Capacity Planning of Mobile Agent Systems

Designing Efficient Intranet Applications

Dissertation

zur Erlangung des Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)

vorgelegt dem Fachbereich 5,
Institut für Informatik und Wirtschaftsinformatik
der Universität Duisburg-Essen (Campus Essen)

von
Corinna Flüs, geboren in Letmathe

Datum der mündlichen Prüfung: 24. Februar 2005

Gutachter:

Prof. Dr. Bruno Müller-Clostermann
(Universität Duisburg-Essen)

Prof. Dr. Wilhelm R. Rossak
(Friedrich-Schiller-Universität Jena)

To my parents, Katharina and Lothar Flüs

Acknowledgements

I am going to write the following lines in my native language, German. The people I would like to address all understand German, furthermore, it is easier for me to express my feelings and gratefulness with my mother tongue.

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftliche Mitarbeiterin in der Arbeitsgruppe Systemmodellierung im Institut für Informatik und Wirtschaftsinformatik an der Universität Duisburg-Essen. Das Gelingen dieser Arbeit haben ganz unterschiedliche, mir allesamt liebe Menschen möglich gemacht, die mich in fachlicher wie auch privater Hinsicht unterstützt haben.

Zuerst möchte ich meinem Doktorvater, Herrn Prof. Dr. Bruno Müller-Clostermann herzlich danken. Ich habe ihm den größten eigenen Abschnitt im Vorwort versprochen - hier ist er nun. Dieser eigene Abschnitt liegt nicht nur in der Tatsache begründet, dass er meine Arbeit betreut hat. Er hat mir dabei große Entwicklungsfreiräume gelassen, mich aber stets auf den "rechten Weg" zurückgebracht, wenn ich drohte, mich in der Vielfältigkeit des Themas zu verlieren oder Gefahr lief, in eine Sackgasse zu geraten. Darüber hinaus schafft er in seiner Arbeitsgruppe ein kollegiales, menschliches und produktives Arbeitsklima, in dem es Spaß macht zu arbeiten. Schließlich hat er mir in für mich privat sehr schwierigen Zeiten alle eben möglichen Freiheiten gewährt und mir mit großem Verständnis und Mitgefühl geholfen.

Herrn Prof. Dr. Wilhelm Rossak danke ich für die spontane Zusage, das Zweitgutachten für diese Arbeit zu erstellen und für seine große Flexibilität und Unterstützung bei der zügigen Abwicklung des Promotionsverfahrens. Weiterhin danke ich ihm dafür, dass er es seinen Mitarbeitern an der Friedrich-Schiller-Universität Jena ermöglichte, das Mobile-Agenten-System *Tracy* zu entwickeln, was ein wichtiger Bestandteil meiner Arbeit geworden ist.

Ich danke Herrn Prof. Dr. Klaus Echtle für die spontane Übernahme des Vorsitzes der Prüfungskommission und für seine wertvollen Tipps und Anregungen.

Lieben Dank auch an die Kollegen für die anregenden Diskussionen, das gegenseitige Bedauern in Krisenzeiten und für jede Menge Spaß in einer sehr produktiven Zeit. Besonders hervorheben möchte ich die Unterstützung durch Dr. Peter Braun von der Friedrich-Schiller-Universität Jena, den "Vater" von *Tracy,* Milen Tilev, der diverse praktische Tools und Anwendungen implemen-

tiert hat und Dr. Kay Wilhelm für den gefüllten Krisenzettel und dessen Abarbeitung ☺. Dank auch an den Meister der englischen Sprache, Roland Kempter, für's Korrekturlesen und an Andreas Pillekeit für die Hilfe beim Kampf mit Photoshop.

Im privaten Bereich gilt mein herzlichster Dank meinen Eltern, die mir stets liebevoll zur Seite standen und mich ermutigt haben, meine Ziele zu verfolgen. Sie haben meinen Sinn für das Wesentliche im beruflichen wie im privaten Leben geschärft und mir stets den Rücken gestärkt. Weiterhin gilt mein besonderer Dank meinem lieben Mann Dr. Jörg Hintelmann für die fruchtbaren fachlichen Diskussionen sowie für seinen Beistand in schwierigen Zeiten. Insgesamt herzlichen Dank an meine Familie für ihre Unterstützung. Schließlich möchte ich mich bei allen Freunden, insbesondere bei Aloys von der Stein, für die lieb gemeinten Sticheleien bedanken und für ihr Drängen auf die Fertigstellung dieser Arbeit.

Abstract

Mobile agents are a quite new and interesting paradigm for the implementation of distributed systems. As with most distributed systems, mobile agent applications are usually developed and installed without regarding performance aspects. Typically, methods and tools for capacity planning differ fundamentally from methods and tools for system development, thus system developers often avoid additional modelling and planning effort. This dissertation helps to solve this problem by presenting an approach to easy integrate performance modelling into the development process of mobile agent applications. Most mobile agent applications contain the same basic scenarios, which include stationary agents with the role of servers and mobile agents as clients. Based on these scenarios, this dissertation describes a new modelling approach and a methodology for capacity planning of mobile agent systems with an emphasis on intranet applications.

The core idea of the new modelling approach is to directly integrate byte code of real agents in a simulation environment. Thus, it is not necessary to describe agents' behaviour on a high abstraction level. Their behaviour results from their program code. To build performance models, a system developer mainly has to specify the infrastructure of the mobile agent system and parameters for time consumption. Moreover, this dissertation focuses on providing algorithms to increase the efficiency of simulation models of mobile agent systems. As existing approaches are not applicable to the presented modelling technique, new methods are developed which consider special features of mobile agent systems and which regard the objectives of this dissertation. A methodology for capacity planning of general heterogeneous IT systems is adjusted to mobile agent systems according to the developed modelling techniques.

The modelling concepts and the methodology for capacity planning are first presented and explained. They are implemented using the mobile agent platform *Tracy*[1] and the simulation package *JavaDEMOS*[2]. Finally, the applicability of these approaches are demonstrated by a realistic case study.

---

1. *Tracy* has been developed at the Friedrich-Schiller-University of Jena by the research group of Prof. Dr. Wilhelm Rossak.
2. *JavaDEMOS* has been developed at the University of Essen by the research group of Prof. Dr. Bruno Müller-Clostermann.

Contents

1 Introduction **1**
1.1 Capacity Planning 2
1.2 Mobile Agents for Intranet Applications 2
1.3 Objectives and Contribution of this Dissertation 3
1.4 Related Approaches 4
1.4.1 Performance Measurement and Benchmarking 4
1.4.2 Performance Tuning 4
1.4.3 Performance Modelling 5
1.5 Outline 6

2 Simulation of Mobile Agent Systems **7**
2.1 Model Scenarios 8
2.2 The Modelling Method 8
2.2.1 Requirements for Performance Modelling 9
2.2.2 The Modelling Paradigm 9
2.3 Implementation of the Modelling Method 10
2.3.1 Tracy 11
2.3.2 JavaDEMOS 13
2.4 JaDEMAS: A Simulation Environment for Tracy Agent Systems 14
2.4.1 Agent Servers 15
2.4.2 Communication Mechanisms 18
2.4.3 Agents 19
2.4.4 Network Links 22
2.4.5 Workload Generation 23
2.4.6 Input Parameters 24
2.4.7 Modifications of Real Agent Code 25
2.4.8 Output Analysis 26
2.5 Example 27
2.5.1 Parameterising via the Graphical User Interface 27
2.5.2 Code Modification 30
2.5.3 Results 31
2.6 Portability to Other Mobile Agent Systems 33
2.7 Summary 34

3 Existing Approaches to Increase Simulation Efficiency **35**
3.1 Aggregation and Decomposition 36
3.2 Simalytic Hybrid Modelling 38
3.3 Rare Event Simulation 38
3.4 Hybrid Simulation 39
3.5 Small Scale Hi-fidelity Reproduction of Network Kinetics 40
3.6 Summary 44

4 Efficient Simulation of Mobile Agent Systems **47**
4.1 Experimental Environment 48
4.2 Concatenated Servers 49
4.2.1 Concatenating Mobile Agent Servers 49
4.2.2 Calculation of the Accumulated Delay 51
4.3 Concatenated FIFO Servers (CFS) 52

4.3.1 Calculation of Service Agents' Residence Time at the I/O Device . . . 52
4.3.2 Error Estimation . . . 55
4.4 Empirical Evaluation of CFS . . . 56
4.4.1 An Introducing Example . . . 56
4.4.2 Steady State Analysis . . . 59
4.4.3 Finite Horizon Analysis . . . 64
4.5 Concatenated Round Robin Servers (CRRS) . . . 70
4.5.1 Calculation of Service Agents' Residence Time at the Processor . . . 71
4.5.2 Multi Processors . . . 72
4.5.3 Error Estimation . . . 72
4.6 Empirical Evaluation of CRRS . . . 75
4.6.1 Steady State Analysis . . . 75
4.6.2 Finite Horizon Analysis . . . 78
4.7 Summary . . . 83

5 Capacity Planning of Mobile Agent Systems . . . 85
5.1 Baseline Modelling . . . 86
5.1.1 Specification of the Infrastructure in the Model . . . 87
5.1.2 Workload Specification . . . 87
5.1.3 Modification of Agent Program Code . . . 89
5.1.4 Execution of Experiments and Preparation of Results . . . 89
5.1.5 Model Validation and Calibration . . . 91
5.2 Performance Prediction . . . 91
5.2.1 Specification of Future Scenarios . . . 92
5.2.2 Quality of Service Requirements . . . 92
5.2.3 Execution of Experiments . . . 92
5.2.4 Analysis of Results . . . 93
5.3 Mobile Agent Laboratory MOLAB . . . 93
5.3.1 Infrastructure . . . 93
5.3.2 Workload Generation . . . 94
5.3.3 Monitoring . . . 94
5.4 Summary . . . 95

6 Case Study . . . 97
6.1 Collegiate Timetable Service . . . 97
6.2 Baseline Modelling . . . 98
6.2.1 Specification of the Infrastructure in the Model . . . 99
6.2.2 Workload Specification . . . 100
6.2.3 Modification of Agent Program Code . . . 101
6.2.4 Execution of Experiments and Preparation of Results . . . 101
6.2.5 Model Validation and Calibration . . . 102
6.3 Performance Prediction . . . 105
6.3.1 Specification of Future Scenarios . . . 105
6.3.2 Quality of Service Requirements . . . 106
6.3.3 Execution of Experiments . . . 106
6.3.4 Analysis of Results . . . 107
6.4 Summary . . . 110

7 Summary and Outlook . . . 111
7.1 Techniques for Performance Modelling . . . 111
7.2 Methodology for Capacity Planning . . . 112
7.3 Applicability of the Approaches . . . 113

7.4 Outlook into Future Research 113

List of Figures 115

List of Tables 119

Bibliography 121

Appendix A Model Results 125

1 Introduction

Future challenges concerning development and operation of distributed information systems consist of handling the continuously growing number and heterogeneity of computers and mobility of users and devices. In the networks of a single institution there are heterogeneous components with different capacities and very different architectures. Nevertheless, these infrastructures are the foundations for Intranets where integrative applications shall run. Often, these intranet applications are implemented using web technology with classical client/server approaches. Alternatively, the paradigm of mobile agents can be used.

A mobile agent is a program - nowadays typically written in Java - which autonomically moves from node to node in a heterogeneous network. The computer where the mobile agent is generated is called *home server*. A mobile agent visits several remote servers and finally returns home. This journey is called *round trip*. A mobile agent always acts on behalf of its owner. During its round trip it is able to behave according to a self developed plan and to react to arising errors or other events. Mobile agents can communicate with the computer (*agent server*) on which they are just located and they can use services of this computer. Furthermore, there exist stationary agents in a mobile agent system. Stationary agents are usually regarded trustworthy, they have permissions to access local system resources. Stationary agents welcome mobile ones and provide system services. Mobile agents are untrusted, they only have restricted rights. Only mobile agents can migrate between agent servers. Furthermore, agents are able to transfer messages among each other, so they can be used to collaboratively solve problems. Agents require a special software, a so-called *agent platform* at the agent server, where they are hosted. The agent platform is located on top of the Java virtual machine and has the function of a middleware.

Mobile agent systems should not be mistaken for multiagent systems. Agents in multiagent systems are usually intelligent and not mobile. They are used to work collaboratively and, therefore, usually use artificial intelligence. Mobile agents may or may not be intelligent, but, in most applications they are not.

A general advantage of mobile agents compared to client/server architectures is the saving of bandwidth, which usually results in a smaller network latency. Moreover, mobile agents allow for an easy implementation of load balancing algorithms. With mobile end devices the main advantage lies in the fact that the link between mobile device (client which generates the agent)

and server does not have to exist permanently. The client can interrupt the link and re-establish it later to get the results from its agent. The focus of this dissertation is not to contribute to the discussion on a "killer application" for mobile agent systems or to argue whether the mobile agent paradigm is better than the client/server approach. In fact, the focus lies on efficient capacity planning for mobile agent applications.

Even though developers of distributed systems are faced with continuously growing networks and a growing number of users, typically, performance analysis of distributed applications or even capacity planning are disregarded. This often leads to severe performance problems when the application is already running in a production environment. Then, improving network performance becomes a costly and time consuming endeavour. System developers often avoid to analyse performance aspects in advance because methods and tools for performance modelling fundamentally differ from known methods and tools for system development. Hence, in practice, the additional effort for modelling is avoided.

Before taking a closer look at the objectives of this dissertation and at related research approaches, further basic items of the subject of this dissertation will be explained.

1.1 Capacity Planning

In the literature, different specifications of activities which belong to capacity planning there can be found. According to [6], p. I-3 (modified), in this dissertation we use the following definition: Capacity planning comprises all activities, to provide necessary resources to guarantee specified quality of service requirements of applications. This includes, e.g., tuning/extension of hardware resources and modification of software. Capacity planning is directed to future IT systems.

Performance modelling is an integral part of capacity planning: Resources necessary for future systems and performance characteristics of applications have to be predicted to provide the required quality of service. Performance models are used to predict the performance of various system versions and can be used to analyse what-if-scenarios to find a high-performance system architecture for future applications.

1.2 Mobile Agents for Intranet Applications

The paradigm of mobile agents is widely applicable. Originally, it has been developed for distributed applications in the Internet. The question if mobile agents become accepted in business is basically associated with security issues. Companies which run a mobile agent system allow foreign mobile agents to be executed at their agent servers. Mobile agents can act autonomously and the program code of foreign agents is usually not transparent. Thus, there is the risk that mobile agents could contain malicious code which corrupts server resources. For this reason, current mobile agent platforms contain several security mechanisms which protect servers against harmful mobile agents, but as well, to protect mobile agents against malicious agent servers, resp. stationary agents. Nevertheless, an element of risk remains.

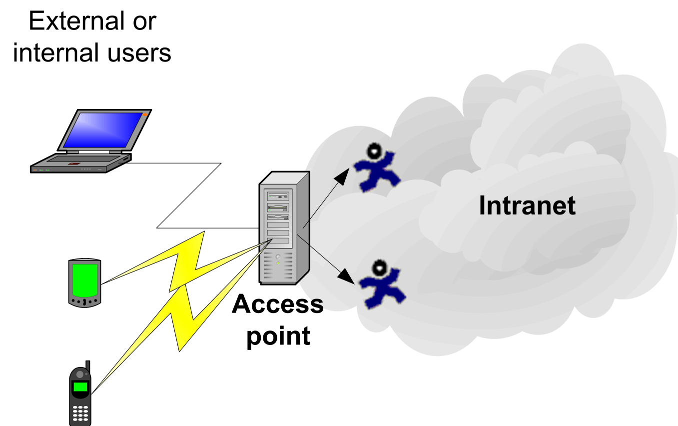

Figure 1-1: **Generation of mobile agents with intranet applications**

A higher level of security is provided if system operators and producers of mobile agents belong to the same company. This is the case if mobile agent applications run in intranets with dedicated network access points. This means, that mobile agents are generated under control of the company at these access points. Figure 1-1 shows such a scenario. Because of these considerations, this dissertation concentrates on capacity planning for mobile agent applications in intranets.

1.3 Objectives and Contribution of this Dissertation

This dissertation presents a methodology and techniques for efficient capacity planning of mobile agent systems. Thereby, special attention is drawn to performance modelling, which is a central aspect of capacity planning. Mobile agent systems are as well considered from a global point of view which includes application and infrastructure.

The capacity planning process as described e.g. in [6] is not identically portable to mobile agent systems. Such systems provide special challenges for capacity planners, e.g. different from common distributed applications they are more dynamic and they act autonomously. Usually, their behaviour is not predictable, i.e., e.g., their routes through the network or the servers they visit are not predictable. Hence, the capacity planning process is adjusted to special features of mobile agent systems.

The methodology and especially the techniques for performance modelling developed allow for the integration of capacity planning in early steps of the implementation of a mobile agent system. The basic idea is to transfer agents from the real system to a performance model with only minor modifications. Thus, agents alternatively can run in the real system or in a performance model. This way, the additional modelling effort for system developers is manageable. To implement this idea, performance models are built by means of simulation.

