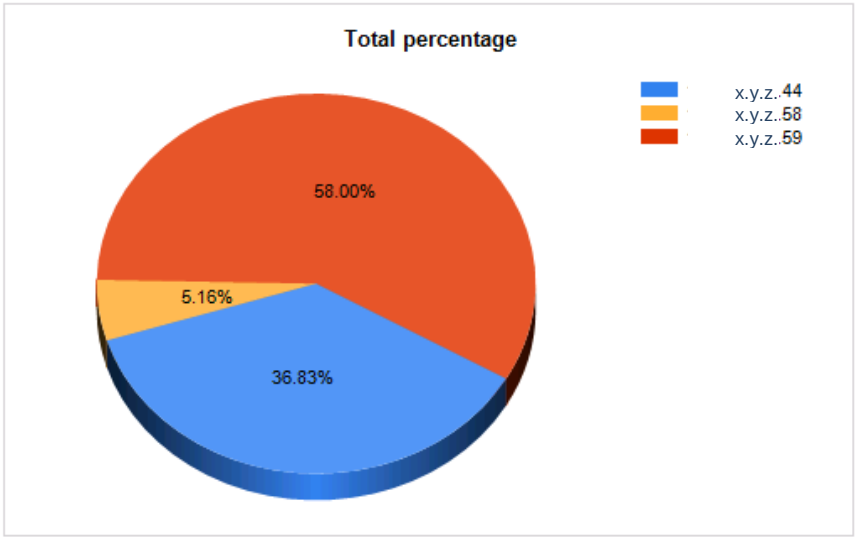
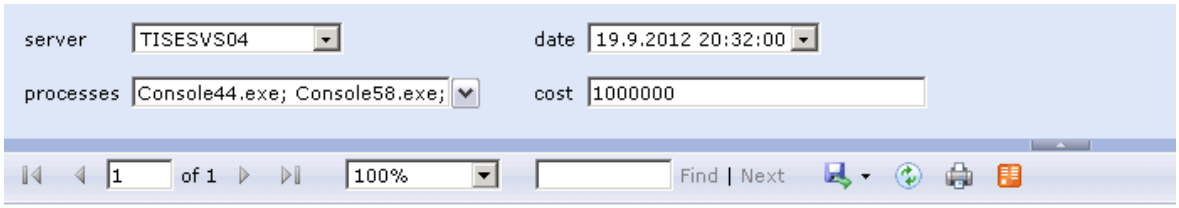


Figure A.7: Scenario two for TISESVS02, standard.



Server: TISESVS04

Date: 19. September 2012 - 20:32:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS04	x.y.z.44	Console58.exe	9.62%	96,178 kr.
TISESVS04	x.y.z.44	FileServer.exe	39.33%	393,325 kr.
TISESVS04	x.y.z.58	Console44.exe	10.49%	104,909 kr.
TISESVS04	x.y.z.58	Console58.exe	21.56%	215,637 kr.
TISESVS04	x.y.z.58	Console59.exe	9.18%	91,779 kr.
TISESVS04	x.y.z.59	Console58.exe	9.82%	98,172 kr.

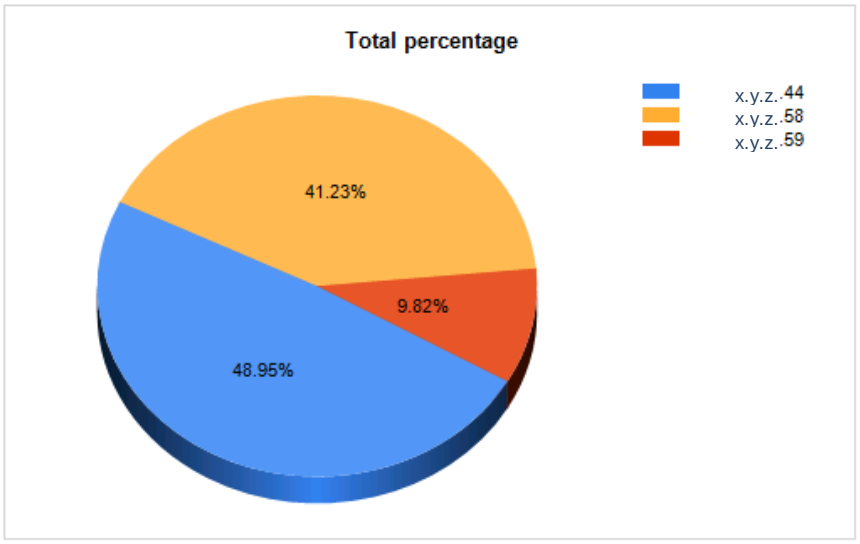
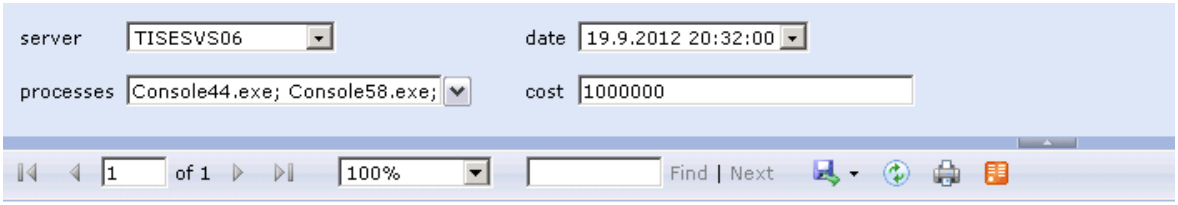


Figure A.8: Scenario two for TISESVS04, standard.



Server: TISESVS06

Date: 19. September 2012 - 20:32:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS06	x.y.z.44	Console59.exe	10.51%	105,089 kr.
TISESVS06	x.y.z.58	Console59.exe	10.67%	106,722 kr.
TISESVS06	x.y.z.58	FileServer.exe	39.89%	398,890 kr.
TISESVS06	x.y.z.59	Console44.exe	9.12%	91,232 kr.
TISESVS06	x.y.z.59	Console58.exe	9.04%	90,369 kr.
TISESVS06	x.y.z.59	Console59.exe	20.77%	207,697 kr.

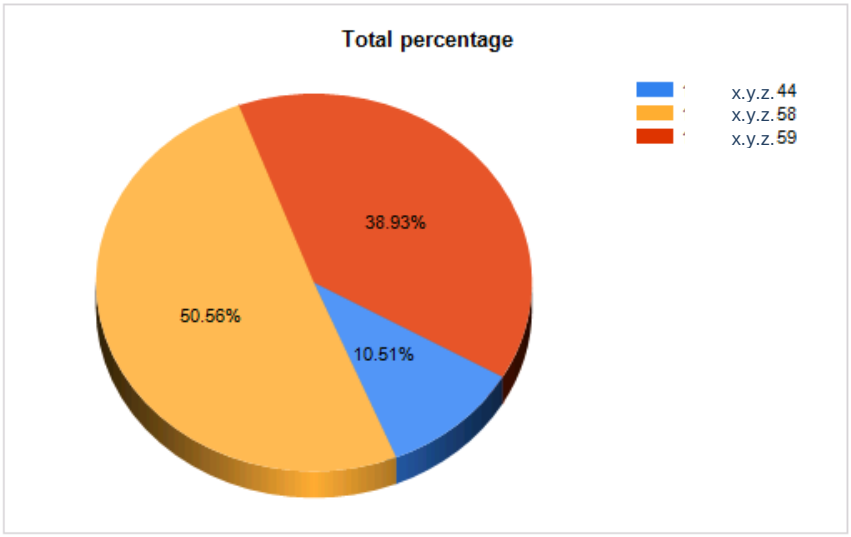
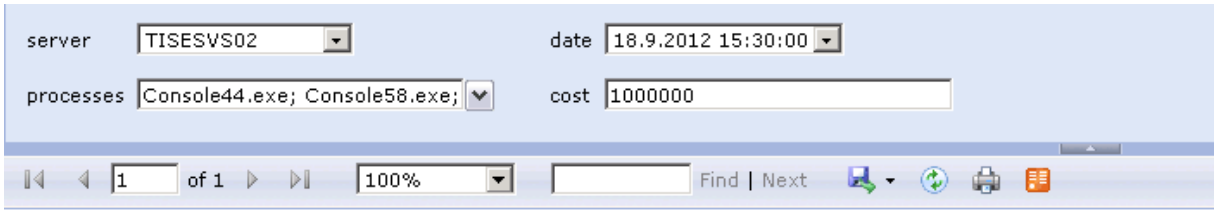


Figure A.9: Scenario two for TISESVS06, standard.

Random timing

After execution of the planned scenario the cost model for each server was executed. See Figure A.10 for the corresponding cost model report of TISESVS02 (44), Figure A.11 for cost model report of TISESVS04 (58) and Figure A.12 for cost model report of TISESVS06 (59).



Server: TISESVS02

Date: 18. September 2012 - 15:30:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS02	x.y.z..44	Console44.exe	16.96%	169,564 kr.
TISESVS02	x.y.z..44	Console58.exe	9.04%	90,356 kr.
TISESVS02	x.y.z..44	Console59.exe	10.68%	106,765 kr.
TISESVS02	x.y.z..58	Console44.exe	6.41%	64,056 kr.
TISESVS02	x.y.z..59	Console44.exe	8.76%	87,617 kr.
TISESVS02	x.y.z..59	FileServer.exe	48.16%	481,641 kr.

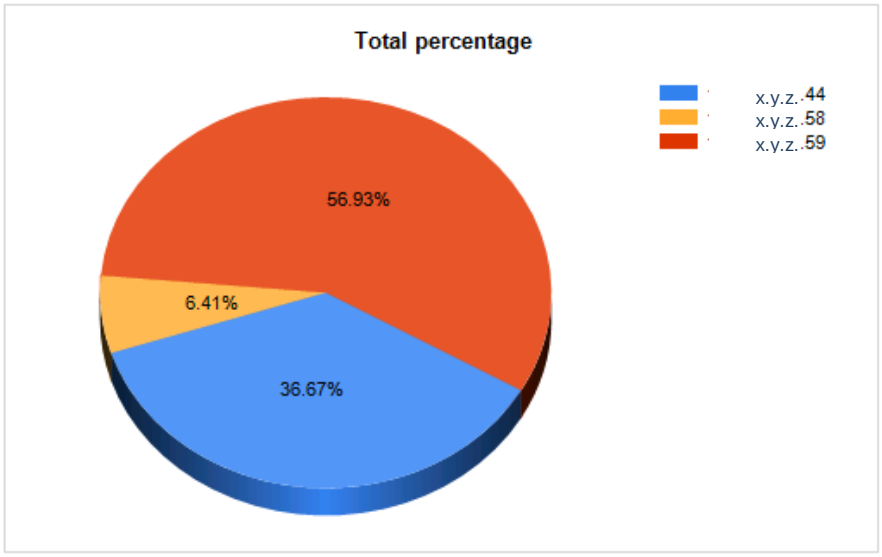
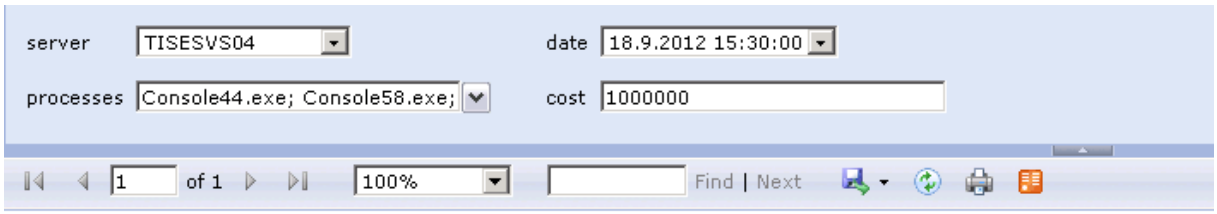


Figure A.10: Scenario two for TISESVS02, random.



Server: TISESVS04

Date: 18. September 2012 - 15:30:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS04	x.y.z..44	Console58.exe	5.64%	56,440 kr.
TISESVS04	x.y.z..44	FileServer.exe	44.87%	448,698 kr.
TISESVS04	x.y.z..58	Console44.exe	9.15%	91,504 kr.
TISESVS04	x.y.z..58	Console58.exe	18.85%	188,535 kr.
TISESVS04	x.y.z..58	Console59.exe	11.76%	117,621 kr.
TISESVS04	x.y.z..59	Console58.exe	9.72%	97,203 kr.

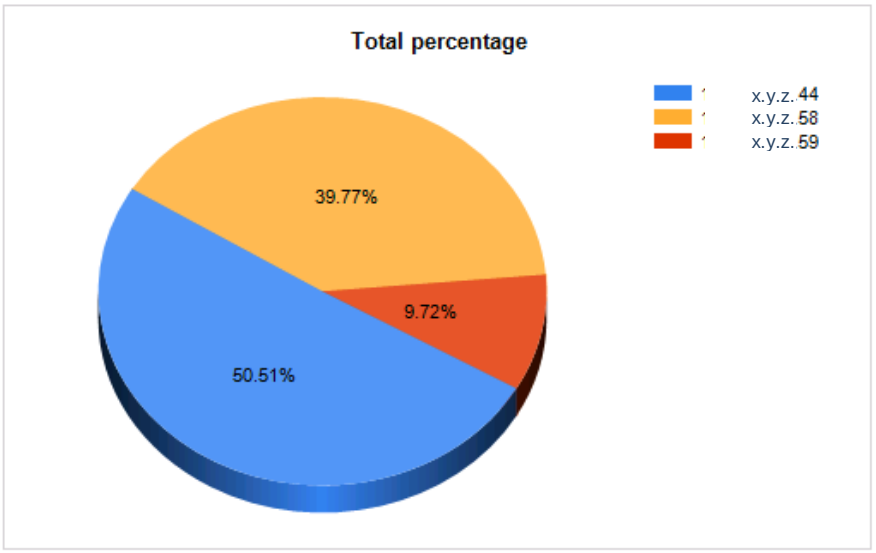


Figure A.11: Scenario two for TISESVS04, random.

Server: TISESVS06

Date: 18. September 2012 - 15:30:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS06	x.y.z.44	Console59.exe	8.58%	85,840 kr.
TISESVS06	x.y.z.58	Console59.exe	9.18%	91,829 kr.
TISESVS06	x.y.z.58	FileServer.exe	44.97%	449,713 kr.
TISESVS06	x.y.z.59	Console44.exe	9.11%	91,097 kr.
TISESVS06	x.y.z.59	Console58.exe	8.98%	89,843 kr.
TISESVS06	x.y.z.59	Console59.exe	19.17%	191,678 kr.

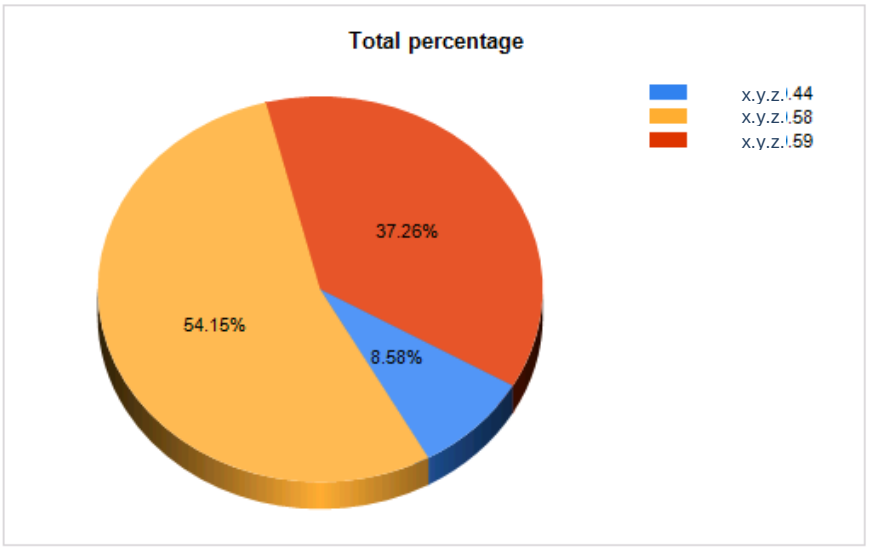


Figure A.12: Scenario two for TISESVS06, random.

Third scenario

The third scenario had the time between memory and processor sampling behaviours decreased.

Standard timing

After execution of the planned scenario the cost model for each server was executed. See Figure A.13 for the corresponding cost model report of TISESVS02 (44), Figure A.14 for cost model report of TISESVS04 (58) and Figure A.15 for cost model report of TISESVS06 (59).

Server: TISESVS02

Date: 18. September 2012 - 15:54:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS02	x.y.z.44	Console44.exe	19.27%	192,737 kr.
TISESVS02	x.y.z.44	Console58.exe	9.02%	90,206 kr.
TISESVS02	x.y.z.44	Console59.exe	9.04%	90,381 kr.
TISESVS02	x.y.z.58	Console44.exe	8.54%	85,435 kr.
TISESVS02	x.y.z.59	Console44.exe	10.21%	102,105 kr.
TISESVS02	x.y.z.59	FileServer.exe	43.91%	439,135 kr.