As generally known, simulation reaches its limits with very large or complex models. Hence, within this dissertation, approaches have been developed to increase simulation efficiency of mobile agent system models by using hybrid modelling techniques. Particularly, models of large agent systems benefit from these techniques.

Existing research concerning mobile agent performance mostly focuses on increasing the performance of mobile agent platforms by enhancing the design of the software. These functional modifications of mobile agent platforms is not the objective of this dissertation. Instead, existing

platforms are measured and modelled. If capacity planning results in the need of improving the performance, the agent code (the application) or the underlying infrastructure has to be tuned.

1.4 Related Approaches

Since 1997, research in the area of mobile agents developed very rapidly. In parallel to the wide spreading of Java-based applications, mobile agents can be found in research projects at universities and in a few industrial projects. Examples for mobile agent systems which were developed by industrial companies are *Aglets* from IBM, *Voyager* from ObjectSpace, *Concordia* from Mitsubishi, *Grashopper* from IKV++ Technologies AG and *Jade* from Telecom Italia Lab. Examples for universitary systems are *Mole* developed at the University of Stuttgart, *Ara* from University of Kaiserslautern, *MAP* from University of Catania and *Tracy* from University of Jena.

Former research was basically focused on programming and communication languages for mobile agents. Furthermore, security aspects were of major interest. Researchers who touched subjects of capacity planning only investigated single aspects of the whole process. The development of a methodology for performance modelling or capacity planning was not intended. Furthermore, mainly single parts of the architecture of the mobile agent system were looked into, mostly aspects of network load. A global view on the system has been missing, which includes the application level and the underlying infrastructure.

Existing performance models are limited either to mathematical solutions or to pure simulation. Both paradigms are used at a high abstraction level. Some approaches intend to generally tune mobile agent systems. Therefor, the agent platform software was tuned. The following sections give an overview about performance aspects in mobile agent research.

1.4.1 Performance Measurement and Benchmarking

Some work has been done to measure and compare existing mobile agent systems. Dikaiakos et al. [16] define some micro-benchmarks to evaluate certain working processes of a mobile agent system, such as agent creation, messaging, and agent roaming. Silva et al. [54] compare eight mobile agent systems. Their results show the influence of several factors, e.g. the number of agent servers to be visited on one tour, the influence of the agent's size and the influence of class caching on the performance of mobile agents.

1.4.2 Performance Tuning

A few mobile agent systems have been explored regarding migration performance, e.g. [30]. Braun [7] removes drawbacks of todays mobile agents by using sophisticated migration strategies. He developed a new migration model called *Kalong*, which provides a flexible way to migrate mobile agents. "Using Kalong, a migration is no longer a monolithic transmission of code and data. It is possible to send only those pieces of code and data that are used at the next destination platform with high probability." [7], p. i. Measurements were executed to show that *Kalong* improves the migration performance compared to other migration techniques.

Knabe [32] deals with performance tuning of mobile agent systems. He describes mechanisms to improve the transmission of mobile agents. These mechanisms allow for the transmission of pro-

grams in different representations (source code, byte code or machine code). This shall help to save compile effort. Furthermore, Knabe decreases the volume of the program code which has to be transported by sending only code which does not belong to default libraries of the programming language or of the agent server. Finally, he recommends lazy compilation to only compile code which will actually be used. He implemented his approach using the programming language *Facile* and evaluated it by experiments.

Hohl et al. [24] propose code servers within their agent platform *Mole* to increase the performance of class loading. Code servers are specialised agent servers, which are located closer to the destination server than the agent's home server. Thus, if agent classes have to be loaded, the code is sent from the nearest code server instead of the home server. Each agent server has to know the set of code servers nearby, the distance (function of network latency and network throughput) and managed classes at the code servers. To further increase performance, Hohl et al. recommend to load classes in advance if they will likely be used later.

Soares and Silva [55] propose hierarchic code servers. Agent code which shall be loaded is first searched at its current location, then at its last location, at its home location, and finally, if necessary at further code servers.

Publications concerning the mobile agent platform *MAP* (see [50]) focus on the guarantee of quality of service agreements. Their concepts are demonstrated with tunnel agents for the management of IP/RSVP networks. Mobile agents are used as management units.

1.4.3 Performance Modelling

So far, performance modelling of mobile agent systems has been investigated rarely. Most performance analyses have been used to show that the application of mobile agents results in lower network cost than the traditional client/server approach. Important papers are due to Vigna [59], Carzaniga et al. [12], Strasser and Schwehm [56], and Iqbal et al. [28]. The basic idea is that it is more efficient to send a small piece of program code to a remote server to process a huge amount of data instead of transferring the data to the location of the program code. To prove this hypodissertation, several researchers built mathematical models, which compare the network load of mobile agents with remote procedure calls related to certain application domains. These models shall help to decide which paradigm to use with a concrete application. The models are analytical ones and quite static. The underlying assumptions usually do not reflect the dynamic character of mobile agent applications.

There exist a few simulation environments for agent systems, which usually focus on the modelling of multiagent systems. *Swarm* [39] is a wide spread environment for the simulation of multiagent systems. Originally, it was developed by the Santa Fe Institute. Agents are modelled by "Swarm" objects which can be hierarchically combined. Models have to be implemented using the programming language Objective C. *SeSAm* [52], [31] is a simulation environment for multiagent systems on a higher abstraction level. It is based on individual simulation of usually intelligent agents. Typically, *SeSAm* is used to model agents in biological applications. Further simulation environments for multiagent systems are, e.g. *PECS*, *SDML* and, *AgentSheets*. Kluegl [31], pp. 189 - 192 gives a good overview.

1.5 Outline

This dissertation is organised as follows: Chapter 2 deals with the central aspect of capacity planning, the process of performance modelling. It presents a new approach to model mobile agent systems by integrating real agent's program code into simulation. To demonstrate the applicability of this technique, the simulation environment *JaDEMAS* has been developed. Chapter 3 and chapter 4 deal with the subject of increasing efficiency of simulations of large systems. Chapter 3 describes existing approaches to increase model efficiency and shows their deficits in the context of the simulation of mobile agent systems. Chapter 4 presents new approaches which do increase the efficiency of simulation models of mobile agent systems. Chapter 5 explains the methodology developed for capacity planning, including measurement of input parameters and output values, performance evaluation by simulation and dimensioning of future systems. Chapter 6 demonstrates the applicability of the developed approaches by a realistic case study. Finally, chapter 7 summarises the most important results of this dissertation and gives an outlook into future research.

2 Simulation of Mobile Agent Systems

Performance modelling is a central activity in the capacity planning process: Performance models are used to evaluate future systems, i.e. to examine if specified quality of service requirements are fulfilled in the planned system. Furthermore, performance models are used to analyse what-if-scenarios to find a satisfying future system configuration. They can as well be used to analyse bottlenecks or unexplainable phenomenons in existing systems.

According to the goal of this dissertation, this chapter describes a concept and a model environment to easy integrate performance modelling into the development of a mobile agent system. Generally, several modelling paradigms can be used. Because of the special characteristics of mobile agent systems and because of the objectives of this work, simulation has been chosen as modelling technique. The basic idea is to directly transfer the program code of real agents into simulation. Additional parameters for the performance models are a specification of the infrastructure of the planned agent system, i.e. agent servers and network links which connect those servers, and service amounts of agents at agent servers. Chapter 5 describes how to obtain these parameters.

Hence, a developer[1] of a mobile agent system can evaluate the performance during system and code development with manageable additional effort. He is able to build the performance models himself, even if he is not a specialist in performance modelling.

This chapter is organised as follows: section 2.1 gives an overview of typical scenarios which can be modelled. Section 2.2 to section 2.3 sketch the modelling method and its implementation. Section 2.4 describes the modelling environment. Finally, in section 2.5 the new modelling concept and environment are demonstrated with an example. Section 2.6 discusses the portability of the approach to arbitrary mobile agent systems, section 2.7 gives a summary.

---

1. To simplify matters, the system developer is written about as male person. Of course, she can be as well a female character.

2.1 Model Scenarios

Originally, the paradigm of mobile agents has been developed for the implementation of distributed applications in the Internet. However, for security reasons, its applicability might be limited there. Beside all security precaution of the mobile agent software, operators will hardly risk access to their resources by foreign agents which could be disguised trojaners or another type of malicious code. Hence, it can rather be assumed that mobile agent systems will actually be applied in Intranets where system operators and agent developers are from the same organisation.

Hence, a modelling environment is built for the analysis of mobile agent systems which implement intranet applications. A single network access point is assumed (mobile agents' home server). Mobile agents are sent out from this home server and finally return there. Figure 2-1 shows a typical scenario for the developer of a mobile agent system. Here, users submit their requests via a web form at the mobile agent home server. Mobile agents are sent out to perform service according to the user requests. At each agent server they contact service providing, stationary agents. Only stationary agents have access to system resources which are necessary to fulfil services. Finally, the mobile agents return home and the results are delivered to the users.

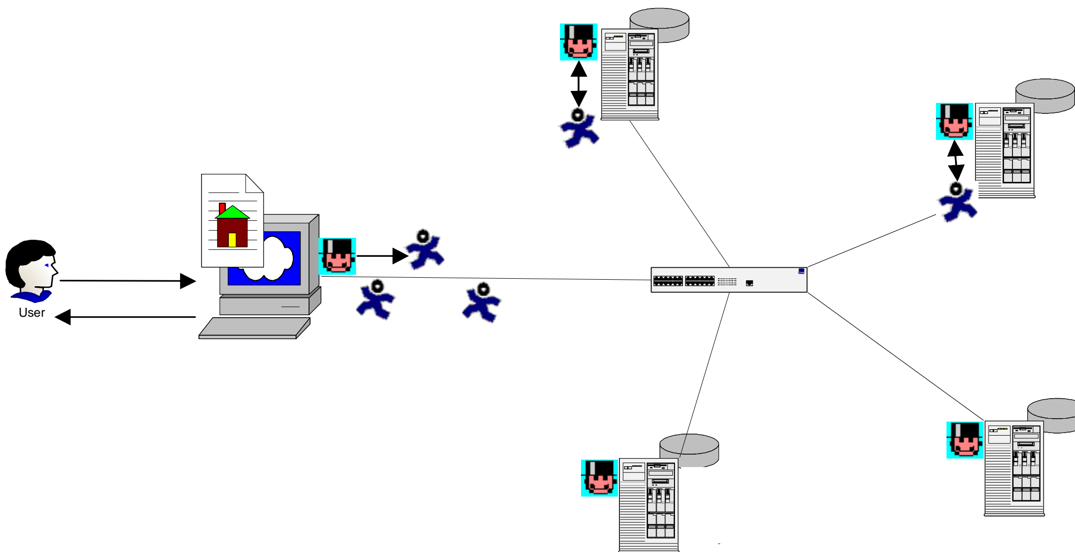

Figure 2-1: **Typical model scenario**

The process of communication between agents in this scenario is always the same. A mobile agent arrives at an agent server and sends a message to a stationary service agent. If the service agent is prepared to handle the message it fulfils the service requested by the message and sends the result back to the mobile agent. This process allows for the development of building blocks concerning agent servers, agents and their communication.

2.2 The Modelling Method

The modelling method is determined by certain requirements which are set for performance analysis. Furthermore, system developers shall be able to analyse and predict performance issues during the implementation of the system.

2.2.1 Requirements for Performance Modelling

The requirements for performance modelling of mobile agent systems are partly due to the characteristics of mobile agent systems, partly they are caused by the demands of this dissertation. One major objective of is to provide methods to make modelling as easy as possible for the developer of a mobile agent system. The system developer (modeller) should evaluate the performance during system development, i.e., he should be able to build the performance models himself. Also, results of the model experiments should be as meaningful as possible. Hence, the following requirements are specified:

1. The modeller should not have to accomplish complex workload tests or benchmarks to evaluate the system or to gain input parameters for performance models; it should be easy to measure necessary input parameters. Chapter 5 describes the fulfilment of this requirement in more detail.
2. Knowledge about stochastic characteristics concerning agent's behaviour inside the mobile agent system should not be necessary. Particularly, the transfer rates of mobile agents from one server to another or their visit counts at a server need not be known. Besides, these values are difficult to calculate because of the mobile agents autonomy to select their route.
3. Arbitrary distributions of arrival rates of agents and service times should be usable.
4. The analysis of the performance results should include transient and steady state analysis. Point estimators as well as interval estimators should be provided. Beyond the mean value, the second central moment and histograms of performance values should be output. In case of transient analysis the variation of performance values along the time axis has to be observable.

2.2.2 The Modelling Paradigm

One of the first decisions in performance modelling concerns the model paradigm. In general, it has to be decided whether to use mathematical or simulation models. Mathematical modelling requires high abstraction and knowledge about stochastic characteristics of the modelled processes. Simulation allows for modelling at a lower abstraction level, i.e. usually, simulated systems are modelled in much more detail and closer to the real system. I appears that some of the input parameters for a mathematical model are results of a more detailed simulation model. But, obtaining input parameters for simulation with sufficient level of detail is a not neglectable problem. Nevertheless, there are major arguments to use simulation models for mobile agent systems:

- This dissertation intends to provide methods and tools to evaluate the performance of the mobile agent system at a time when (maybe first versions of) the agents are programmed. Hence, the behaviour of the agents is sufficiently described by their program code. By using the same programming language for simulation and programming of agents it is easy to integrate agent code directly into simulation. Thus, the overhead for performance analysis is reduced immensely with simulation compared to mathematical modelling. Agent code can be exchanged between the simulation model and the real system with only minor modifications.
- Because of the mobile agent's autonomy, the agent's behaviour can depend on the current system state. Such systems can hardly be modelled mathematically.

Figure 2-2 illustrates the concept for the development of performance models. A simulation environment, called *JaDEMAS* (JavaDEMOS for Mobile Agent Systems), has been developed to simulate mobile agent systems using the simulation package *JavaDEMOS*. *JaDEMAS* fulfils the requirements specified in section 2.2.1.

The infrastructure of the agent system is specified by the modeller and it is automatically transformed to model components. *JaDEMAS* contains a workload generator which reads binary agent program code and delivers the stationary agents to their specified home servers in the model. Furthermore, it generates the system workload by means of mobile agents which are generated at their home server with a specified arrival rate. The agents then behave in the model according to their program code.

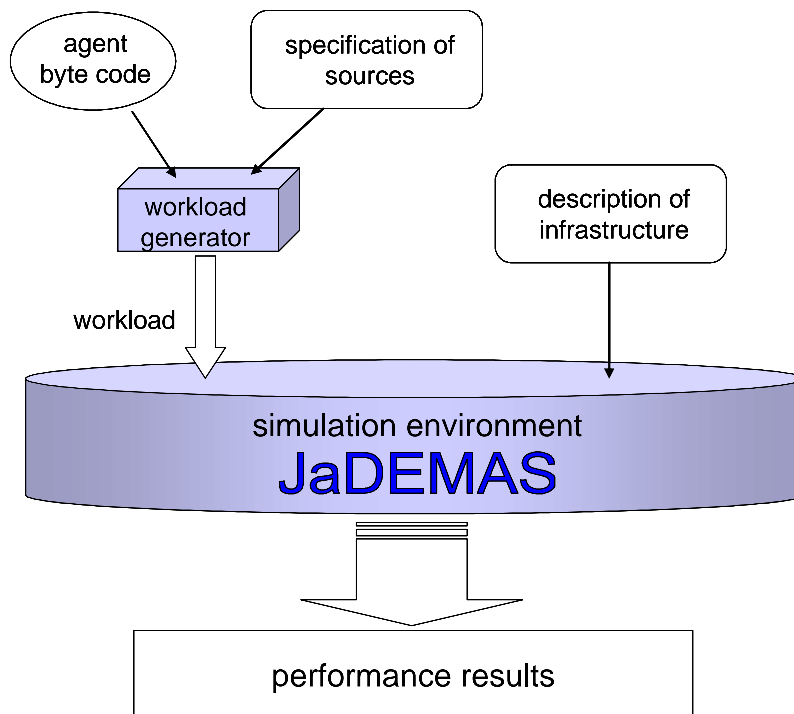

Figure 2-2: **Concept of agent system simulation model**

Finally, *JaDEMAS* allows for several performance analyses by its meaningful result values.

2.3 Implementation of the Modelling Method

In general, the developed modelling concepts can be applied to any mobile agent system. The problem is that, so far, standardisation of concepts and methods for mobile agent systems is not highly developed. "Except for two systems, Aglets and Grashopper, which support the MASIF migration protocol proposed as OMG standard [...] it is virtually impossible to make two systems interoperable. [...] However, even this systems are not willingly used, because of their complexity and size." [8].

Hence, to demonstrate the applicability, one mobile agent system has been selected for the implementation of the modelling concepts. The mobile agent system *Tracy* was chosen. *Tracy* imple-

ments *JAM* [8], a model for agencies which addresses high compatibility with other mobile agent systems. Also, it is of manageable complexity and size.

Furthermore, a simulation environment had to be chosen. *JavaDEMOS* was selected, a programming language with a graphical user interface, especially designed for discrete event simulation.

2.3.1 Tracy

JaDEMAS is designed for *Tracy* version 0.61. If *Tracy* is mentioned afterwards its features always refer to this version.

Tracy is a general-purpose mobile agent system. It was developed at the Friedrich Schiller University of Jena by P. Braun, J. Eismann and C. Erfurth [9]. It distinguishes between trusted stationary agents, that have permission to access the local file system or open network connections, and untrusted mobile agents, that do not have these rights. Stationary agents welcome mobile ones and provide system services. Only mobile agents can migrate between agent servers. At their creation, agents get a unique name, so-called first name. Agents can be created either by the agent server or another agent.

Migration Process

Mobile agents decide on their own when and where to migrate. The *Tracy* software handles the transfer of the mobile agent from one server to another. Therefor, agent's classes and its state are marshalled to be sent and demarshalled at the destination server to be executed there. *Tracy* only supports weak migration, i.e. not the complete agent state is moved. Thus, agent code cannot be interrupted at one server at an arbitrary break point and be continued at another server exactly at this point.

Tracy supports four different migration strategies:

- *push-all-to-next*: The whole mobile agent, i.e. all its classes and its state is transferred from the current agent server to another at each migration.
- *pull-per-unit*: Only mobile agent's state is transferred to the destination server. Necessary agent classes are requested by the destination server, and thus transferred, only if they are needed there and if they do not already exist at the server.
- *pull-all-units*: Like pull-per-unit, but agent classes are generally transferred at once even if they already exist at the destination server.
- *pull-all-to-all*: all agent classes are initially transmitted to all servers the agent is going to visit. This implies, that the agent knows its destination servers from the beginning of its life time.

JaDEMAS is designed for the push-all-to-next strategy.

Tracy provides two transmission strategies through the network: Java's *RMI* and an own agent transfer protocol *SATP* (Simple Agent Transfer Protocol). Both mechanisms use TCP/IP as transport protocol.

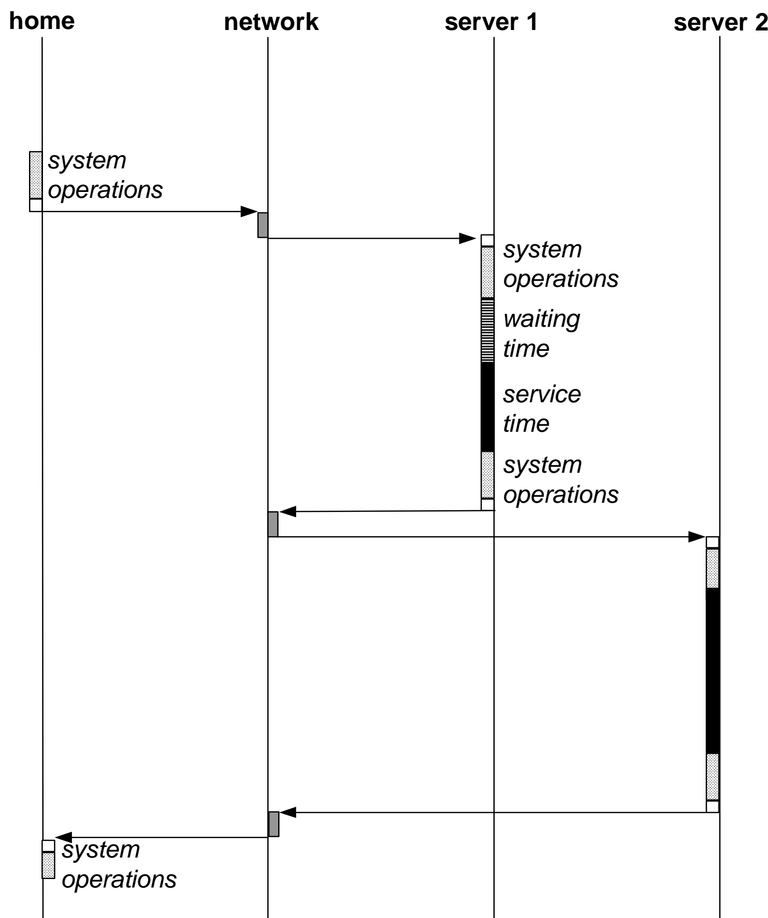

Figure 2-3: **Process of agent trip (example: mobile agent visiting two servers)**

Figure 2-3 illustrates the process of mobile agent execution and migration in *Tracy*, which is typical of a simulation with *JaDEMAS*. First, the mobile agent executes some initial operations at the home platform. Before being sent to another server the agent code and data are marshalled (serialised). Some more time will be consumed by further system operations before the agent can migrate. Then it is transmitted over the network to the next server, where it must first be deserialised (after initial system operations). Afterwards, the agent' code is executed. If the mobile agent requests a stationary service agent (as usual) it possibly has to wait until this service agent is idle. Finally, the mobile agent is served by the service agent before it is sent to the next server.

Communication Mechanisms

Tracy provides two means of communication for agents on the same agent server: agents can send asynchronous messages or use a blackboard. To communicate with other agents on remote agent servers, an agent must migrate to this server.

When an agent is started at an agent server, its own mailbox is created. The mailbox is assigned to the agent as long as it resides at the server and it is deleted when the agent dies or leaves the server. Messages are stored until the receiving agent takes it out of the mailbox in an first-come-first-serve manner. Furthermore, at each agent server there exists one blackboard where agents can put messages. The blackboard's structure is similar to file systems in UNIX. [58]: "There are

nodes representing directories and leaves, which are comparable to files, Both, files and directories maintain read and write permissions. An agent can be granted read or write access to a specific blackboard by the owner of this entry“.

For further details concerning *Tracy* see [8], [9] and [10].

Tracy was chosen because its agent types, communication methods and its whole design is straight forward and well manageable. Furthermore, it includes reasonable mechanisms not to waist performance, as e.g. passive waiting of agents, and methods to monitor some performance values. The fact that *Tracy* agents are implemented in Java simplifies their integration into the simulation environment of *JavaDEMOS*.

2.3.2 JavaDEMOS

DEMOS (Discrete Event Modelling On Simula) is a basic package for discrete event simulation. Originally, it was developed by G. M. Birtwistle as an extension to the simulation programming language *Simula*. *DEMOS* adds mechanisms to simplify discrete event simulation, as Simula on its own is a very complex language which is not always easy to handle. For further details see [4].

JavaDEMOS was implemented by O. Matthes in his diploma dissertation at the University of Essen in 1999 [40]. It transfers the concepts of DEMOS to the programming language *Java*. In addition, it contains a graphical front-end which permits the visualisation of a simulation run and which allows for basic interactions with the simulation system. The user can observe the current objects in the event list, statistical results as, e.g. the usage of a resource object, and the simulation trace. Simulation can run in whole, in single step mode or until reaching of a certain time or entity.

For a thorough description refer to the original *DEMOS* documentation, in particular to the DEMOS text book and the *DEMOS* reference manual both due to Gramme Birtwistle. These documents are available digital form from different sources, see e.g. [4]. *JavaDEMOS* versions of the classical examples are given in [41].

Users of *JavaDEMOS* are expected to be familiar with the principles of modelling and discrete event simulation including random number generation, basic concepts of discrete simulation like the event list, and evaluation of simulation runs to determine estimated mean values and confidence intervals. Additionally, users should be familiar with the building blocks of *JavaDEMOS*.

Entities and their Scheduling

In *JavaDEMOS* the basic concept is the entity. Entities implement behaviour patterns, may acquire and release resources, may wait until certain conditions are fulfilled, are able to interact with each other in a master/slave mode and can of course be scheduled in the event list. The global scheduling methods are `schedule()`, `hold()` and `passivate()`. Some important building blocks are as follows.

- Res (mutual exclusion synchronisation)
- Bin (producer/consumer synchronisation).
- WaitQ (master/slave synchronisation, including a queue for holding coopted entities)
- CondQ (waits until a given condition is fulfilled, avoiding the active wait for resources)

Random Numbers and random variates

JavaDEMOS offers the random number generators which are used to generate variates of types constant, empirical, Erlang, negative exponential, normal, uniform, Poisson and Bernoulli. Especially mentionable is the method of generating well spread seeds to generate (quasi-) independent streams of random numbers. The user can choose between the original DEMOS basic random number generator and the generator „MRG32k3a“by Lecture [38] with period length $\approx 2^{191}$.

Reporting

JavaDEMOS contains reporting aids like class `Report`. On generation, each facility object is entered into a special Report reserved for its type. There are data collection devices like `Count` (incidences), `Tally` (time independent data), `Accumulate` (time dependent data), and `Regression` (for linear regressions). Another one is class `Histogram` (`Tally` plus a bar chart) which extends class `Tally`.

An additional feature of *JavaDEMOS* is the observation of time dependent behaviour of some performance measures. Furthermore, features for an extended output analysis have been developed. There are the classes `BatchMeans` and `ConfidenceInterval` for the analysis of interval estimates. For further details concerning *JavaDEMOS* see [26].

Although *JavaDEMOS* is a powerful simulation package, its features and building blocks are not sufficient for the purpose of this dissertation, to easily model mobile agent systems. Thus, *JavaDEMOS* is extended to easily integrate real *Tracy* agent code and parameters of the infrastructure of an agent system into *JavaDEMOS* simulation models. *JavaDEMOS* is extended to *JaDEMAS.*

2.4 JaDEMAS: A Simulation Environment for Tracy Agent Systems

To simulate *Tracy* agent systems, about 50 additional *JavaDEMOS* classes have been developed. With this new classes it is possible to analyse the performance of a *Tracy* mobile agent system during the implementation process of the agents. Only agent code and some additional parameters are necessary for simulation, no physical agent system has to be installed. The agent system developer can analyse several configurations of the system by easily varying server and network capacities in the simulation model. So he can find the best realisation for the real agent system.

JaDEMAS allows for the analysis of specific performance metrics like, e.g. agents’ round trip times, utilisation of agent servers, system throughput, etc.. The new classes and methods have been implemented corresponding to the class hierarchy in *Tracy.* Thus, in simulation the *JaDEMAS* classes replace the real agent system. Thus, the whole agent system runs in form of a simulation at a single computer. A graphical user interface (GUI) has been developed to facilitate enter of input parameters of *Tracy* simulation models.

The following sections give an overview over important components of a mobile agent system and how they were implemented in *JaDEMAS.* Figure 2-4 describes the symbols used to describe the main parts of the simulation system in the following sections. The symbols refer to [48], but are slightly modified and extended.

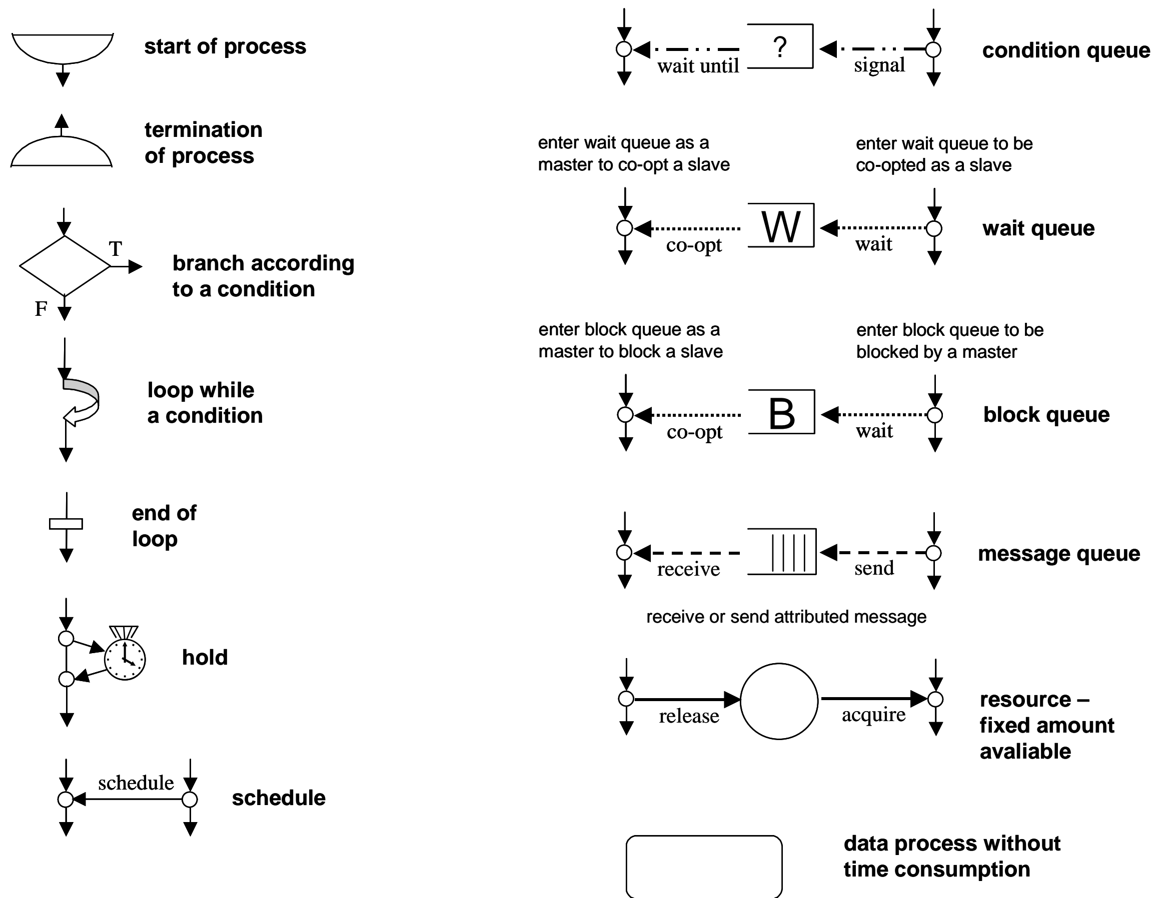

Figure 2-4: **Menu of diagram symbols (according to [48] with modifications)**

2.4.1 Agent Servers

An agent server is a place where all types of agents can reside and to which mobile agents can migrate.

The agent server class models accesses to server resources. It is assumed that a mobile agent system consists of dedicated agent servers, i.e. servers are either specialised in providing CPU power (compute servers) or they are specialised in transferring data to/from I/O devices (file servers). Thus, it is adequate to model an agent server's main function by a corresponding queuing station (for either CPU or I/O device).

File Servers

Requests at I/O devices are - with a high-level point of view - served in FIFO (first-in-first-out) order. I/O devices are not modelled in detail with regard to the limited possibilities to get detailed input parameters for the models from measurement. Even if a file server consists of multiple I/O devices it will be modelled as a station with a single service unit to keep models manageable. Otherwise, models had to consider hardly detectable, performance relevant issues, e.g., the spreading of data records to several devices to determine which requests can be handled in paral-

lel. Hence, file servers are modelled by FIFO servers with a single processing unit. The *JavaDEMOS* class `Res` is used to model the I/O device (see figure 2-5). In general, only stationary agents have access to server resources. Thus, these agents acquire the I/O device and release it after their service time elapsed. If an agent tries to acquire the device while it is in use the agent is put into the device's FIFO queue until it is idle again.

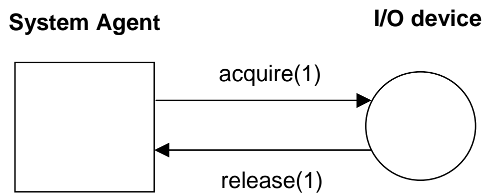

Figure 2-5: **Simulation of a file server**

Section 4.3 shows an alternative method to model FIFO servers, and thus I/O devices, more efficiently.

Compute Servers

The *JaDEMAS* class `AgentServer` contains an object for CPUs in round robin mode (new class *CPU*). All jobs at the CPU are served in a cyclic fashion and are given a fixed piece of processing time ("time slice") one after another. A job has finished if the sum of slices it got accords to its service time. There is no separate wait queue, waiting time arises when other jobs are served within the CPU.

Figure 2-6 describes the simulation of resource access at a compute server. A CPU can only be allocated by a new developed object called `Job`. Thus, if a stationary agent wants to access the CPU it has to create a corresponding job instance. By generating a job the agent is blocked as long as the job is served by the CPU. This problem led to the development of the new *JavaDEMOS* object *BlockQ*. It provides one queue for master and one for slaves. Slave objects waiting in the queue are blocked until they are co-opted by a master process. In a sense, *BlockQ* is very similar to *JavaDEMOS*' class *WaitQ*. The main difference is that the *BlockQ* objects are not presented in *JavaDEMOS* result reports. The object *Job* is a building block which interacts with the CPU on behalf of its creator. This provides only little additional simulation code in the creator object to simulate CPU service. A job puts itself into the CPU's *WaitQ* where jobs wait for being scheduled. The CPU co-opts one job after another in round robin mode for a specified time slice, each. After one slice is elapsed the job is rescheduled and puts itself again into the *WaitQ* until its service amount is fulfilled. When a job is served pursuant to its service amount it unblocks and reschedules its creator object (which is usually a system agent).

Section 4.5 shows an alternative method to model round robin servers, and thus compute servers, more efficiently.