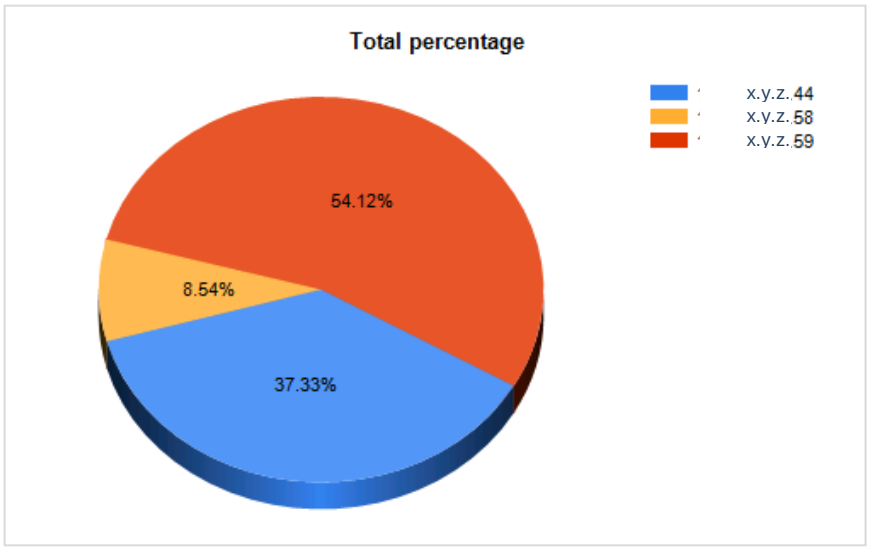


Figure A.13: Scenario three for TISESVS02, standard.

Server: TISESVS04

Date: 18. September 2012 - 15:54:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS04	x.y.z..44	Console58.exe	7.37%	73,725 kr.
TISESVS04	x.y.z..44	FileServer.exe	44.31%	443,122 kr.
TISESVS04	x.y.z..58	Console44.exe	9.12%	91,227 kr.
TISESVS04	x.y.z..58	Console58.exe	18.23%	182,281 kr.
TISESVS04	x.y.z..58	Console59.exe	9.68%	96,753 kr.
TISESVS04	x.y.z..59	Console58.exe	11.29%	112,892 kr.

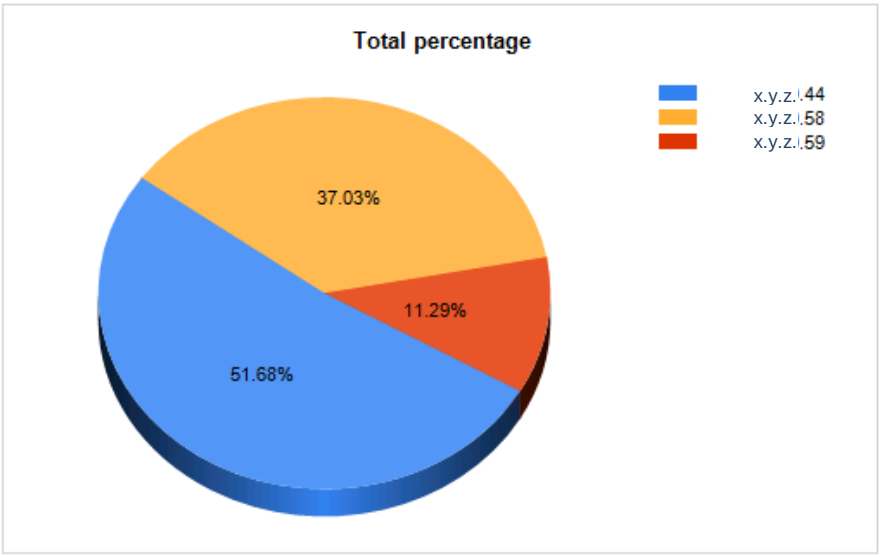
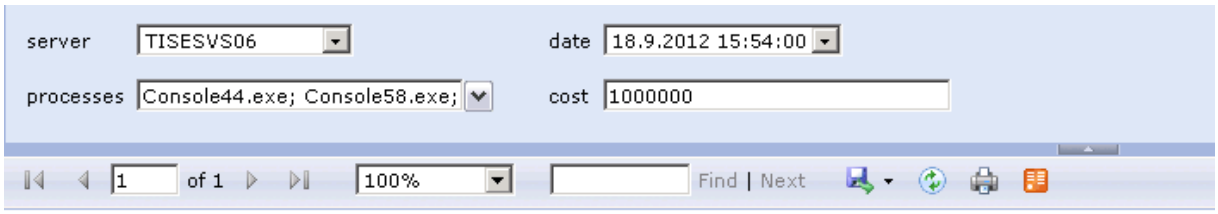


Figure A.14: Scenario three for TISESVS04, standard.



Server: TISESVS06

Date: 18. September 2012 - 15:54:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS06	x.y.z..44	Console59.exe	11.51%	115,143 kr.
TISESVS06	x.y.z..58	Console59.exe	9.85%	98,525 kr.
TISESVS06	x.y.z..58	FileServer.exe	40.08%	400,818 kr.
TISESVS06	x.y.z..59	Console44.exe	9.10%	90,950 kr.
TISESVS06	x.y.z..59	Console58.exe	8.97%	89,698 kr.
TISESVS06	x.y.z..59	Console59.exe	20.49%	204,866 kr.

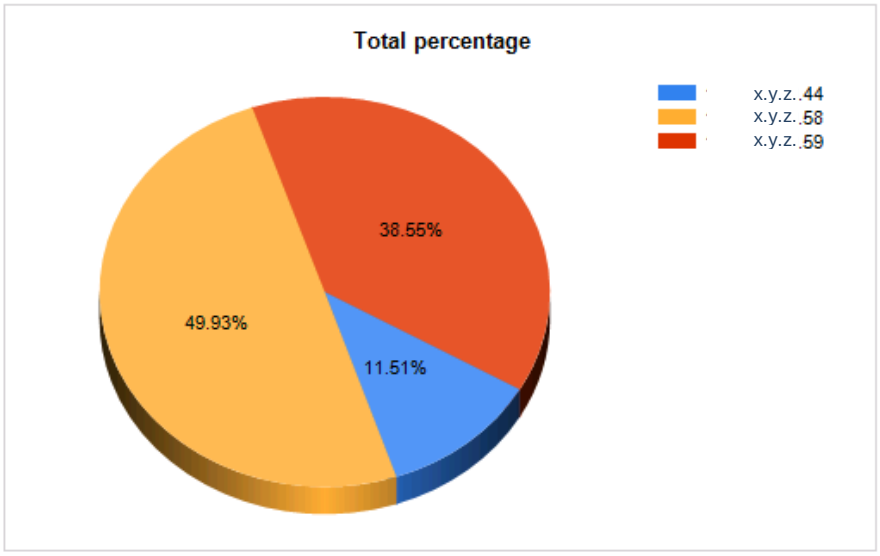
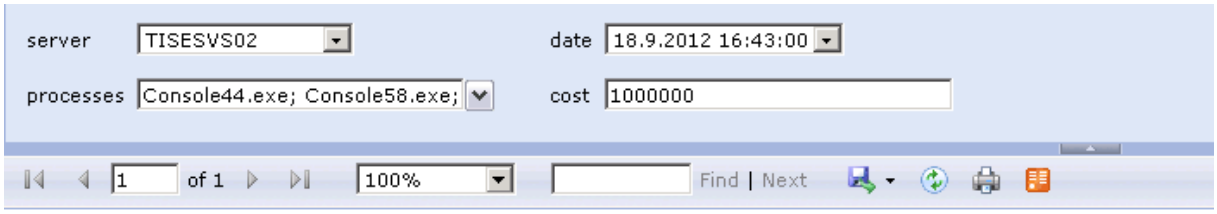


Figure A.15: Scenario three for TISESVS06, standard.

Random timing

After execution of the planned scenario the cost model for each server was executed. See Figure A.16 for the corresponding cost model report of TISESVS02 (44), Figure A.17 for cost model report of TISESVS04 (58) and Figure A.18 for cost model report of TISESVS06 (59).



Server: TISESVS02

Date: 18. September 2012 - 16:43:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS02	x.y.z..44	Console44.exe	19.89%	198,863 kr.
TISESVS02	x.y.z..44	Console58.exe	9.01%	90,141 kr.
TISESVS02	x.y.z..44	Console59.exe	9.03%	90,316 kr.
TISESVS02	x.y.z..58	Console44.exe	10.44%	104,429 kr.
TISESVS02	x.y.z..59	Console44.exe	9.28%	92,826 kr.
TISESVS02	x.y.z..59	FileServer.exe	42.34%	423,426 kr.

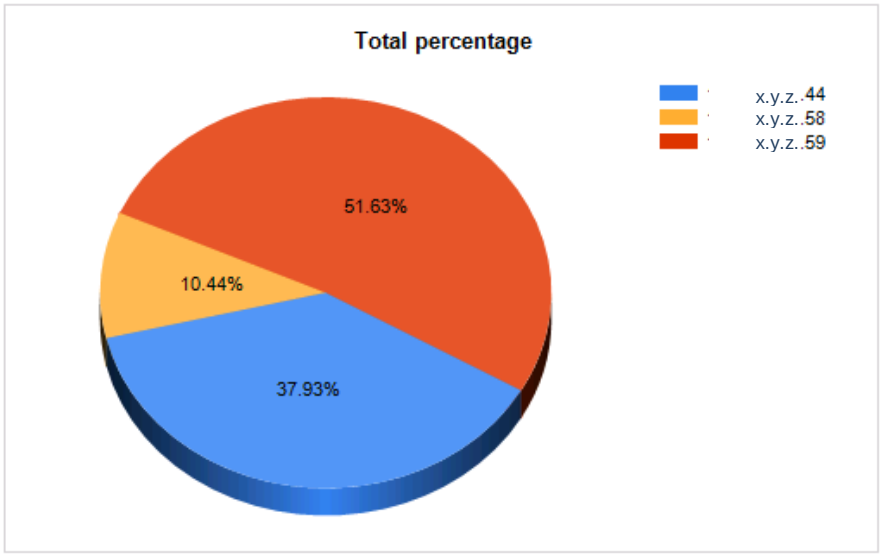
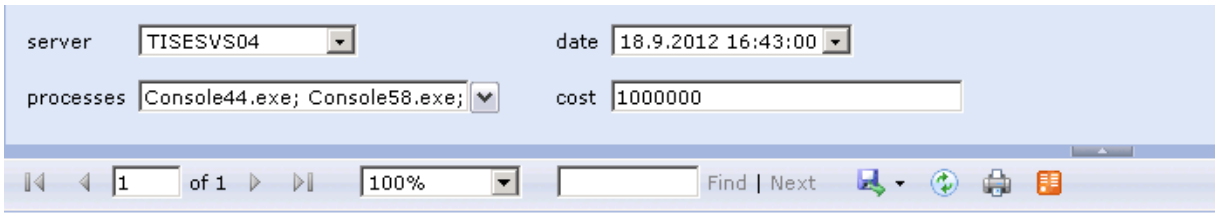


Figure A.16: Scenario three for TISESVS02, random.



Server: TISESVS04

Date: 18. September 2012 - 16:43:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS04	x.y.z.44	Console58.exe	4.84%	48,376 kr.
TISESVS04	x.y.z.44	FileServer.exe	49.54%	495,425 kr.
TISESVS04	x.y.z.58	Console44.exe	9.15%	91,474 kr.
TISESVS04	x.y.z.58	Console58.exe	18.10%	181,029 kr.
TISESVS04	x.y.z.58	Console59.exe	10.10%	100,988 kr.
TISESVS04	x.y.z.59	Console58.exe	8.27%	82,708 kr.

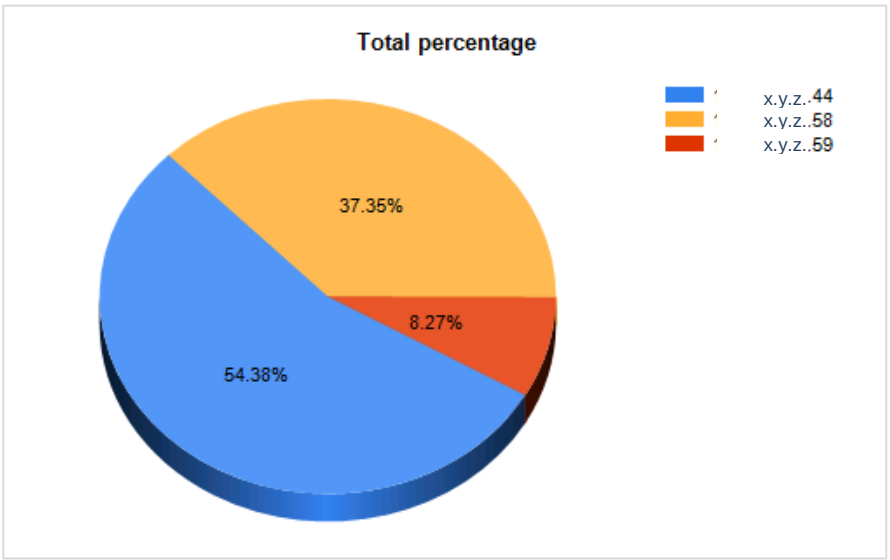
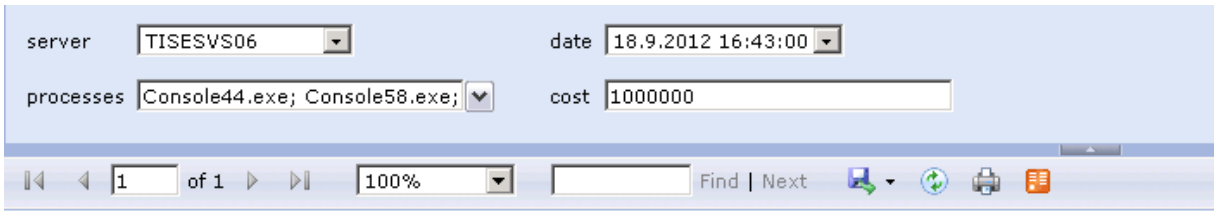


Figure A.17: Scenario three for TISESVS04, random.



Server: TISESVS06

Date: 18. September 2012 - 16:43:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS06	x.y.z..44	Console59.exe	9.22%	92,168 kr.
TISESVS06	x.y.z..58	Console59.exe	6.19%	61,925 kr.
TISESVS06	x.y.z..58	FileServer.exe	51.04%	510,398 kr.
TISESVS06	x.y.z..59	Console44.exe	9.11%	91,062 kr.
TISESVS06	x.y.z..59	Console58.exe	8.98%	89,808 kr.
TISESVS06	x.y.z..59	Console59.exe	15.46%	154,639 kr.

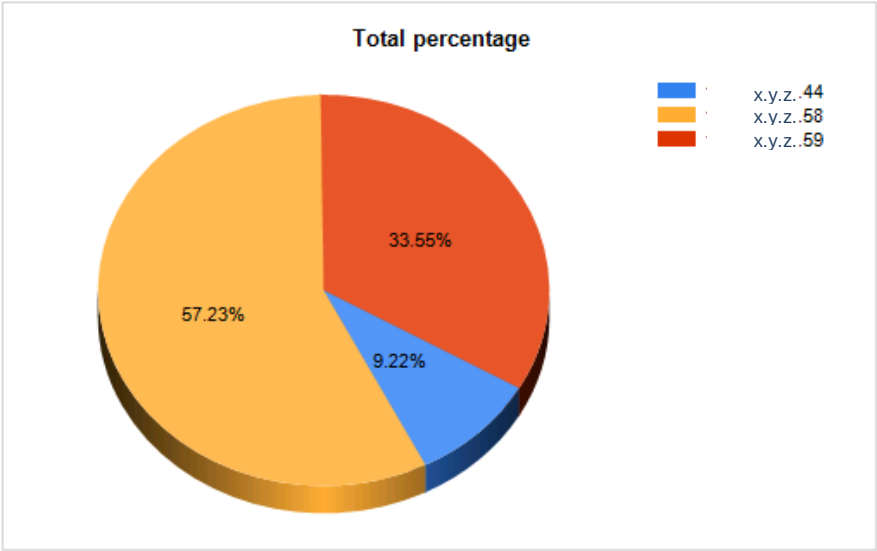


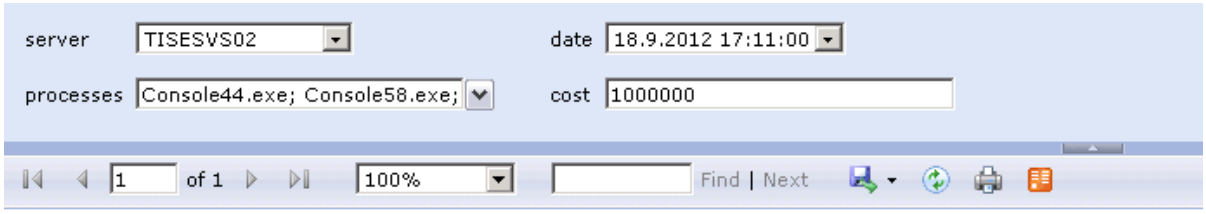
Figure A.18: Scenario three for TISESVS06, random.

Fourth scenario

The fourth scenario had the time between connection sampling behaviour increased.

Standard timing

After execution of the planned scenario the cost model for each server was executed. See Figure A.19 for the corresponding cost model report of TISESVS02 (44), Figure A.20 for cost model report of TISESVS04 (58) and Figure A.21 for cost model report of TISESVS06 (59).



Server: TISESVS02

Date: 18. September 2012 - 17:11:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS02	x.y.z. 44	Console44.exe	13.68%	136,810 kr.
TISESVS02	x.y.z. 44	Console58.exe	9.01%	90,145 kr.
TISESVS02	x.y.z. 44	Console59.exe	9.03%	90,320 kr.
TISESVS02	x.y.z. 58	Console44.exe	3.75%	37,545 kr.
TISESVS02	x.y.z. 59	Console44.exe	3.12%	31,222 kr.
TISESVS02	x.y.z. 59	FileServer.exe	61.40%	613,958 kr.