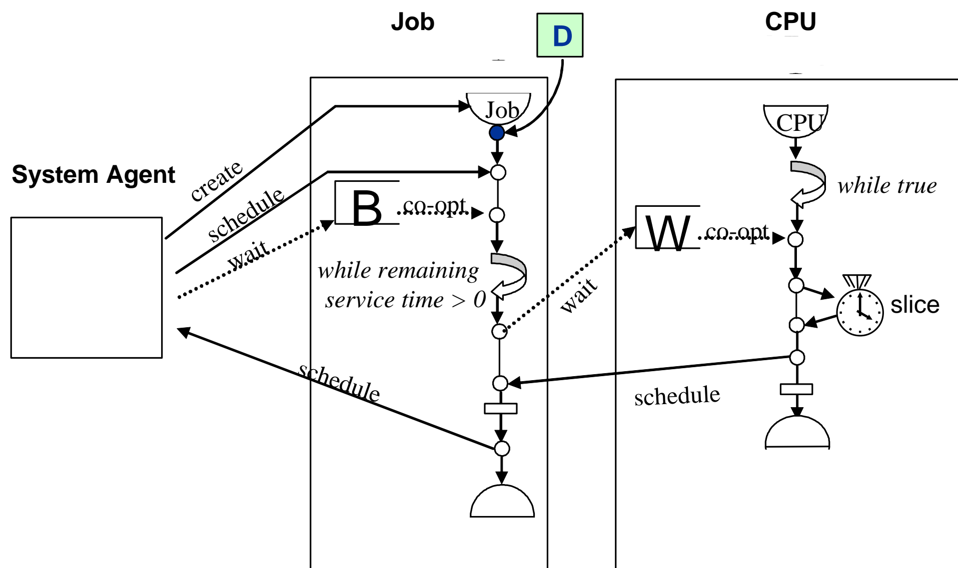

Figure 2-6: **Simulation of a CPU**

Multi Processors

To simulate multi processors, multiple instances of CPUs are created at a server. The new developed class `Dispatcher` is used to allocate a CPU to a job. When a job is created by an agent it calls the dispatcher to allocate a CPU. The process described in figure 2-7 takes place at job creation, see the dot marked "D" in figure 2-6. The dispatcher checks the number of jobs which are currently residing at the CPUs and selects the CPU with the lowest number of jobs. So, long term load balancing is provided. After a CPU is allocated, activities between job and the CPU proceeds as described in figure 2-6.

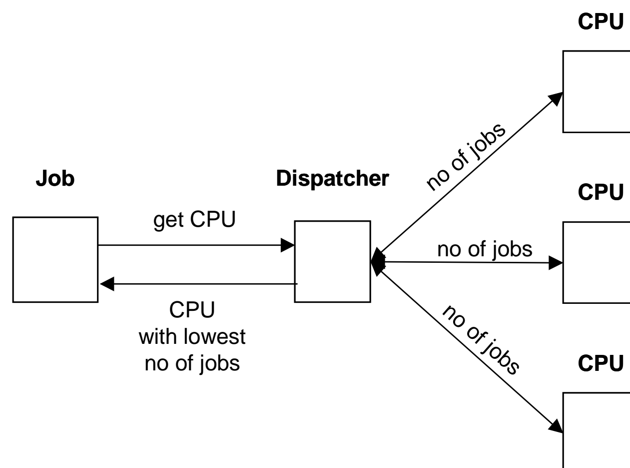

Figure 2-7: **Modelling of multi processors**

System and Migration Overhead

Experiments with *Tracy* have shown that it is feasible to distinguish three types of workload in the models. Workload arises when an agent consumes resources in order to provide its primary service for the user, e.g. the access to a database for information retrieval. This workload shall be called *user load*. Another type of workload arises at the agent system to manage agents' activities such as communication, calling of system routines etc. As this type of workload can be interpreted as overhead of the primary agent service, it shall be called *system overhead*. The third type of workload describes resource consumption for initialising agents and for all activities resulting from migration of mobile agents. This workload is called *migration overhead*. Figure 2-8 shows the structure of workload.

user load	`"handler.resolve(query);"`
system overhead	overhead by application activities, e.g. sending of messages
migration overhead	(de)-marshalling, access to network interface, class loading, ...

Figure 2-8: **Structure of work load**

The former describe server types (file and compute server) model resources which are accessed by user load. System and migration overhead are modelled by separate load dependent FIFO servers. These resources are primary used for model calibration. Therefor, the modeller has to specify the service rates which the FIFO servers provide with a certain number of residing agents. The service time of an agent at such a resource is calculated by its service amount divided by the service rate which depends on the number of agents which are currently served by the resource.

With *Tracy* version 0.61 it can be observed that successive mobile agents sometimes are delayed in the migration process, independent of the current utilisation of agent servers. It can be assumed that deadlock situations arise which can be resolved afterwards. Therefor, *JaDEMAS* provides a so called *ghost delay*. The modeller can specify phases where mobile agents are additionally delayed. The mean number of agents within and without a ghost delay phase and the mean additional delay in a ghost delay phase have to be specified. Duration of the phases (in number of mobile agents) and additional migration delays are drawn from negative exponential distributions.

2.4.2 Communication Mechanisms

Beyond the modelling of server resources, the agent server class contains objects which support agent communication. In *Tracy*, agents can only communicate directly with each other if they are located at the same agent server. *Tracy*'s communication mechanism by sending asynchronous messages is implemented in *JaDEMAS*. Communication via blackboards is implemented only rudimentary. Figure 2-9 shows the interaction of a mobile and a system agent at an agent server.

Messages are exchanged via message queues (mailboxes). When an agent is started at a server (or arrives at a server after migration) its own message queue is generated. Messages from other agents are stored in the message queue. In this example, mobile agent *QueryAgent* sends a message into the message queue of system agent *ServiceAgent*. *ServiceAgent* is idle and gets a signal about the new message arrived. It reads the request in the message and fulfils it by using the server resource. When the service is completed, *ServiceAgent* generates a reply message with the service result, puts it into the message queue of *QueryAgent* and sends a signal to the waiting *QueryAgent* that the result message has arrived.

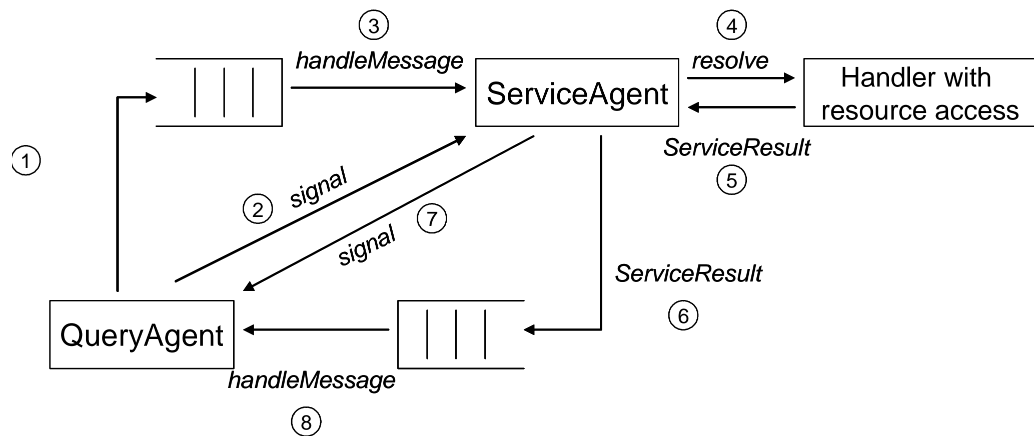

Figure 2-9: **Serving of mobile agents by system agents**

In the `AgentServer` class there is a queue where agents can reside if they are waiting for messages. In *Tracy*, this waiting process is passive, i.e. waiting agents free all their resources which they have occupied so far. If the agent is in a waiting state, i.e. it resides in the agent wait queue, the arrival of messages is signalled to the agent thus, it can get active again and handle one message after another. The *JavaDEMOS* "condition queue" (`CondQ`) object is used for implementing the agent wait queue.

2.4.3 Agents

All agents are active objects and are separated into gateway, mobile and system agents (corresponding to *Tracy*). They all share the same methods for communication and have all basic attributes like identifier, current location and home location (where agents are first started). Since gateway and system agents cannot move, their current location is always the same as their home location. Mobile agents have special methods providing their mobility.

To integrate real agents in simulation, their program code will be directly put into *JaDEMAS*. Figure 2-10 shows an example. The real agents `QueryAgent` and `ServiceAgent` are transferred from the real system into *JaDEMAS*. There, they use *JaDEMAS*' instead of *Tracy*'s classes.

More precisely, Figure 2-11 illustrates how agents interact and drive simulation in *JaDEMAS*. Classes `Agent`, `MobileAgent` and `SystemAgent` are *JaDEMAS* classes which model the behaviour of the corresponding *Tracy* classes. Again, *QueryAgent* and *ServiceAgent* are agents from a real system.

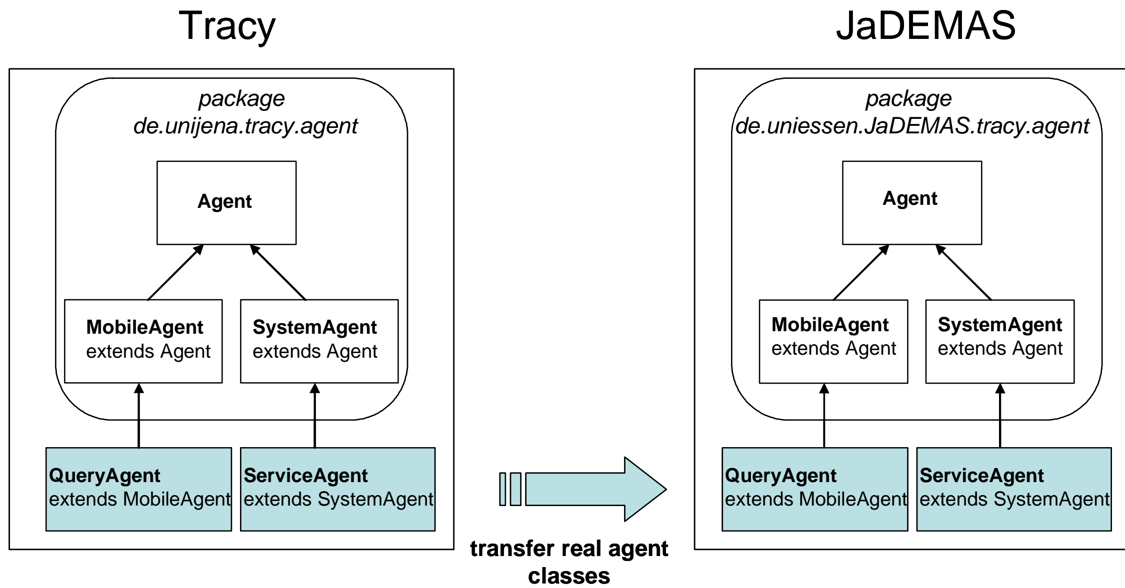

Figure 2-10: **Integration of real agent classes in JaDEMAS**

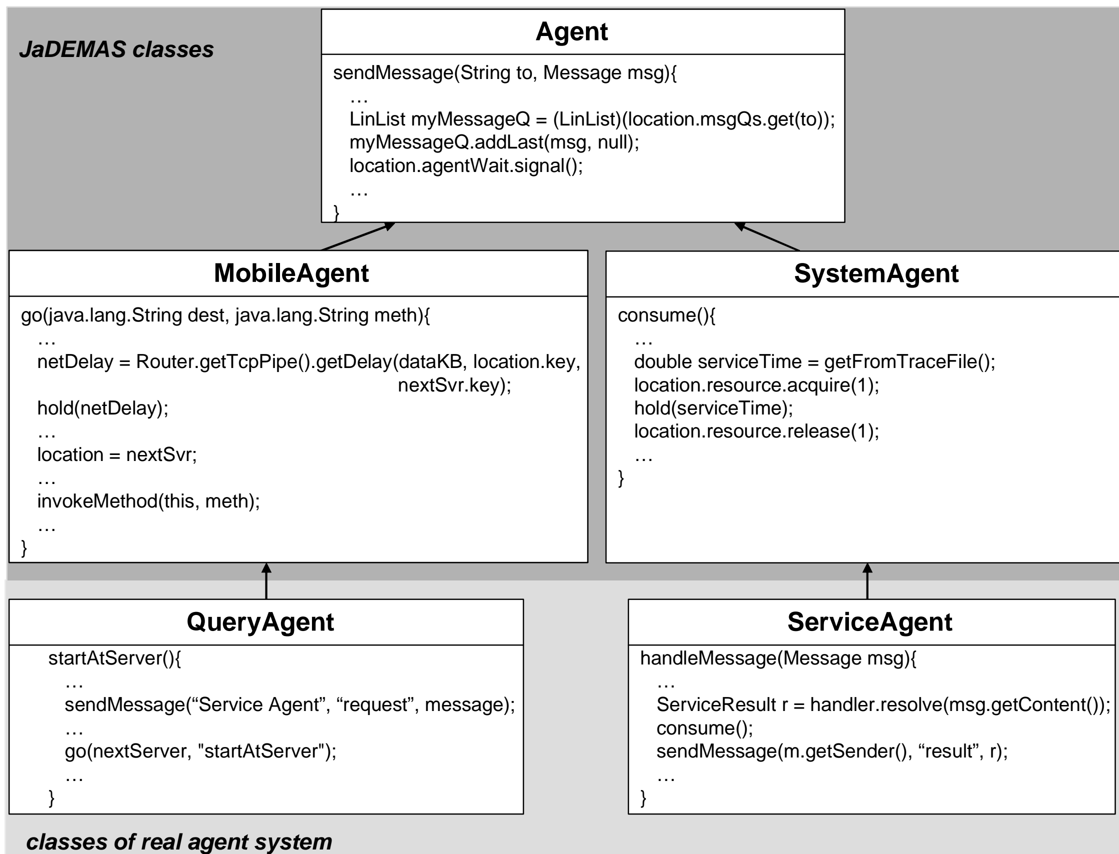

Figure 2-11: **Inheritance hierarchy of agents in JaDEMAS with code fragments of JavaDEMOS**

As soon as arrived at an agent server, `QueryAgent` sends a message to `ServiceAgent` to request a certain service. This is implemented by the method `sendMessage` of the super class `Agent`. This method puts the message into the message queue of the receiving agent and sends a signal to this agent, that a new message has arrived (also, compare Figure 2-9). This invokes the abstract method `handleMessage` which has to be specified by each real agent. If `ServiceAgent` gets a request message it fulfils the service, consumes the resulting service time and sends an answer message back to the mobile agent. The time consumption is implemented by the method `consume` of the super class `SystemAgent`. The service time can be specified directly or can be read from a trace file. Then, the system resource of the server is occupied for the service time.

After reception of the request result, `QueryAgent` decides which server it wants to visit next and calls the method `go` of the its super class `MobileAgent`. Method `go` determines the route between the agents current location and the next server and interrupts the mobile agent for the network delay resulting from the properties of the links on the route and the data volume of the mobile agent. The interruption is done by calling the *JavaDEMOS* method `hold`. After the network delay has elapsed, the specified method is invoked at the next agent server. Figure 2-12 describes the underlying model in graphical notation.

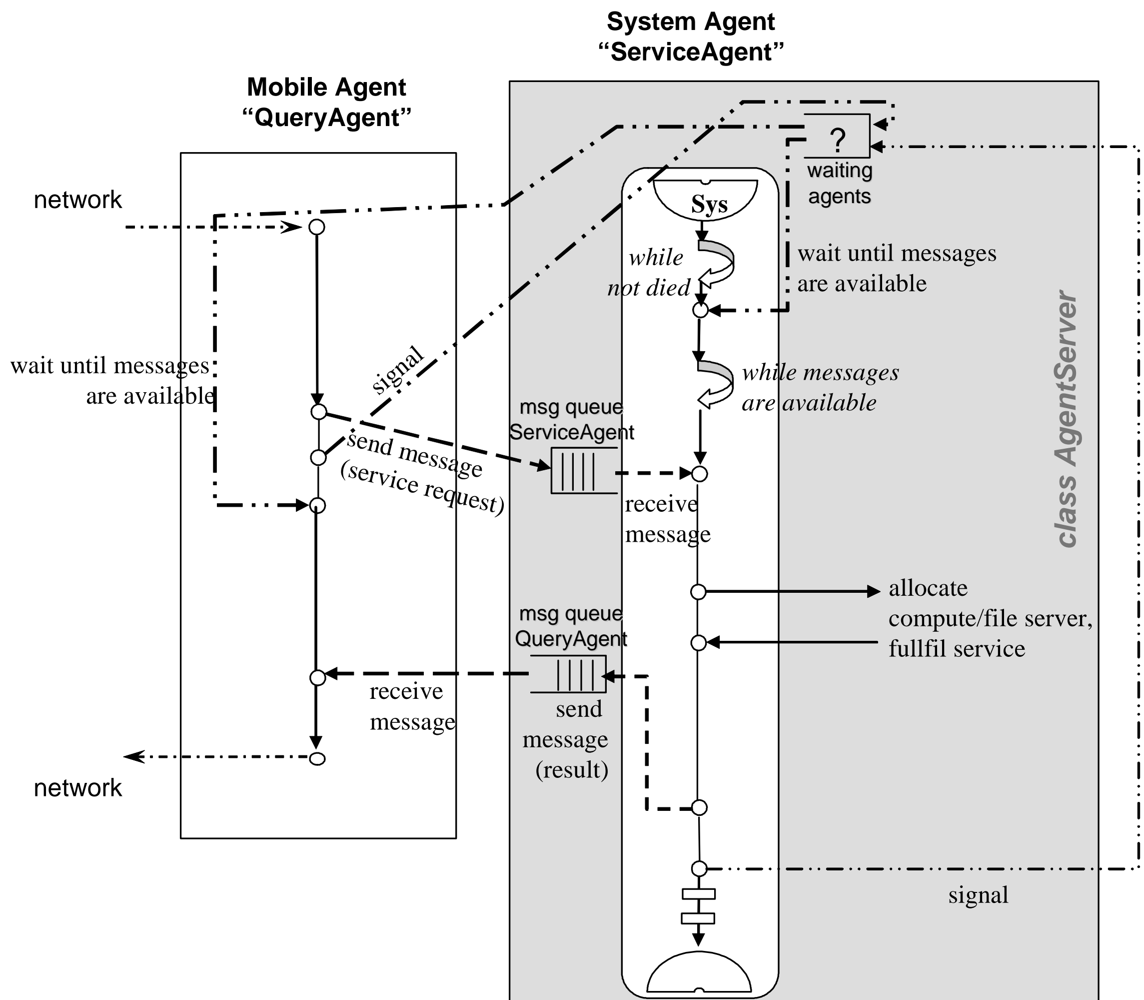

Figure 2-12: **Simulation of interaction between mobile and service agent**

2.4.4 Network Links

With *JaDEMAS*, the modeller can chose between two types of network models. A simple one is provided for a rough approximation of network delays. This type is recommended if detailed network parameters are unknown. The other type models a TCP pipe between agent servers. Both types are mathematical models.