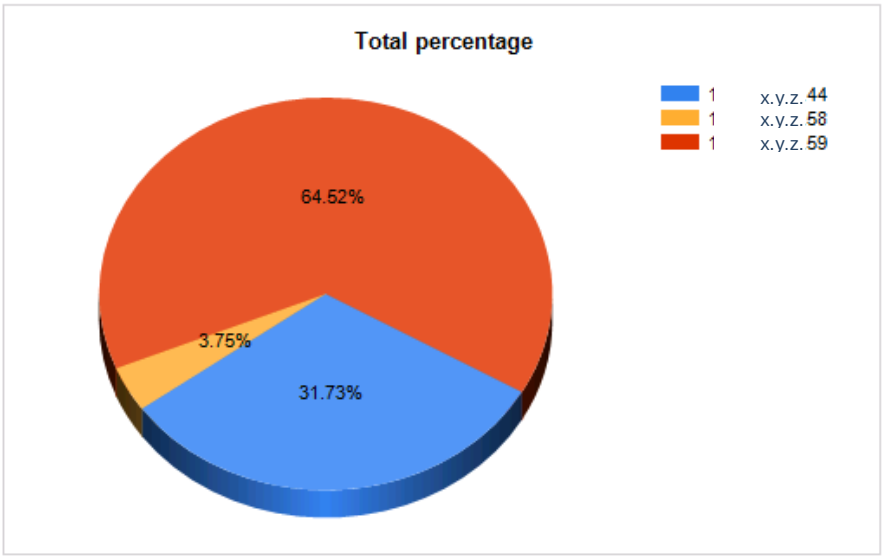


Figure A.19: Scenario four for TISESVS02, standard.

Server: TISESVS04

Date: 18. September 2012 - 17:11:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS04	x.y.z..44	Console58.exe	4.01%	40,097 kr.
TISESVS04	x.y.z..44	FileServer.exe	55.88%	558,805 kr.
TISESVS04	x.y.z..58	Console44.exe	12.88%	128,788 kr.
TISESVS04	x.y.z..58	Console58.exe	13.88%	138,811 kr.
TISESVS04	x.y.z..58	Console59.exe	9.02%	90,194 kr.
TISESVS04	x.y.z..59	Console58.exe	4.33%	43,305 kr.

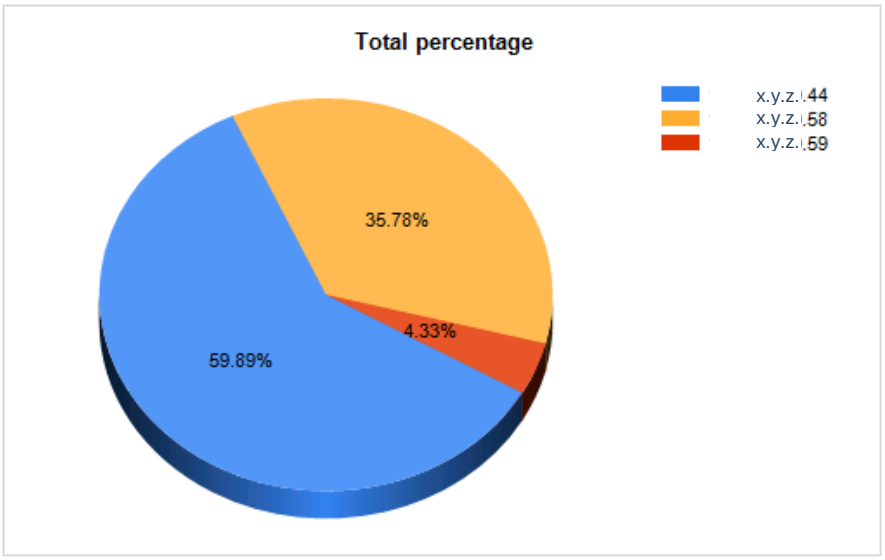


Figure A.20: Scenario four for TISESVS04, standard.

Server: TISESVS06

Date: 18. September 2012 - 17:11:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS06	x.y.z.44	Console59.exe	4.57%	45,746 kr.
TISESVS06	x.y.z.58	Console59.exe	3.78%	37,790 kr.
TISESVS06	x.y.z.58	FileServer.exe	51.67%	516,723 kr.
TISESVS06	x.y.z.59	Console44.exe	9.09%	90,918 kr.
TISESVS06	x.y.z.59	Console58.exe	13.87%	138,686 kr.
TISESVS06	x.y.z.59	Console59.exe	17.01%	170,137 kr.

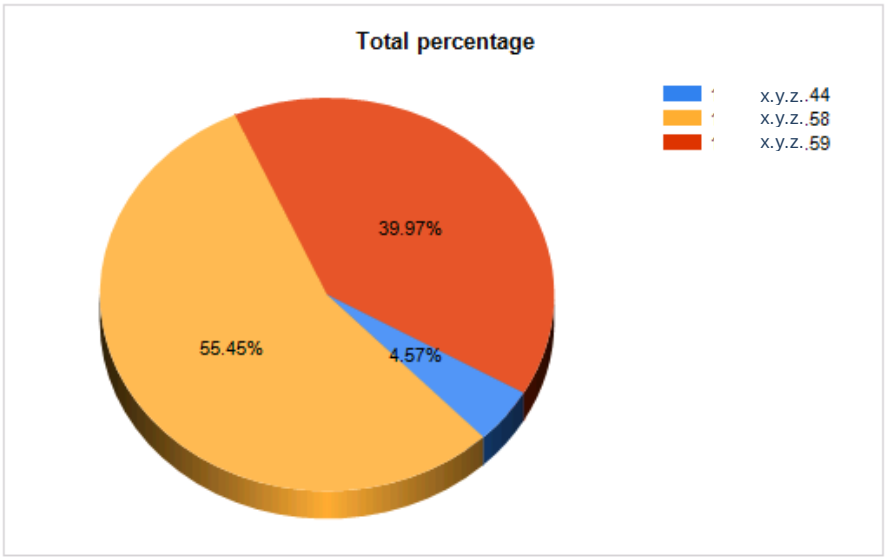


Figure A.21: Scenario four for TISESVS06, standard.

Random timing

After execution of the planned scenario the cost model for each server was executed. See Figure A.22 for the corresponding cost model report of TISESVS02 (44), Figure A.23 for cost model report of TISESVS04 (58) and Figure A.24 for cost model report of TISESVS06 (59).

Server: TISESVS02

Date: 18. September 2012 - 16:19:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS02	x.y.z..44	Console44.exe	19.15%	191,530 kr.
TISESVS02	x.y.z..44	Console58.exe	9.03%	90,335 kr.
TISESVS02	x.y.z..44	Console59.exe	9.05%	90,510 kr.
TISESVS02	x.y.z..58	Console44.exe	5.68%	56,798 kr.
TISESVS02	x.y.z..59	Console44.exe	4.58%	45,804 kr.
TISESVS02	x.y.z..59	FileServer.exe	52.50%	525,024 kr.

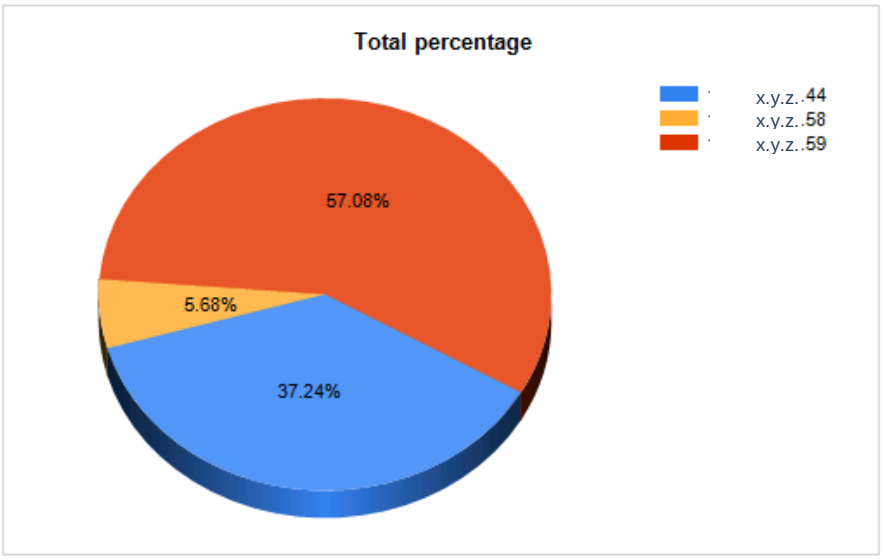
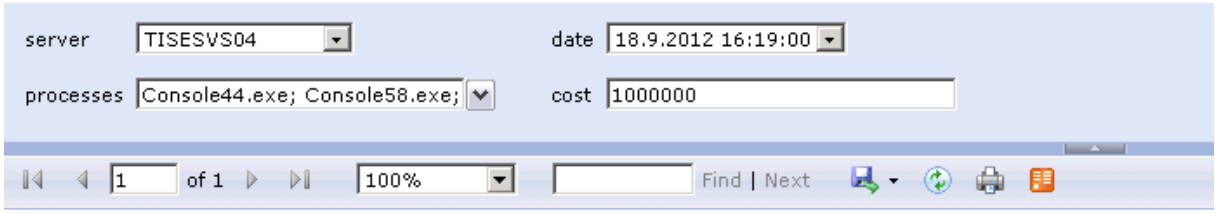


Figure A.22: Scenario four for TISESVS02, random.