Figure 2-13 shows the general modelling of a mobile agent's transport from one server to another: After the work is done at an agent server, the mobile agent gets the network delay from the mathematical network model. Then, it interrupts itself for the network delay. After the network delay has elapsed, the mobile agent arrives at the next server.

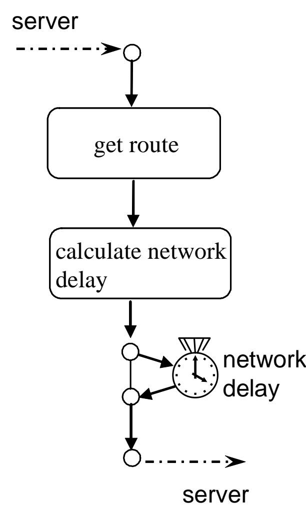

Figure 2-13: **Simulation of network links**

Simple Network Model

With the simple network model a network link is only characterised by its bandwidth. A router class provides a routing table that specifies the connectivity of agent servers including the bandwidth of the connecting links.

A simple analytic model then computes the network delay consumed by the agent on its way to the next server:

$$d_{net} = \frac{\text{agent data volume}}{\text{link bandwidth}} \qquad \text{(Eq. 1)}$$

This simple model is sufficient to roughly model network delays, as it is assumed that nowadays, most network links are point-to-point links or that the bandwidth is high enough to transport the quite small mobile agents (between Kilobytes and maximal a few Megabytes) without significant detention. Because of the small data volume of mobile agents, network links should not be the bottleneck in an agent system, thus often, it is not necessary to model them more detailed.

Extended Network Model

For more detailed modelling, *JaDEMAS* provides an extended network model. *Tracy* agents are transported through the network via the TCP protocol, thus, a TCP pipe is mathematically modelled.

The network delay is calculated according to the following algorithm:

First, the shortest path (minimum hop route) to the destination server is determined by Dijkstra's algorithm [15]. The effective bandwidth on this route is defined as

$$X = min(X_b, X_t) \quad \text{(Eq. 2)}$$

where

X_b = theoretical bandwidth of the bottleneck link on the route, X_b can be decreased by a ratio of bandwidth which cannot be used by application due to background load or overhead (e. g. encryption).

$$X_t = \frac{W_{avg}}{rtt} \quad \text{(Eq. 3)}$$

where

W_{avg} = average window size of the TCP protocol and rtt = mean round trip time of TCP segments from source to destination server. rtt is estimated as the sum of round trip times of the single links on the route.

Next, the payload $datavol$, i.e. data volume of the migrating agent, has to be increased by TCP and IP headers (20 Bytes, each). The number of headers is calculated considering the size of a TCP segment ($\leq 64KBytes$) and of an IP datagram ($\leq 1500Bytes$). Link layer protocols are not considered.

Furthermore, the modeller can specify an additional delay d_{add}, consisting of a delay on each node on the route and a delay for each agent migration caused by a DNS look up to find the IP address of the destination server. Both resource types, nodes and DNS server, are modelled as load dependent FIFO servers in *JaDEMAS*.

Finally, the network delay is calculated by

$$d_{net} = \frac{datavol}{X} + d_{add}. \quad \text{(Eq. 4)}$$

The extended network model is incorporated in a new *JavaDEMOS* network package [26].

2.4.5 Workload Generation

There exist special generator classes which generate model components for the system infrastructure, consisting of agent servers, network links between servers and sources (representing users) which generate the workload.

To generate workload in the mobile agent model, the agents' program codes are loaded. Several stationary agents are started once at their home locations. The same stationary agent can have several different home locations. With *JaDEMAS'* generator classes, mobile agents can be loaded and started at a single home location. It is assumed, that there exists a central agent system access point which is the home server of all mobile agents. The mobile agents are started at the home server by an adjustable number of sources (see figure 2-2). The arrival rates of mobile agents per source and the burstiness of the interarrival times (which is modelled by the coefficient of variation, c.o.v.[A]) have to be specified. Interarrival times with $\text{c.o.v.}[A] > 1$ are modelled by a Cox-2[1] distribution, $\text{c.o.v.}[A] = 1$ is modelled by a negative exponential distribution, and $\text{c.o.v.}([A] < 1)$ by an Erlang-k distribution. After being started at the home server, the mobile agents travel through the agent system (model) according to their program code.

Besides *JaDEMAS* generator classes, it is possible to import self developed classes to generate agents.

2.4.6 Input Parameters

For a simulation of an agent system the developer has to specify model input parameters, as there are

Workload description:

- Arrival rates and burstiness of requests and variation during simulation,
- agent classes (path) and their first names in the agent system,
- home server(s) per agent.
- Data volume of mobile agents,
- distribution of service times of agents at server resources.

Infrastructure of mobile agent system:

- Agent server's DNS names and resources as there are
    - type (file or compute server),
    - number of CPUs and time slice (constant value) in case of compute servers,

  Calibration parameters:
    - service rates for serving user load, system and migration overhead,
    - mean ghost migration delay and mean duration (in number of agents) of on and off phases of "ghost server",
    - fixed system overhead which arises per server when starting or migrating an agent.
    - Service amount for DNS look up and service rates of DNS server.
- Routing table which specifies the characteristics of links between servers. Simple model: bandwidth. Extended model: bandwidth, portion of bandwidth which cannot be used by application which is due to background load or overhead, TCP round trip time at each link, service rates of nodes on each link.

---

1. Assumption: both phases are evenly utilised, i. e., $\mu_2 = a_1 \cdot \mu_1$, where μ_1 and μ_2 are the service rates of the two phases and a_1 is the probability to pass through phase 2 after phase 1.

- When using an external generator and/or initialisation class: pathes to this classes and necessary parameters.

Output analysis[1]:

- In case of steady state analysis:
  - confidence level for the confidence interval,
  - threshold for the relative statistical error.
- In case of finite horizon analysis:
  - period of time to be modelled,
  - window size for moving average of round trip time, throughput and utilisation,
  - server which utilisation is evaluated along the time axis,
  - number of simulation run (simulation runs with different numbers get different seeds of random number generators),
  - parameters of histogram for analysis of round trip time.

Default values are proposed with the graphical user interface for simulating *Tracy* agents with *JavaDEMOS* whenever possible and sensible.

2.4.7 Modifications of Real Agent Code

Minor modifications have to be made to transfer real agent code into *JaDEMAS*:

- Imports of *Tracy* packages have to be changed to the corresponding *JaDEMAS* packages.
- Methods `startAgent` and `addAgent` got as additional parameter the DNS name of the agent's home server.
- The distributed agent system is simulated on a single computer, thus, if the modeller uses his own initialisation methods he has to take care that agents are started at each simulated agent server. In the real system, the initialisation methods are possibly executed identically at each agent server. Which means, if, e.g. the initial method in the real system contains the method `startAgent` once, it is executed at each of 10 agent servers when starting these servers. Thus, at each server a corresponding agent is running. In simulation, the method `startAgent` has to be called 10 times in the initialisation class (each time with the corresponding server name as additional parameter) to assure the same effect.
- Access to system functions which provide undesired values for simulation or which block simulation have to be exchanged, e.g.:
  - Calls to the real system time have to be changed to calls to the model time (in *JavaDEMOS*: `Scheduler.getClock()`).
  - Threads which wait for events inside the simulation have to be changed to simulation entities. Otherwise they can block simulation.
- The time consumption at server resources have to be added to the agent program code. It is assumed that primary system agents consume time at the server resources as mobile agents do not have access to the resources. Hence, in the code of the system agents, near the statements which initiate resource access, the method `consume()` or `con-`

1. About the meaning of parameters concerning the output analysis see section 2.4.8.

`sume(`<service time>`)` has to be called. With the first method, *JaDEMAS* assumes that their exists a trace file with service times to use one after another. This means, *JaDEMAS* supports trace driven simulation.

- If stationary agents access system functions or applications (data bases, user applications etc.) these must be accessible from the computer which runs the simulation. If this cannot be provided with the simulation computer, the modeller has to implement the access to remote computers where the applications run.

2.4.8 Output Analysis

JaDEMAS provides mechanisms to analyse the long term behaviour of a system in terms of steady state analysis. Furthermore, the dynamic behaviour of performance values along the time axis can be analysed for a specified time interval. This is called transient analysis, resp. finite horizon analysis.

Steady State Analysis

The steady state analysis is done via the batch means method ([29] page 432 et. seq.) for mobile agents' round trip times. Batch size is set to 5000 observations of round trip time. The first two batches are assumed to be in the transient phase, i.e. they are deleted. Results are gained according to the method of sequential simulation [47]: the simulation stops when the relative statistical error (ratio of the half-width of the confidence interval and the point estimate) is smaller or equal the given threshold. The model time at this event is not predictable.

The following performance results are provided with steady state analysis:

- confidence interval of the mobile agents' round trip times, mean value, variance, minimum and maximum,
- residence times of mobile agents at agent servers (mean value, variance, minimum and maximum),
- network delays of mobile agents,
- utilisation of agent server resources over the modelled time,
- system throughput (mobile agents per second) over the modelled time,
- length of waiting queues (mean value, variance, minimum and maximum).

Finite Horizon Analysis

In case of finite horizon analysis the system is simulated for a certain duration of time. It is assumed that the simulation starts at time 0.0 with an empty system. The variation of mobile agents' round trip times, the system throughput and the utilisation of a dedicated agent server is surveyed along the simulation time axis.

To compare results along the time axis it is necessary to gain result values at the same timestamps. This is not the case if, e. g., single round trip times are surveyed just at the return of mobile agents at the home server as they return at different timestamps in different model runs. Thus, results are averaged within moving time windows of fixed size. Further, it is advisable to run multiple replications of a model with different random number streams to get statistically significant results. The results should than be averaged over the replications. Summarising, results

are averaged per simulation run over a fixed window size and these mean values are again should be averaged over all replications. This method is not fully automatically supported by *JavaDEMOS*. By specifying the number of the particular simulation run *JaDEMAS* guarantees that random numbers are used which are different from the ones of the previous runs.

Results of the variation of the round trip times, system throughput and utilisation are stored in separate files per run. The modeller himself has to take care for the averaging of values of the replication runs.

Summarising, the following performance results are provided with finite horizon analysis:

- variation of mobile agents' round trip times along the time axis,
- histogram of round trip times,
- residence times of mobile agents at agent servers (mean value, variance, minimum and maximum),
- network delays of mobile agents,
- utilisation of resources of a dedicated agent server along the time axis,
- system throughput (mobile agents per second) along the time axis,
- length of waiting queues (mean value, variance, minimum and maximum),
- if desired, residence time at single servers, round trip time and network delay for single agents.

The evaluation of the round trip time by a histogram allows for the analysis of its relative frequency and empirical distribution. Thus, predictions concerning the fulfilment of typical service level agreements can be made, e.g. it can be tested if the round trip time of mobile agents is to x% lower than a specified value.

2.5 Example

This section shows an example for the illustration of the concepts described previously. A mobile agent system consisting of 20 agent servers (one home server and 19 remote servers) shall be analysed. There reside 4 system agents at each remote server which provide different calculation algorithms. Mobile agents travel through the agent system and request the calculation algorithms for certain computational functions.

2.5.1 Parameterising via the Graphical User Interface

The agent servers are all compute servers, i.e. they provide mainly CPU power for the agents. All servers have a single CPU with a time slice of 100 ms. Figure 2-14 describes the parameterisation of the infrastructure with the graphical user interface of *JaDEMAS*. No overhead is assumed at the servers.

Furthermore, the network links have to be specified by the routing table. This is a matrix in a simple text file. The indices of rows and columns correspond to the server numbers given by the order of the specified servers in figure 2-14. Every server is connected with each other by a network link with 10 Mbit/sec bandwidth.

Next, the stationary (service) agents have to be specified. In this example, four system agents with first names *ServiceSys1*, *ServiceSys2*, *ServiceSys3*, *ServiceSys4* reside at all but one agent servers. Figure 2-15 shows the parameter specification with the graphical user interface.

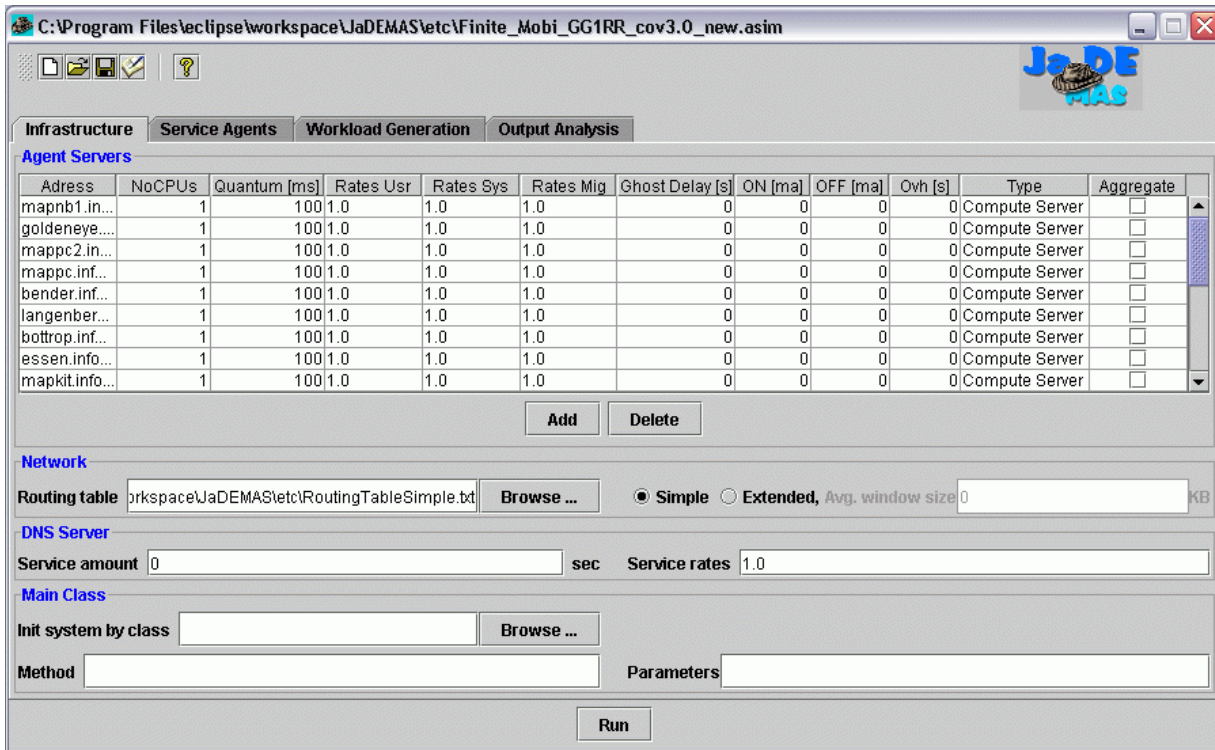

Figure 2-14: **Infrastructure of the mobile agent system**

Finally, the workload and the output analysis method have to be defined. Figure 2-16 and figure 2-17 show the corresponding entries. At simulation start, one user generates mobile agents with a mean rate of 0.05 agents per second and coefficient of variation[1] 3.0. At simulation time 20,000 seconds 5 further users with the same parameters each are added. At simulation time 60,000 seconds the 5 additional users vanish. Mobile agents with first name *Mobi* have the home server *Moneypenny*. The data volume of the mobile agents (which loads the network links) is 150 KByte.