Server: TISESVS04

Date: 18. September 2012 - 16:19:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS04	x.y.z. 44	Console58.exe	9.83%	98,256 kr.
TISESVS04	x.y.z. 44	FileServer.exe	47.41%	474,112 kr.
TISESVS04	x.y.z. 58	Console44.exe	9.15%	91,456 kr.
TISESVS04	x.y.z. 58	Console58.exe	16.36%	163,552 kr.
TISESVS04	x.y.z. 58	Console59.exe	9.03%	90,301 kr.
TISESVS04	x.y.z. 59	Console58.exe	8.23%	82,323 kr.

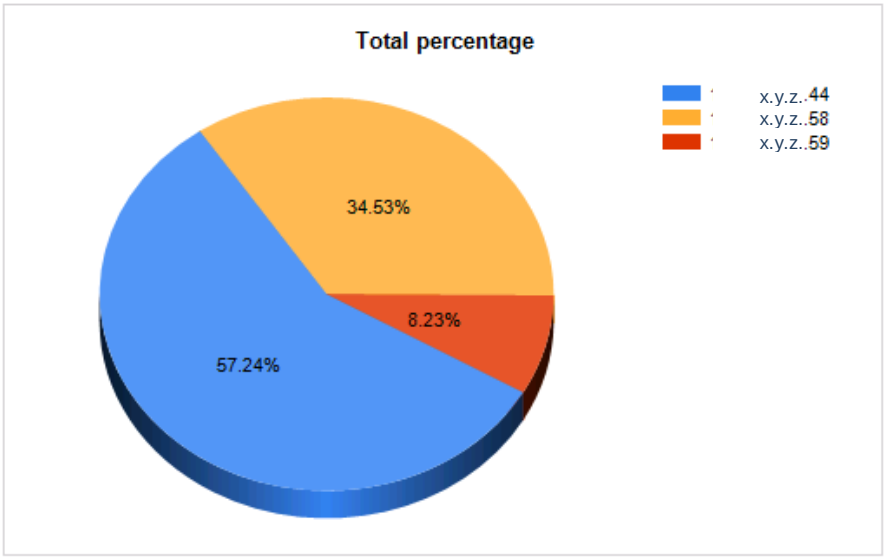
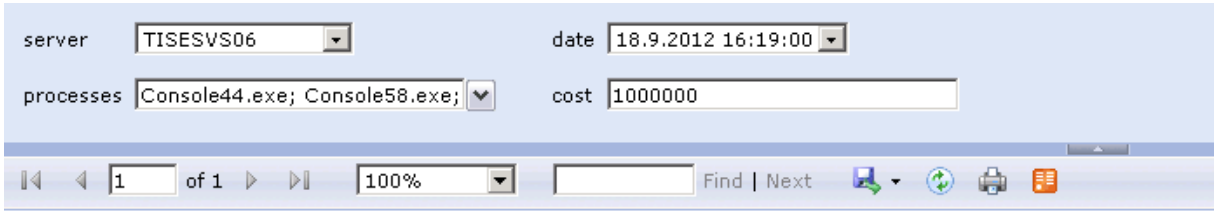


Figure A.23: Scenario four for TISESVS04, random.



Server: TISESVS06

Date: 18. September 2012 - 16:19:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS06	x.y.z..44	Console59.exe	11.92%	119,240 kr.
TISESVS06	x.y.z..58	Console59.exe	7.65%	76,494 kr.
TISESVS06	x.y.z..58	FileServer.exe	44.25%	442,511 kr.
TISESVS06	x.y.z..59	Console44.exe	10.04%	100,414 kr.
TISESVS06	x.y.z..59	Console58.exe	10.86%	108,583 kr.
TISESVS06	x.y.z..59	Console59.exe	15.28%	152,759 kr.

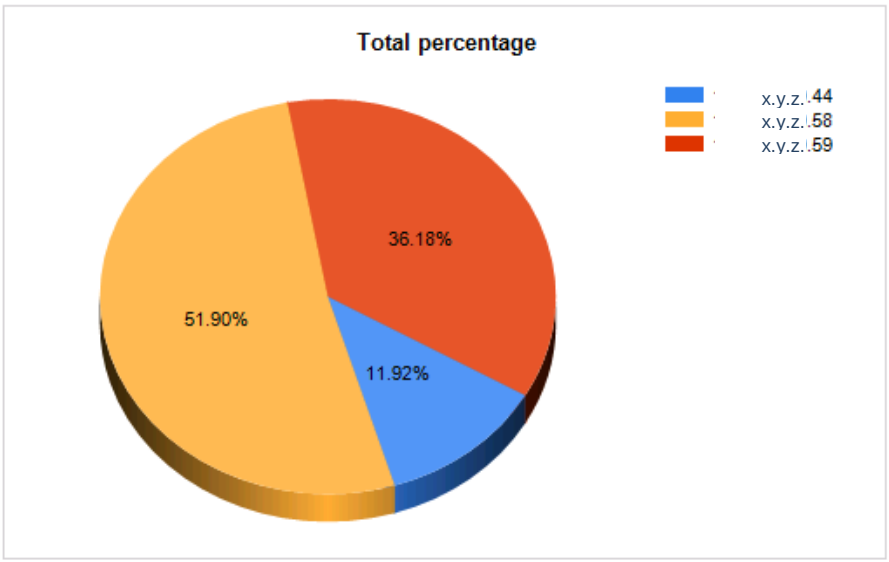


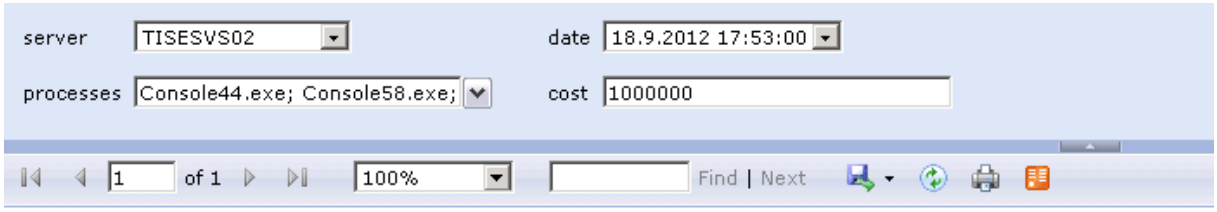
Figure A.24: Scenario four for TISESVS06, random.

Fifth scenario

The fifth scenario had the time between connection sampling behaviour decreased.

Standard timing

After execution of the planned scenario the cost model for each server was executed. See Figure A.25 for the corresponding cost model report of TISESVS02 (44), Figure A.26 for cost model report of TISESVS04 (58) and Figure A.27 for cost model report of TISESVS06 (59).



Server: TISESVS02

Date: 18. September 2012 - 17:53:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS02	x.y.z..44	Console44.exe	26.39%	263,909 kr.
TISESVS02	x.y.z..44	Console58.exe	18.02%	180,176 kr.
TISESVS02	x.y.z..44	Console59.exe	18.05%	180,526 kr.
TISESVS02	x.y.z..58	Console44.exe	8.24%	82,375 kr.
TISESVS02	x.y.z..59	Console44.exe	6.49%	64,874 kr.
TISESVS02	x.y.z..59	FileServer.exe	22.81%	228,141 kr.