This scenario shall be analysed with the finite horizon method. Total simulation time is 80,000 seconds. Results as there are round trip time, server utilisation, system throughput are output as moving averages with a window size of 4000 seconds. The utilisation of server "bond" is observed along the time axis.

1. Coefficients of variation > 1.0 are generated by a Cox-2 distribution, coefficients of variation = 1 are generated by a negative exponential distribution, coefficients of variation < 1 are generated by an Erlang-k distribution.

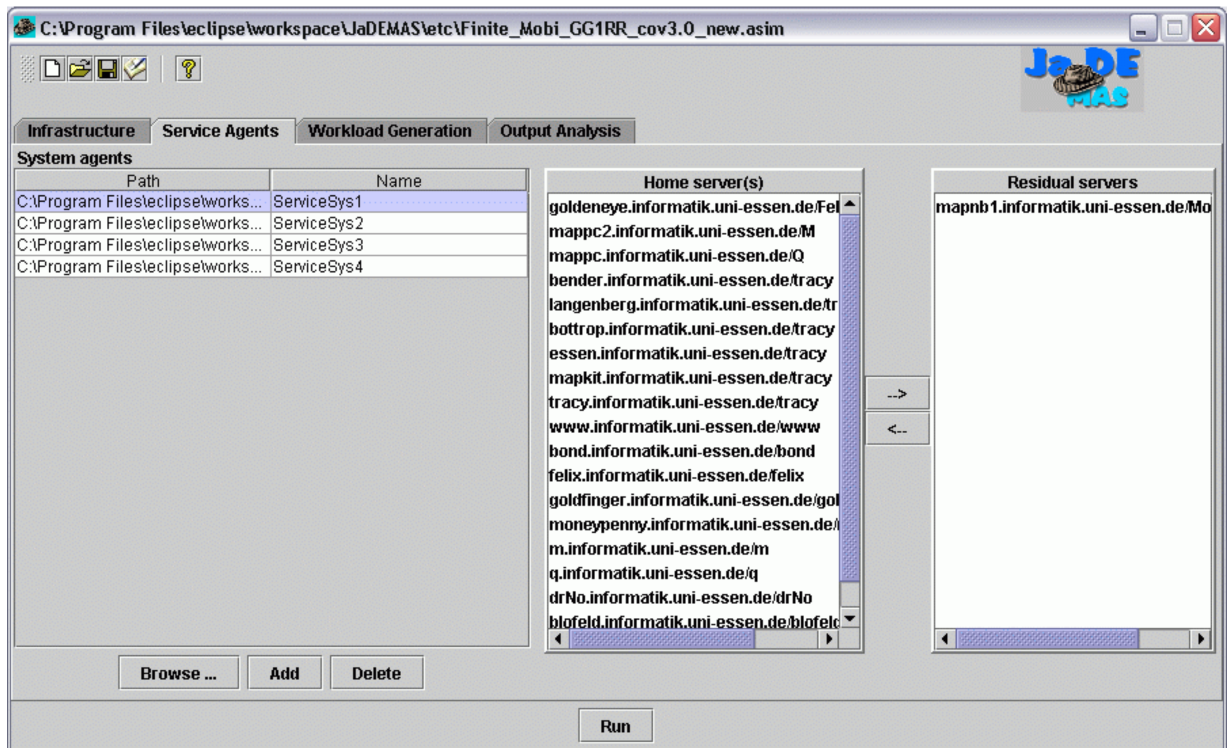

Figure 2-15: **Agent classes and allocation to home servers**

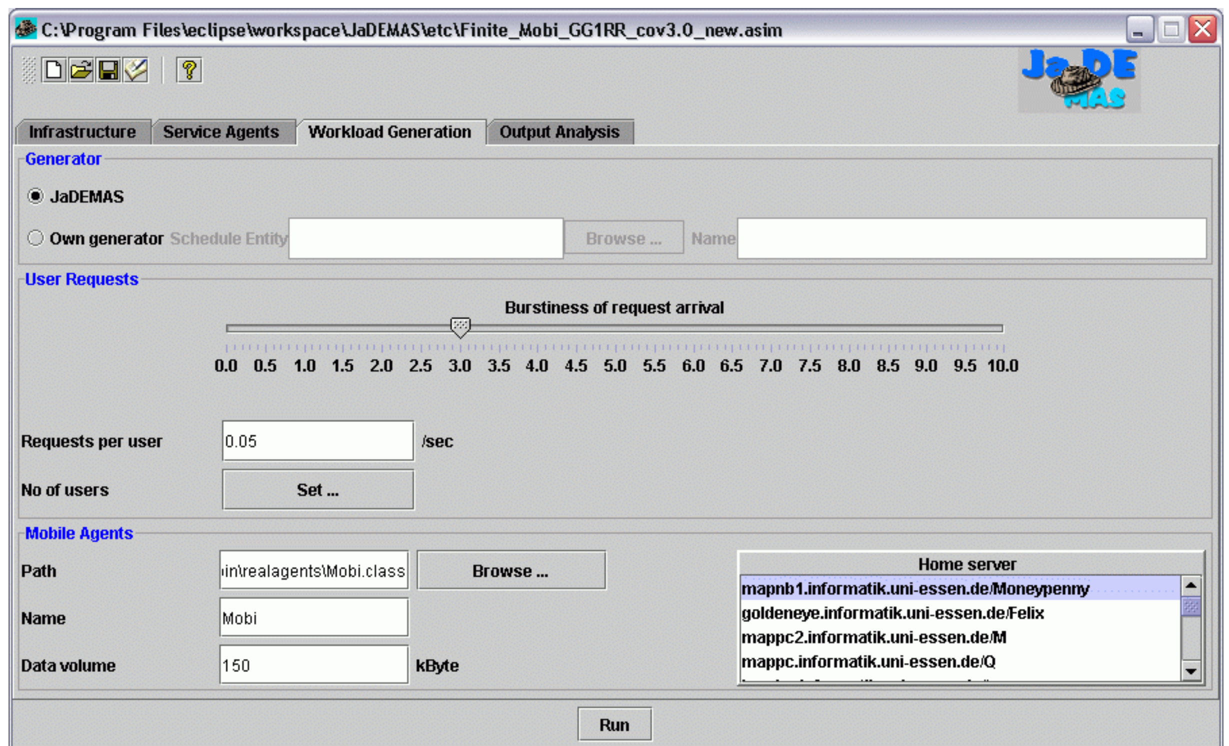

Figure 2-16: **Parameterising of workload**

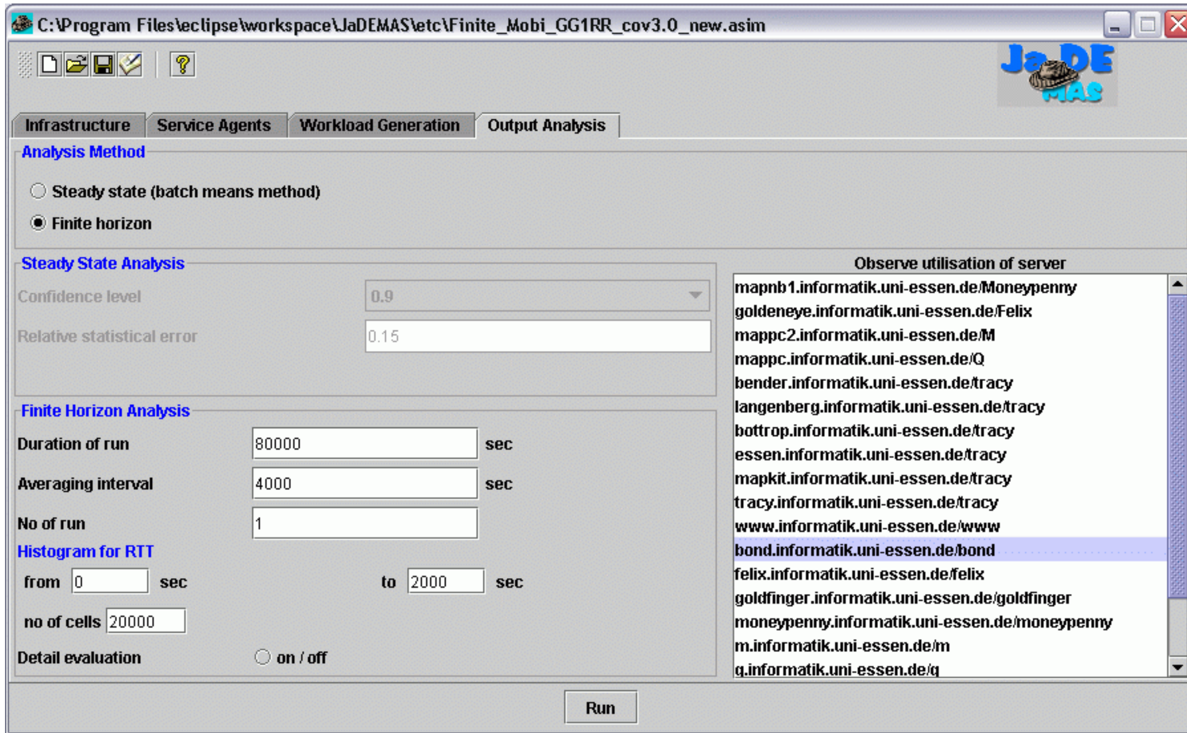

Figure 2-17: **Parameterising of output analysis**

2.5.2 Code Modification

Import packages have to be renamed in all agent classes according to *JaDEMAS* packages' names. No further modification is necessary with the program code of mobile agent *Mobi*. Figure 2-18 shows the necessary modifications of program code at the exemplary system agent *ServSys1* (bold items).

```
/******************************************************************
* Time consuming algorithm
******************************************************************/
private float calc (float n){

     ...    // calculation //

    /*************************************************************
    * Time consumption
    *************************************************************/
    double serviceTime = 0.2;
    consumeServTimeUsr(serviceTime);

    /* End time consumption */

    ...
}
```

Figure 2-18: **Modifications with the code of the system agent** *ServSys1*

The time consumption has to be added. Each system agent in the example has a deterministic service amount. In case of *ServiceSys1* the service time is 0.2 seconds per request. The service times of the other system agents are 1.0, 0.5 and 0.25 seconds.

The route of the mobile agents through the system is implemented in the program code of the mobile agents and does not have to be modified. In this example all mobile agents visit each agent server in an arbitrary order and request one of the system agents at each server.

2.5.3 Results

Figures 2-19 through 2-22 contain the results of the finite horizon analysis for a single simulation run. Figure 2-19, figure 2-21 and figure 2-22 show the variation of the performance values under varying workload along the time axis. The round trip times are averaged over a time window of 4000 seconds (as specified). Within the time interval between 20,000 and 60,000 seconds the workload is multiplied by 5. The reaction of mean round trip times, utilisation and system throughput is significant. Figure 2-20 shows the distribution of round trip times. Here, quantiles can be determined, e.g. 70.4% off all round trip times are below than 340 seconds.

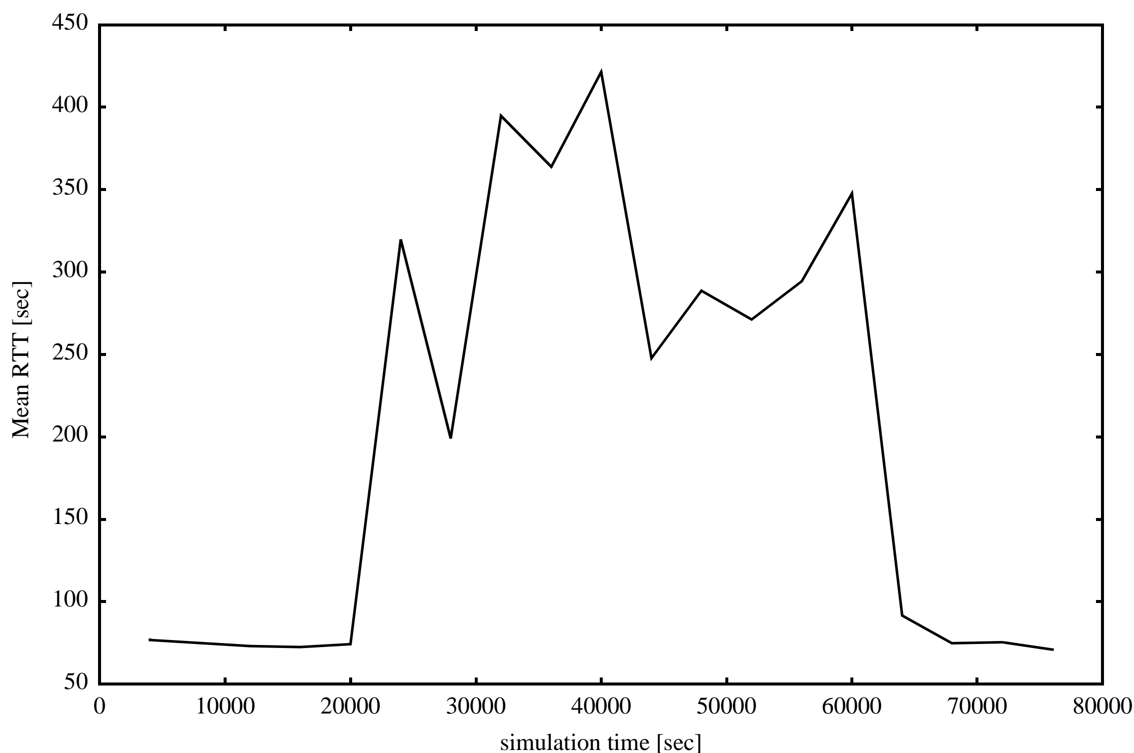

Figure 2-19: **Mean round trip time (single simulation run)**

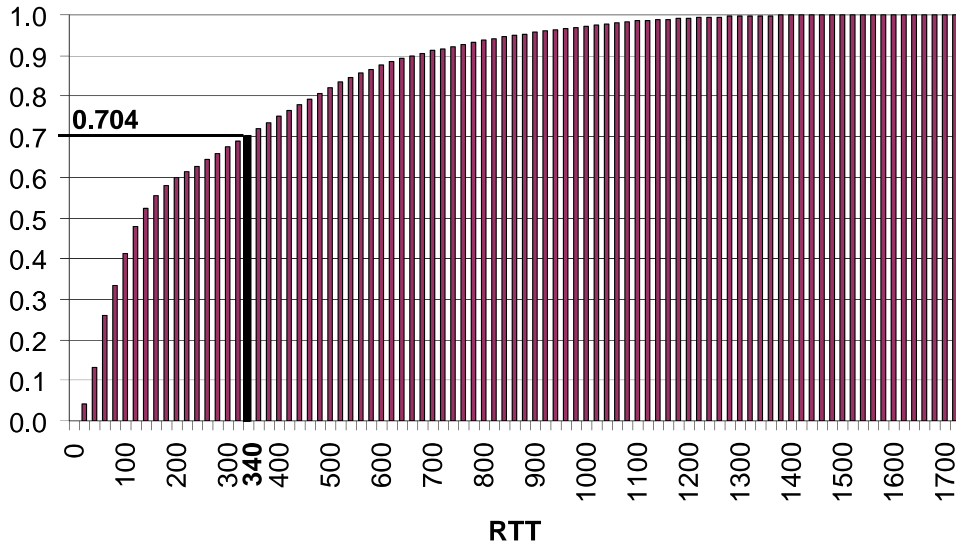

Figure 2-20: **Distribution of round trip time (RTT) (single simulation run)**

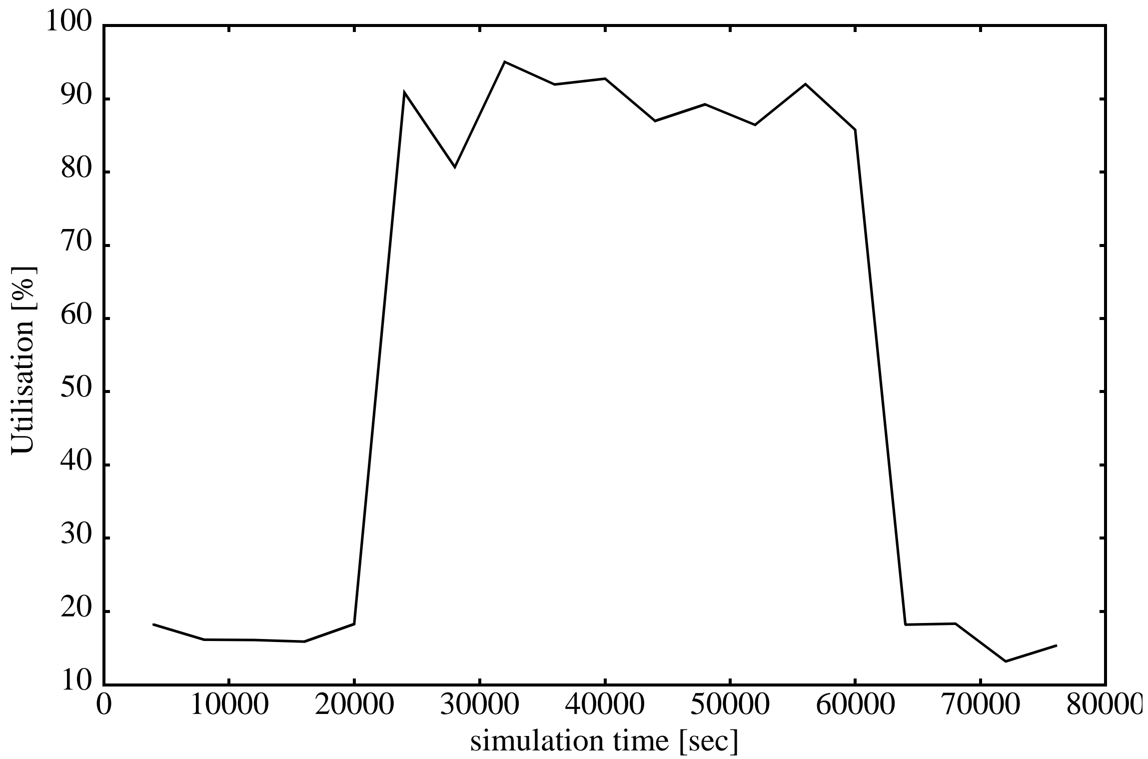

Figure 2-21: **Utilisation of server "bond" (single simulation run)**

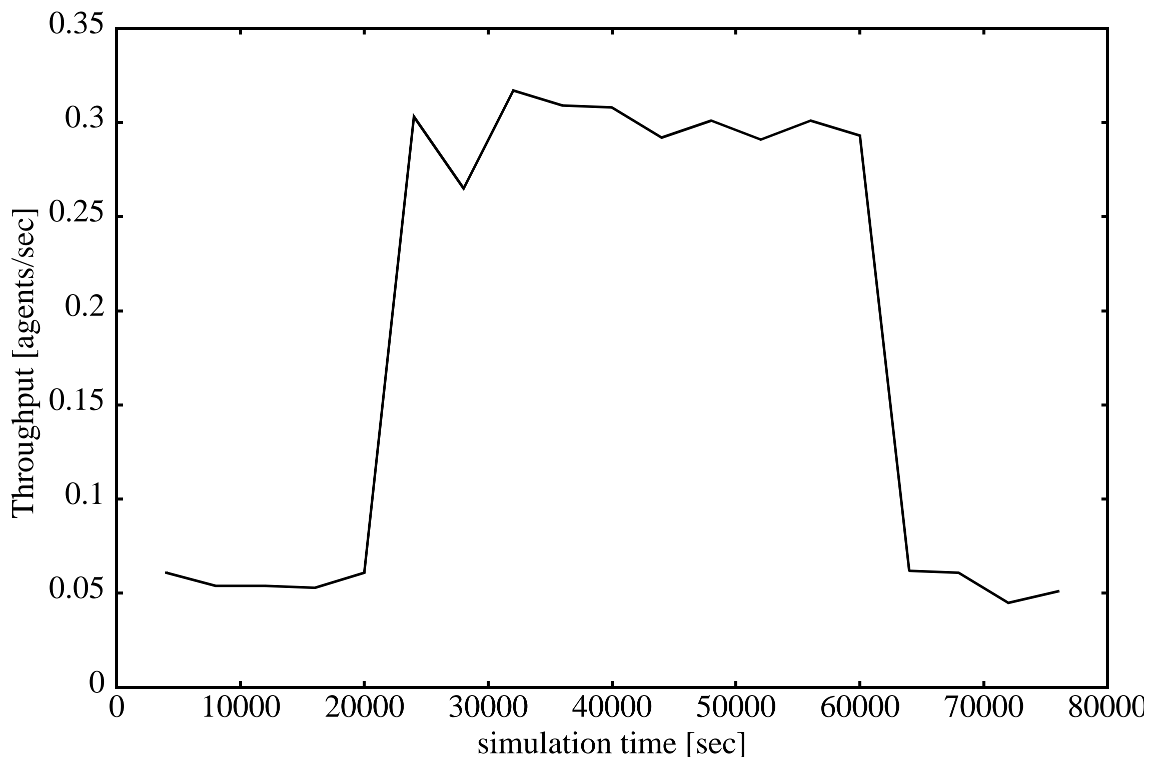

Figure 2-22: **System throughput (single simulation run)**

2.6 Portability to Other Mobile Agent Systems

The problem concerning portability to other mobile agent systems is that there exist no real standards for the methods provided by an mobile agent platform. Otherwise, mobile agents could be exchanged between platforms without problems. E.g., some mobile agent systems allow for exchanging of messages between agents which are not located at the same server whereas others (as *Tracy*) do not. Some systems have a more hierarchical structure than *Tracy* has, e. g. *Grasshopper* distinguishes between *regions*, *agencies* (corresponds to *Tracy*'s agent servers) and *places* (there can be multiple places where agents can meet within one agency).

JaDEMAS models *Tracy*'s basic methods concerning communication, migration and service processes. These are implemented from an performance point of view, i.e. functional modelling of technical details of the agent system, for example concerning migration (weak/strong) or the transport protocol for the mobile agents is irrelevant for performance simulation. Performance relevant issues caused by these technical details have to be expressed by the overhead parameters for agents at each server. The implemented basic methods should be similar in most mobile agent platforms. Hence, mapping *JaDEMAS* to another system should mainly include an adjustment to the API of the mobile agent system and should not touch the modelled mechanisms for communication, migration, scheduling, etc.. The effort to port the *JaDEMAS* to another agent system can be estimated by the effort to port agents from one mobile agent software to another.

2.7 Summary

The described simulation concepts for modelling mobile agent systems enable modellers to analyse performance issues during the development of a mobile agent system. The developed models and building blocks can be used to model mobile agent systems which implement intranet applications. It is assumed that these applications have the same basic communication mechanisms in common.

A simulation environment *JaDEMAS* has been developed which replaces a real *Tracy* mobile agent system in simulation models. Within these models, the program code of real agents (with minor modifications) is executed and determines the behaviour of the agents just like it does in the real agent system. *JaDEMAS* internally models communication, waiting processes, contention scenarios and scheduling strategies. The modeller does not have to model these operations himself. Besides the agent bytecode, he just has to specify parameters for time consumption and the infrastructure of the mobile agent system. *JaDEMAS* provides two types of agent servers: file and compute servers. Moreover, three different types of overhead at agent servers can be modelled. There exist two types of network models: a simple one where only bandwidth between linked servers has to be specified and an extended network model which models a TCP pipe between source and destination server. The latter requires multiple input parameters. With *JaDEMAS,* performance analysis and prediction of mobile agent systems in development can be easily accomplished at a time when design and implementation strategies have to be made.

3 Existing Approaches to Increase Simulation Efficiency

Zeigler et al. state: "The inescapable fact about modelling is that it is severely constrained by complexity limitations. Complexity, is at heart, an intuitive concept - the feeling of frustration or awe that we all sense when things get too numerous, diverse, or intricately related to discern a pattern, to see all at once - in a word, to comprehend." [63]. Detailed simulation may reach its limits when modelling large, complex mobile agent systems. On the one hand, detailed parameters for simulation of large/complex systems are often not completely predictable. On the other hand, modelling of detailed processes in a complex system increases the duration of simulations significantly.

When applying methods to increase model efficiency of the agent system models particular features of mobile agent systems have to be regarded. One important goal of this dissertation is to preserve the agent's program code in simulation to reduce the effort of building a performance model during implementation. Agents' behaviour is determined by their code, thus, their behaviour is not really predictable from "outside". Furthermore, model parameters can depend on the current system state if agents modify their behaviour according to the dynamic system development. Finally, approaches to increase efficiency of mobile agent system shall allow for the application of different analysis methods (steady state as well as finite horizon analysis). These requirements and those from section 2.2.1, page 9, restrict the application of a lot of existing approaches to increase efficiency.

This chapter describes existing approaches and their applicability to mobile agent models. There exist several methods to increase model efficiency, e.g. in the area of aggregation, hybrid modelling, etc.. Most approaches result from mathematical analysis of queuing systems where methods have been developed to analyse non-product form queuing networks with a acceptable effort. Section 3.1 describes such approaches and investigates their ability to improve the efficiency of mobile agent models. Section 3.3 explains why rare event simulation does not help to increase efficiency of mobile agent systems. Methods of hybrid modelling by combining simulation with analytic techniques can be found in sections 3.2 and 3.4.. Section 3.5 deals with the approach of *SHRiNK* which increases efficiency by omitting events.

3.1 Aggregation and Decomposition

A common technique to increase models' efficiency is to build aggregated models which means, that submodels are replaced by substitute representations which can be analysed more efficiently. A submodel is mapped to a substitute representation, which is equivalent to the submodel concerning the following aspects ([43]):

- The substitute representation provides the same features as the submodel.
- The service time (concerning all requested services) of substitute representation and submodel are the same.
- The substitute representation is simpler than the submodel, i.e. it was developed by transformation from the original submodel into a submodel which can be analysed more efficiently.

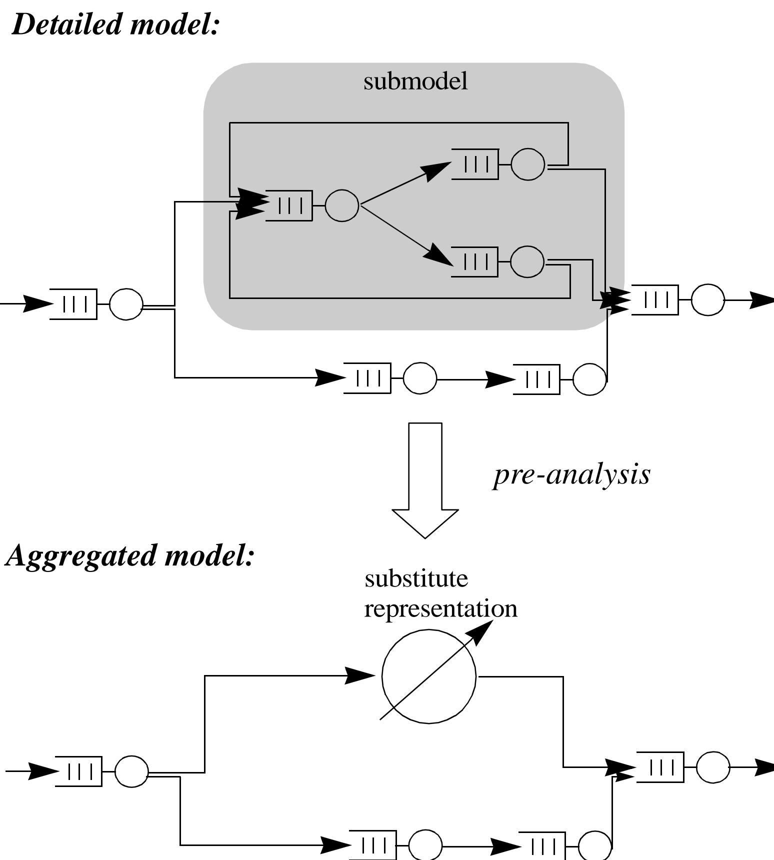

Figure 3-1: **Process of aggregation**

Transformation of a submodel into an equivalent substitute representation is called *pre-analysis*. The whole resulting model which includes the substitute representation of the submodel is called *aggregated model*. Figure 3-1 illustrates the process of aggregation.

A popular technique for the pre-analysis of queuing networks is the flow equivalent aggregation of submodels based on *Norton's theorem*, see [5] pp. 368. System components are aggregated to a substitute queuing station (so called composite queue). Therefor, the parts of the system which should be aggregated are separated from the rest of the system. This subsystem is then analysed

off-line as a closed queuing system with analytical/numerical algorithms, without considering its original environment. This analysis includes the determination of system throughput for each possible population for each class, resulting in respective service rates of the composite queue. Beside the basic approach, which considers only mean values for the service time of the composite queue, there exist a number of advanced approaches which comprise higher moments, too. For an example see [53].

Before the pre-analysis can be executed, the question arises which stations should be considered in a submodel. Baskett, Chandy, Muntz and Palacios developed the class of *BCMP* or *separable networks* [3]. It has be shown that Northon's theorem generally applies to separable networks [13] and thus, flow equivalent aggregation provides exact results, no matter which stations are analysed separately. So, one idea could be to build aggregated models of mobile agent systems within the BCMP model world.

But, BCMP networks include a number of restrictions. The following important restrictions should be mentioned exemplarily:

1. Distribution of interarrival time must be negative exponential.
2. Distribution of service time must be negative exponential (at FIFO stations) or Cox.
3. Analytic/numerical algorithms provide only mean values of relevant performance data.
4. Synchronisation, priorities, blocking and losses cannot be taken into account.

Especially restrictions 1. through 3. are inappropriate for the analysis of mobile agents systems in this dissertation, as it is one of our requirements to be able to analyse smooth as well as bulk arrivals. Further, service times are often deterministic and distributions of performance results instead of mean values shall be observed.

Following Courtois [14] it is advisable for non-BCMP networks to build submodels of substructures which are appropriate to be analysed separately. The process of separating substructures from the residual model is called *decomposition*. Models with substructures with no interconnection to other model parts are called *completely decomposable*. These substructures can be analysed separately and then can be reintegrated in the original model as composite queues without any error in model results. Models with substructures with little interconnection to other model parts are called *nearly completely decomposable* (so called *NCD feature*). Building aggregates consisting of these substructures results in minor inaccuracy, because the smaller the interaction of submodel and its environment is the more it is to legitimate to analyse the submodel separately with no regard to the environment. So, it is important to identify substructures where jobs cycle more often within than between the substructure and its environment. This feature is also called *loose coupling*. Northon's theorem and the approach of response time preservation by Agrawal et al. [1] are related to Courtois's pre-analysis algorithm.

The problem in the context of mobile agents is that there is usually no loose coupling in the agent system among agent servers: Mobile agents are sent out by their home server and usually visit their destination servers sequentially before they return home. To be more precise, no one can predict cycles because of the agents' autonomy. So, substructures (servers and networks) with little interconnection to others cannot be identified.

But, mobile agents have a stronger interconnection with resources and service agents which they share an agent server with. Hence, it is feasible to build substitutes for agent servers. This is done in chapter 4, but without using Northon's theorem because the system analysis shall exceed mean value and steady state analysis.

3.2 Simalytic Hybrid Modelling

Simalytic modelling by T. R. Norton integrates analytically modelled components into a simulation framework, see [44] and [45]. He uses more or less complex analytic submodels of system components as clients, servers and networks and uses simulation mainly to model the workload. These submodels could be built e.g. by flow equivalent aggregation or by other methods. During simulation the residence time of a transaction at a component is calculated by solving the analytic model on-line or by determining the value from a result table.

One restriction lies in the fact that the analytic queuing models used can only handle negative exponential interarrival times of transactions. Further, these analytic models provide only mean value results.

Transaction arrival rates are input to the analytic models. They are fetched from simulation by measuring the time period between two successive transactions. Per transaction the interarrival time since its prior transaction is measured, and for each transaction the analytic model is solved. Although transaction rates are grouped into classes, the arrival rate per class can change with each transaction, possibly. These arrival rates are input for analytic models which are qualified for steady state analysis, which means that this procedure constitutes a grave inaccuracy. Even if Norton constitutes that his simulations "model the application over longer periods of time" [44], he does not use the longer observation period, but calculates the interarrival rates short-term. Considering the later publication [45] this inaccuracy can be decreased: A floating mean value is used for calculating the arrival rates. This may extend the calculation to the whole simulation period.

Furthermore, it is not apparent that transactions, which reside at an analytically modelled component, are rescheduled if the workload at the component changes. In case of non-FIFO, e.g. round robin systems, the arrival of a new transaction could influence the residence time of currently present transactions. Norton mentions, that subsystems could be complex analytical models. Probably, most of this systems are no FIFO systems in whole, thus, it could happen that jobs overtake each other at a server or that jobs share resources within the analytic model. In these cases, the error in Simalytic model results increases.

The preliminary results in [44], which describe the differences in modelling with analytic queuing models, pure simulation and Simalytic, are nevertheless sufficient. The reason for this may be the very simple M/M/1 scenario. Further, Norton does not reveal the quality of his results (response times): Information about how he gained the values or about the expected error (confidence intervals) is missing.

Because the level of inaccuracy of Simalytic models is not estimable, this approach is not used to increase efficiency of mobile agent simulation.

3.3 Rare Event Simulation

With rare event simulation samples are modified, thus, rare resp. important events appear more often than with original sample distributions. This usually leads to a variance reduction of the result value under observation. Hence, simulation models provide statistically significant results with lower computational effort. Importance sampling is an example for such a method [19]. Here, the aim is to generate more events which dominate the expectation of the result value.

These methods are not applicable for the modelling of mobile agent systems because rare or important events cannot be determined. Events within the mobile agent system arise from the agent's behaviour and cannot/shall not be predicted in advance.