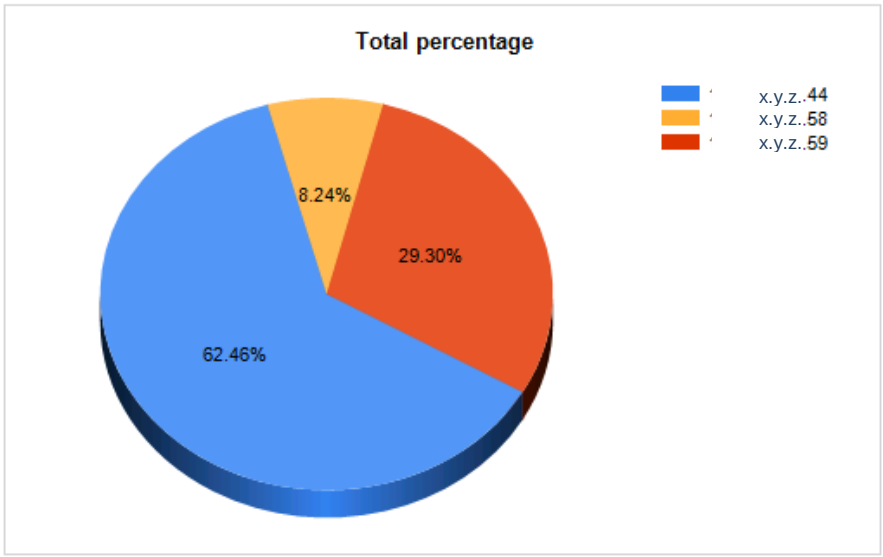


Figure A.25: Scenario five for TISESVS02, standard.

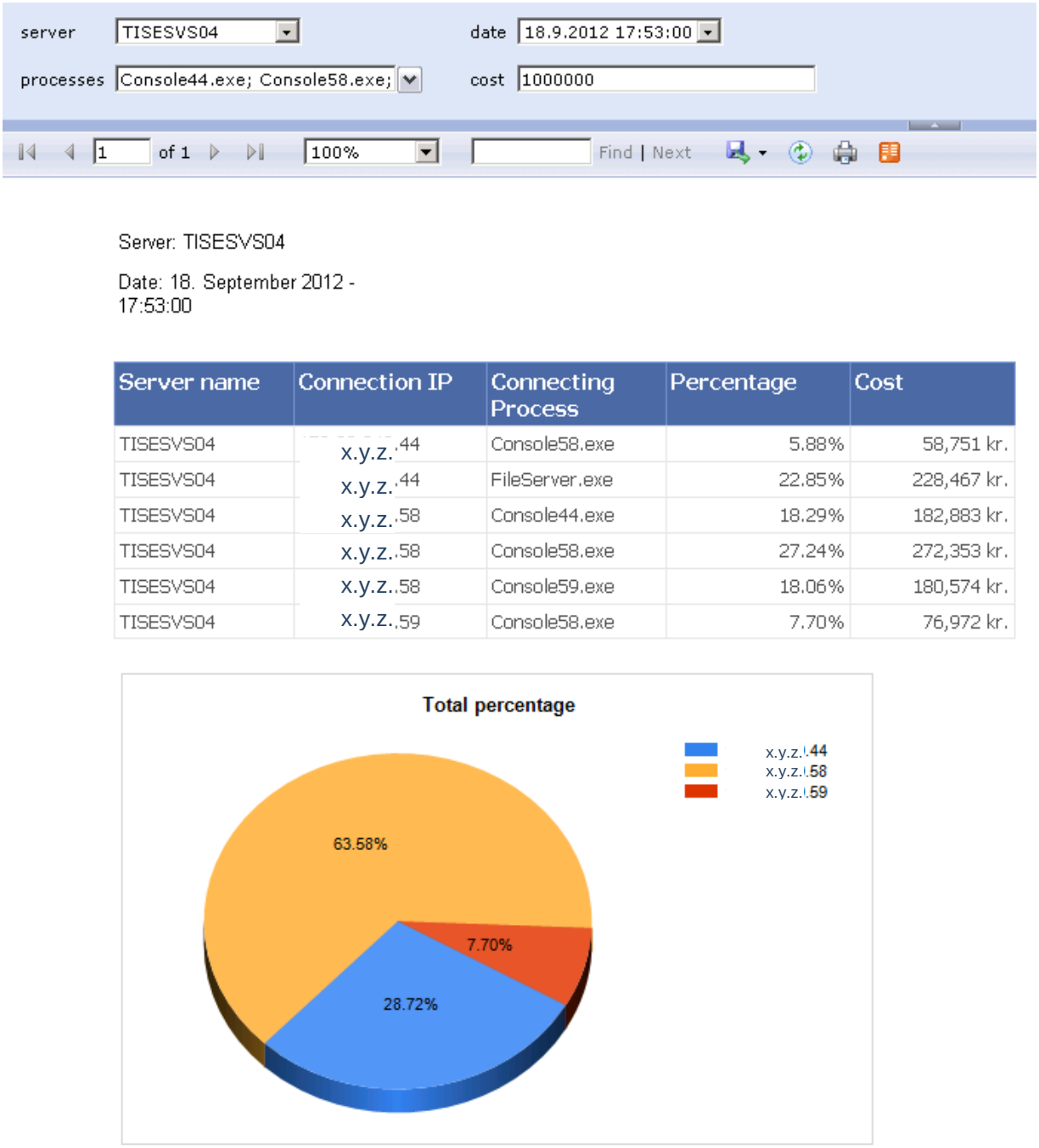
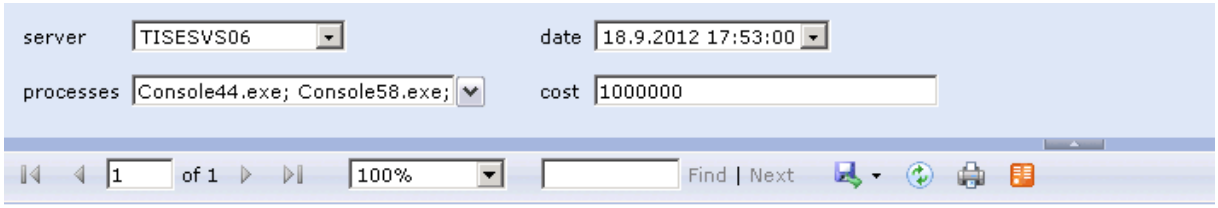


Figure A.26: Scenario five for TISESVS04, standard.



Server: TISESVS06

Date: 18. September 2012 - 17:53:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS06	: x.y.z.44	Console59.exe	8.32%	83,192 kr.
TISESVS06	: x.y.z.58	Console59.exe	8.23%	82,325 kr.
TISESVS06	: x.y.z.58	FileServer.exe	49.87%	498,673 kr.
TISESVS06	: x.y.z.59	Console44.exe	9.08%	90,840 kr.
TISESVS06	: x.y.z.59	Console58.exe	8.96%	89,589 kr.
TISESVS06	: x.y.z.59	Console59.exe	15.54%	155,380 kr.

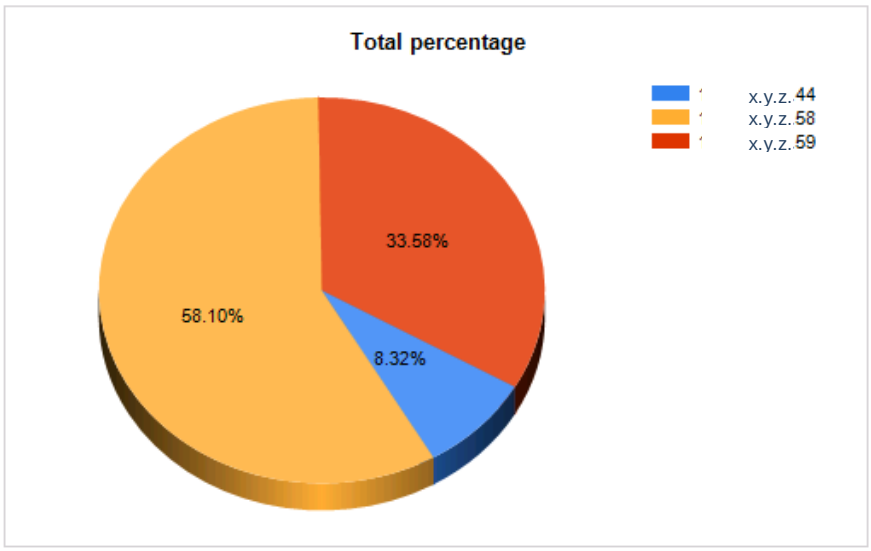
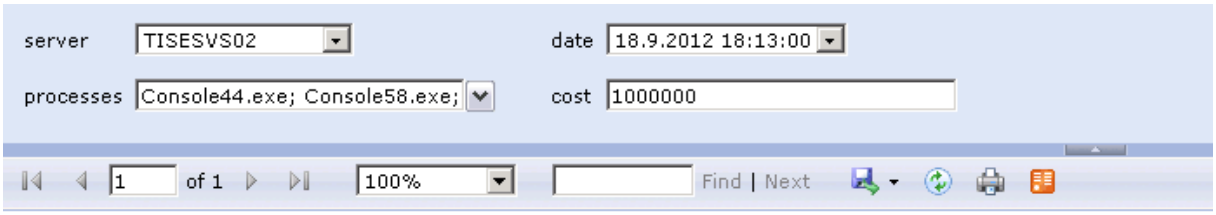


Figure A.27: Scenario five for TISESVS06, standard.

Random timing

After execution of the planned scenario the cost model for each server was executed. See Figure A.28 for the corresponding cost model report of TISESVS02 (44), Figure A.29 for cost model report of TISESVS04 (58) and Figure A.30 for cost model report of TISESVS06 (59).



Server: TISESVS02

Date: 18. September 2012 - 18:13:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS02	x.y.z..44	Console44.exe	25.30%	253,008 kr.
TISESVS02	x.y.z..44	Console58.exe	18.01%	180,128 kr.
TISESVS02	x.y.z..44	Console59.exe	18.05%	180,477 kr.
TISESVS02	x.y.z..58	Console44.exe	6.94%	69,412 kr.
TISESVS02	x.y.z..59	Console44.exe	8.88%	88,769 kr.
TISESVS02	x.y.z..59	FileServer.exe	22.82%	228,206 kr.

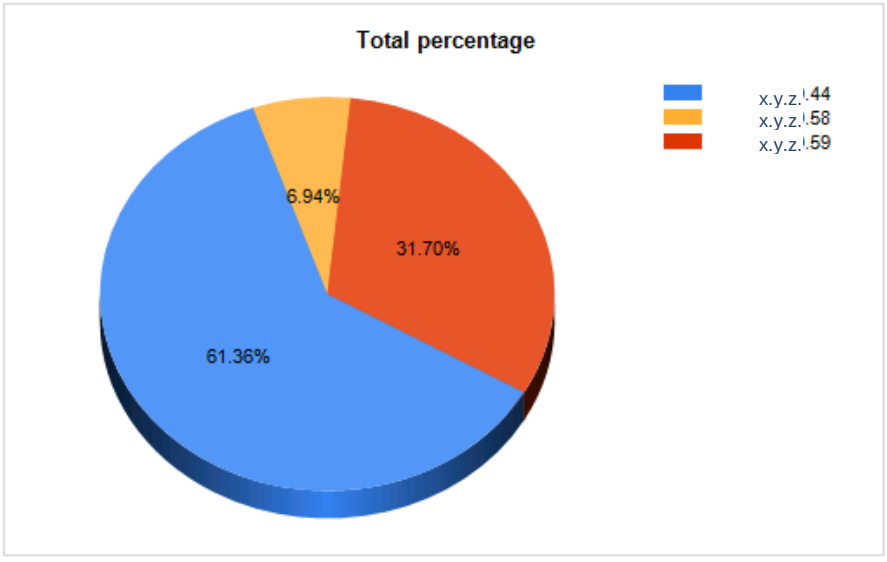
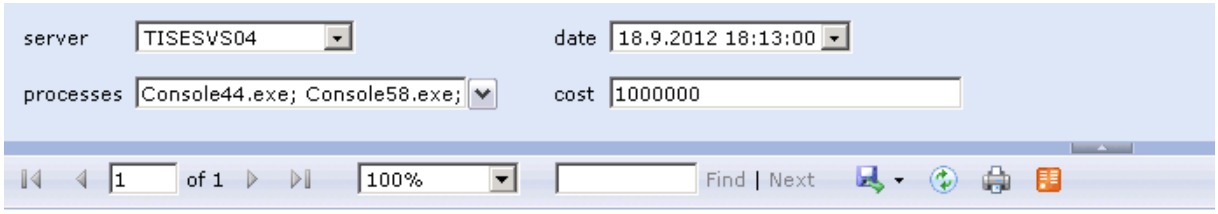


Figure A.28: Scenario five for TISESVS02, random.



Server: TISESVS04

Date: 18. September 2012 - 18:13:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS04	x.y.z..44	Console58.exe	7.01%	70,123 kr.
TISESVS04	x.y.z..44	FileServer.exe	40.43%	404,280 kr.
TISESVS04	x.y.z..58	Console44.exe	9.12%	91,203 kr.
TISESVS04	x.y.z..58	Console58.exe	20.23%	202,310 kr.
TISESVS04	x.y.z..58	Console59.exe	9.01%	90,051 kr.
TISESVS04	x.y.z..59	Console58.exe	14.20%	142,034 kr.

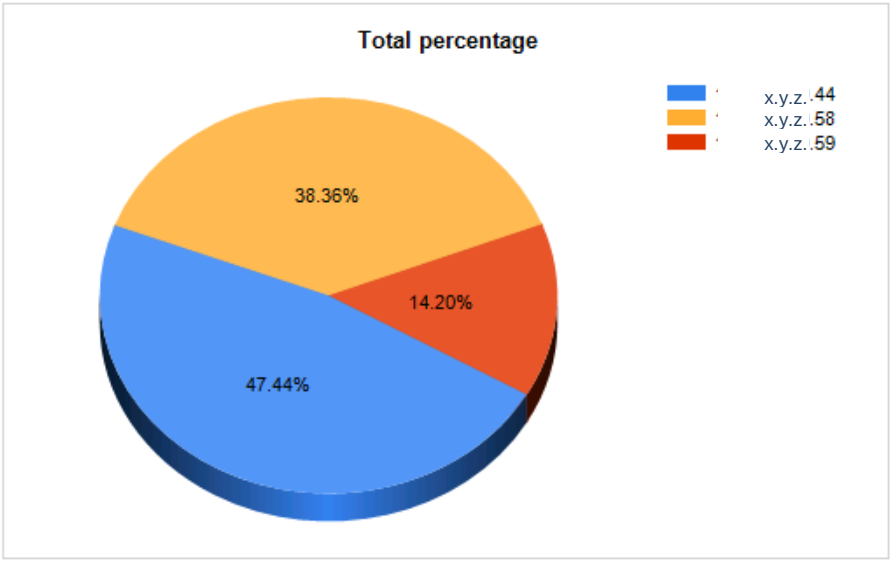
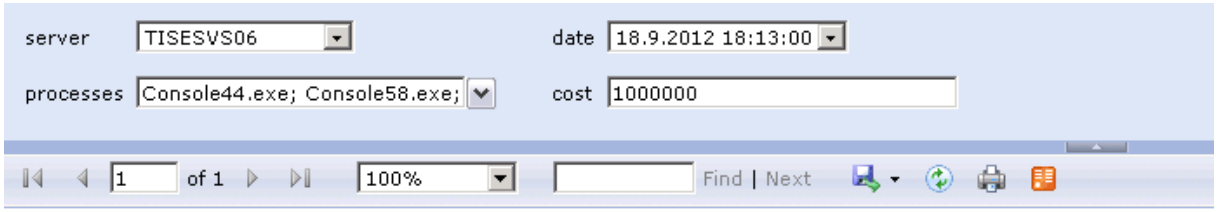


Figure A.29: Scenario five for TISESVS04, random.



Server: TISESVS06

Date: 18. September 2012 - 18:13:00

Server name	Connection IP	Connecting Process	Percentage	Cost
TISESVS06	x.y.z.44	Console59.exe	7.16%	71,609 kr.
TISESVS06	x.y.z.58	Console59.exe	6.98%	69,784 kr.
TISESVS06	x.y.z.58	FileServer.exe	22.89%	228,875 kr.
TISESVS06	x.y.z.59	Console44.exe	18.24%	182,407 kr.
TISESVS06	x.y.z.59	Console58.exe	17.99%	179,896 kr.
TISESVS06	x.y.z.59	Console59.exe	26.74%	267,431 kr.

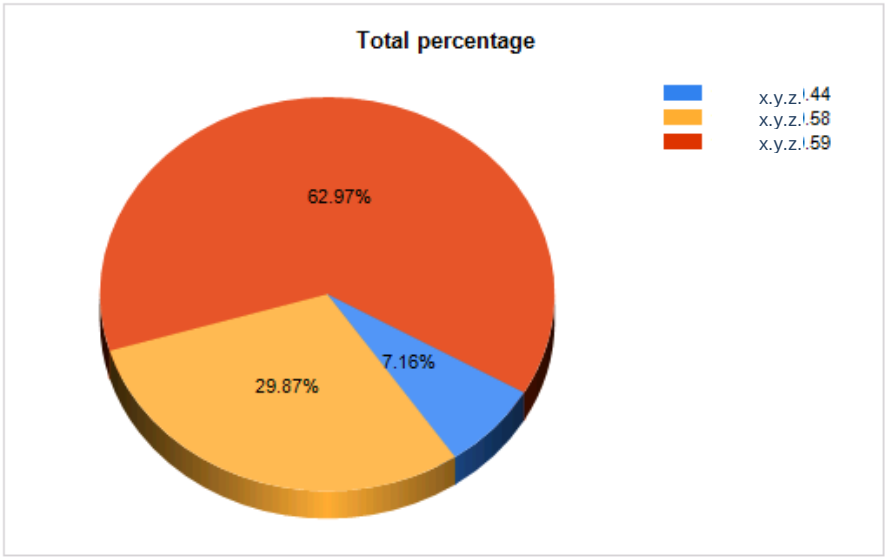


Figure A.30: Scenario five for TISESVS06, random.