3.4 Hybrid Simulation

Schwetman developed an approach to combine simulation with analytical modelling techniques "to produce efficient yet accurate system models" [51]. He gives a simple example for a single computer system where discrete-event simulation is used to model the arrival process and the activation of jobs, and a queuing network model represents the use of system processors.

The allocation of system resources is divided in two phases: in the first phase the jobs arrive at the server and request so called *long-term* resources, e.g. virtual processors (regions, partitions, control points), blocks of main memory. The first phase is modelled by discrete-event simulation. In the second phase, a job has obtained all necessary long-term resources and becomes an active task. Then it consumes *short-term* resources as CPU, I/O processors and I/O devices. The time a job stays in this state is called *active time*. When there is only little interaction between the two phases, such a hybrid model is nearly decomposable [14]. "Since, in computer systems, the consumption of short-time resources occurs at a rate which is typically much greater than changes in the set of active tasks, this approximation [approximation of the active time of a job by an analytical model] should yield valid results for models of computer systems." [51]. The number of parallel active jobs is called level of multiprogramming. This level is controlled by the availability of long-term resources and changes when new jobs arrive or jobs depart.

One main assumption is that the use of short-term resources is cyclic and that there exists an analytic technique to calculate the expected cycle time of an active job. This suggests the use of a closed queuing network algorithm. "This two-phase hybrid model makes use of steady state results (the expected value of the cycle times) as a means of approximating the active time (also called the in-core time or residency time) of individual jobs." [51]. The arrival of jobs at the server and the allocation of long-term resources is modelled by simulation. The residence time of all jobs at the server is calculated analytically, depending on the number of active jobs and their service amounts. Then the job with the shortest residence time is again scheduled in simulation (which models elapsed time), the job is deleted from the list of active jobs, the multiprogramming level is updated and the remaining residence time of the remaining active jobs is anew calculated. This means, the arrival or departure of a job at the server requires the new calculation of the residence times of all active jobs. This effect has previously been denoted by "rescheduling" (section 3.2).

Schwetman gives a central server model as an example for his hybrid simulation approach. There exists a single long-term resource, a set of virtual processors, which controls the level of multiprogramming. Hence, the long-term resource is in this case a workaround to model the limited capacity of the server. The short-term resources model the components of the central server: CPU and I/O devices. Interarrival times of jobs were exponentially distributed, the number of cycles in the short-term resources of a job was drawn from a uniform distribution. The hybrid model was compared to a pure simulation model. The experiments show that the hybrid model is significantly more efficient than the pure simulation model: "The simulation-only versions required from between 18 and 200 times as much CPU time as the equivalent hybrid model." [51]. The results of the two model types were very similar. In most scenarios deviations of model output (mean values) were less than five percent. Only one test, where the NCD feature was violated

(because of higher interactions between long-term and short-term resources), showed higher deviations.

Chapter 4 describes approaches with certain similarities to Schwetman's. There are analytical substitutes of servers, too. These approaches do not calculate steady state results for the submodels, but estimate transient behaviour depending on the current system state. Hence, it is expected that results of transient analysis are more accurate.

3.5 Small Scale Hi-fidelity Reproduction of Network Kinetics

SHRiNK (Small scale Hi-fidelity Reproduction of Network Kinetics) [46], [49] is an approach to increase model efficiency by reducing the number of arrival events in queuing systems. Each arriving job is sampled with probability α, independent of the other jobs. This results in scaling down the arriving rate by factor α ($0 < \alpha < 1$). Respectively, servers are set to a slower speed, i. e. service rates are scaled down by the same factor. Psounis et al. state that SHRiNK provides correct results for M/G/- queues, resp. networks, and they verify the method by simulations and benchmarks of IP networks and web server farms.

At first glance, SHRiNK does not violate any of the requirements (see section 2.2) for modelling mobile agent systems. Hence, the approach shall be analysed in more detail and the applicability of SHRiNK shall be demonstrated with simple examples.

Starting from the well known formulas for the steady state analysis for a single M/G/1 system the SHRiNK approach is feasible: It can be analytically calculated for a single M/G/1 system how scaling effects steady state performance results. Scaling results in multiplying the arrival rate λ with α to λ_s and the service rate μ is modified respectively: $\lambda_s = \alpha\lambda$; $\mu_s = \alpha\mu$.

It is easy to observe that utilisation $\rho = \frac{\lambda}{\mu}$ remains unchanged.

According to the *Pollaczek-Khintschin* formula ([5], p. 111) in combination with *Little's Law* the expected value of the residence time $E[R_s]$ of the scaled system results as follows:

$E[R_s] = \frac{1}{\alpha\mu} \cdot \left(\frac{\rho}{2(1-\rho)} \cdot (V[S_s] \cdot (\alpha\mu)^2 + 1) + 1\right)$, where $V[S_s]$ = variance estimate of the scaled service time.

$V[S_s] = \frac{1}{n-1} \cdot \left(\sum_{i=1}^{n} \frac{s_i}{\alpha} - \frac{\bar{s}_n}{\alpha}\right)^2 = \frac{1}{\alpha^2} \cdot V[S]$, where s_i = service time samples and $\bar{s}_n$ = average of service times.

The outcome of this is: $E[R_s] = \frac{1}{\alpha} \cdot E[R]$.

Hence, to calculate the mean residence time resulting from a SHRiNK model back to the "original" value it has to be multiplied by α.

Under the assumption of Poisson arrival streams, it is valid to leave out arriving jobs with probability α without changing the characteristics (especially the variance) of the arrival process. Respectively, Psounis et al. state that SHRiNK is only correct in M/G/- systems. The authors assume packet flows which arrive with negative exponential distributed interarrival times. The

number of packets per flow (which constitutes the service amount) is assumed to be Pareto distributed [49].
It can be as well calculated that SHRiNK is applicable for open BCMP networks by using the exact algorithms, see [5], p. 300 et seqq.

Applying SHRiNK to G/G/n systems could be feasible when simulating many overlapping arrival streams. Even if the single streams are no Poisson streams the cumulative arrival stream at a station approximates Poisson if more than 25 streams overlap [61].

The applicability of the SHRiNK approach will be investigated by the analysis of a simple M/G/1 system. The applicability is tested for steady state analysis as well as for finite horizon analysis.

According to the method of sequential simulation [47] one should analyse confidence intervals of the important performance values and should not stop simulation until the confidence interval has a desired width. The threshold for the desired width is given by the relative statistical error which is defined as the ratio of the half-width of the confidence interval and the point estimate. The following experiments show the application of this technique for steady state analysis with SHRiNK.

The modelled scenarios are as follows: n identical Poisson sources send requests to the server. Each source produces jobs with service amounts S with the coefficient of variation *c.o.v. [S]*. c.o.v.[S] > 1.0 is implemented by Cox-2 distributed[1] service times, c.o.v.[S] < 1.0 by Erlang distributed service times. The arrival rate per job generating source i is $\lambda_i = 1.0$, the mean service rate is $\mu = 30.0$. The scale factor α is implemented by reducing the number of sources to αn, service amounts are multiplied by factor α, respectively. Resulting residence times are multiplied by α, too. Figure 3-3 and figure 3-3 illustrate the models.

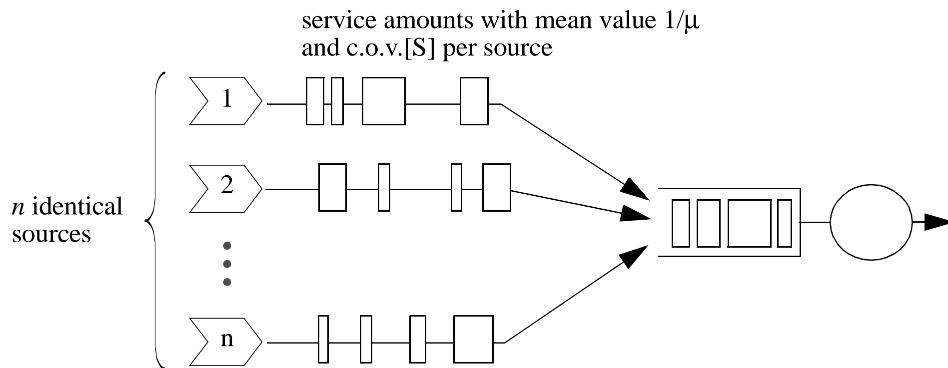

Figure 3-2: **Detailed M/G/1 model**

1. Assumption: both phases are evenly utilised, i. e., $\mu_2 = a_1 \cdot \mu_1$, where μ_1 and μ_2 are the service rates of the two phases and a_1 is the probability to pass through phase 2 after phase 1.

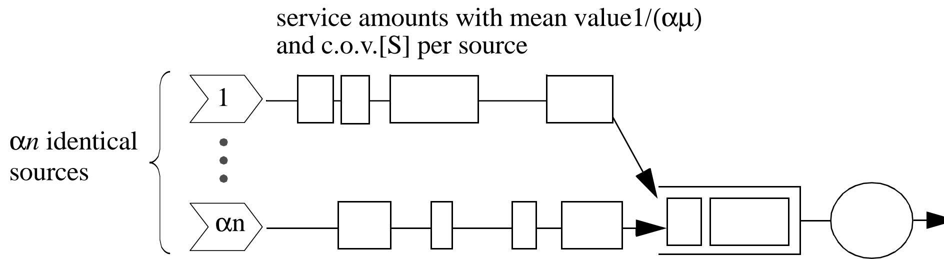

Figure 3-3: **SHRiNK M/G/1 model**

A steady state analysis is executed using the batch means method (see [29], page 432 et seq.) with batch size of 5000 observations. The minimum number of batches for calculating the confidence interval is set to 10. Simulations are stopped when the relative statistical error is smaller or equal 0.05.

To decide whether results match satisfactorily the *paired-t confidence interval* is calculated, a method for the comparison of two alternative systems described by Law and Kelton ([36] page 557 et seqq.). The approach is modified for batch means analysis. Differences of the mean values of the batches between the detailed model and the SHRiNK models are built and a 95% confidence interval is calculated for these differences. The number of compared batches is determined by the minimum needed to achieve the threshold for the relative statistical error for the residence time, e.g. in the experiment described in the third row of table 3-1, resp. of table 3-2, 37 batches are compared as only so many batches are available from the SHRiNK model.

c.o.v.[S]	util %	detail model no. of sources	scale α	detail model 95% CI R	SHRiNK 95% CI R	calc. E[R]	95% CI of differences
0.333	34	10	0.300	[0.041, 0.043]	[0.042, 0.044]	0.043	[0.000, 0.001]
0.333	80	24	0.167	[0.099, 0.109]	[0.102, 0.112]	0.107	[-0.004, 0.011]
4.000	34	10	0.300	[0.176, 0.194]	[0.165, 0.183]	0.179	[-0.033, -0,001]
4.000	80	24	0.167	[1.157, 1.279]	[1.179, 1.303]	1.167	[-0.067, 0.128]
6.000	34	10	0.300	[0.320, 0.354]	[0.318, 0.352]	0.351	[-0.027, 0.020]
6.000	80	24	0.167	[2.146, 2.372]	[2.318, 2.562]	2.500	[0.024, 0.397]
10.000	17	5	0.400	[0.400, 0.442]	[0.404, 0.446]	0.378	[-0.026, 0.034]
10.000	34	10	0.300	[0.951, 1.051]	[0.925, 1.023]	0.901	[-0.108, 0.031]

Table 3-1: **Comparison of steady state results of SHRiNK and detailed M/G/1 models**

The assumption is that SHRiNK models provide the same quality of results as the detailed model if 0 is included in the confidence interval, i.e. the "no-difference" value lies within the interval.

Table 3-1 shows the results. The confidence intervals of the residence times (*95% CI R*) overlap at the analysed scenarios (even if some CI of detailed models as well as SHRiNK models do not meet the analytically calculated mean value *calc. E[R]*). Further, the confidence intervals of the model differences (*95% CI of differences*) include 0 except for two models (shadowed rows in the table). A reason for these deviations could result from the remaining probability of 5% for the mean value not to be in the confidence interval. Another reason might be the high varying number of available batches which are compared. The relative statistical error concerning the

confidence intervals of differences is quite high sometimes. For M/G/1 systems in equilibrium state SHRiNK models provide the same results for residence time as detailed models. This could be proved by mathematical calculation as shown before.

c.o.v. [S]	util %	detail model no. of sources	scale α	Efficiency gains of SHRiNK (decrease of CPU time) [%]	detail model no. of batches	SHRiNK no. of batches
0.333	34	10	0.300	0	10	10
0.333	80	24	0.167	-7	10	12
4.000	34	10	0.300	39	61	37
4.000	80	24	0.167	-38	288	423
6.000	34	10	0.300	3	164	161
6.000	80	24	0.167	-26	644	853
10.000	17	5	0.400	4	376	362
10.000	34	10	0.300	18	491	404

Table 3-2: **Comparison of efficiency of SHRiNK and detailed M/G/1 models**

Table 3-2shows that SHRiNK does not in general increase simulation efficiency in steady state analysis. The execution time of the SHRiNK models (decrease of CPU time[1]) shown in the shadowed lines are either equal to the execution time of the detailed model or significantly higher.

The width of the confidence interval for the residence time is influenced by the variance of the observed values and the sample size, which is here the number of batches. In [46] Psounis et al. state that the variance of the queuing delay increases when decreasing α ($0 < \alpha < 1$). The variance of the response time is directly influenced by the variance of the queuing delay. One could suppose, that the variance of the response time of a SHRiNK model was higher than the variance of the detailed model, where $\alpha = 1$. This would mean an increasing of the number of necessary batches (to keep the threshold for the relative statistical error) when decreasing α. Hence, SHRiNK was generally not applicable to stationary analysis, as there would always be a higher number of necessary batches as in detailed simulation, which results in a worse performance. But, this effect cannot be observed in the scenarios described above. A higher number of batches can be found in some SHRiNK models with different c.o.v.[S] and with different α, but mainly with higher utilisation (80%). The higher number of batches causes the higher execution times of the SHRiNK models.

An analysis of variance estimates of queuing delays in two scenarios (see table 3-3) explains the contradiction between Psounis et al.'s statement and the observed results: An M/G/1 model is analysed with different c.o.v.[S] and different scale factors α[2]. Obviously, Psounis et al.'s statement refers to the variance of the "shrinked" queuing delay (estimated by $V[W_s]$), i.e. the queuing delay which arises at the scaled down server. With decreasing α, $V[W_s]$ increases. But, relevant for the model results is the variance scaled back by α, which corresponds to the results

1. Experiments were executed using JavaDEMOS (based on Java jdk 1.4.1) on a PC with AMD Athlon processor with 1,2 GHz, 512 MByte memory and with Windows 2000 operating system.

2. $\alpha = 1$ corresponds to detailed simulation.

of the detailed simulation model: $V[W] = V[W_s] \cdot \alpha^2$. V[W] finally effects the variance estimate V[R] of the residence times. A general increasing of V[W] with decreasing α is indeterminable.

c.o.v.[S]	scale α	V[W]	$V[W_s]$
4.000	1.000	1.982	1.982
	0.500	1.690	6.760
	0.167	0.790	28.313
6.000	1.000	7.552	7.552
	0.500	12.752	51.008
	0.167	8.330	298.668

Table 3-3: **Variance with different α**

This explains the varying number of batches which are necessary to hold the threshold for the relative statistical error. Hence, SHRiNK is not generally applicable with sequential simulation to increase efficiency of steady state analysis. It depends on the simulated scenario. Experiments with finite horizon analysis show the same results. Moreover, the scale value α should not be too large. Otherwise approximation errors arise. A detailed description of the finite horizon analysis with SHRiNK can be found in Appendix A.1.

Summarising, SHRiNK is not generally applicable with sequential simulation, i.e. it is not in all scenarios more efficient than detailed simulation. Results concerning residence time and utilisation are altogether satisfying. But, to get results of corresponding statistically quality as with detailed models more replication runs in the analysed scenarios have to be done, resp. sometimes more batches with batch means have to be built. This increases CPU time consumption. Unfortunately, it is not precisely predictable in which scenarios it is possible to increase performance by using SHRiNK. Performance and functional analysis were done only for simple examples, since already with these scenarios deficits of SHRiNK became apparent.

Furthermore, SHRiNK is restricted to models of M/G/- systems, resp. to G/G/- systems with more than 25 sources, where the cumulated arriving stream approximates a Poisson stream. Because of these restrictions and because of the fact that SHRiNK is sometimes inefficient, it is not used in this dissertation to increase performance of mobile agent system models.

3.6 Summary

This chapter describes existing approaches which at a first glance might be used to increase performance of mobile agent system simulations. More detailed analyses show that none of these approaches is exactly appropriate considering the former specified requirements (see section 2.2). Hence, alternative methods have to be developed which will be described in the next chapters.

The methods described in chapter 4 have to face the following restrictions resulting from features of mobile agents and from the general objectives which are set in this dissertation:

Feature	Consequence for Simulation
Routing is unknown outside the mobile agent program code. They hold their route in their code and can even change the route autonomously.	Distributions of arriving streams at agent servers and network links are unknown. They result from executing the agent code during simulation.
There are usually no cycles in mobile agents' routes. Thus, there is no loose coupling between parts of the agent system consisting of servers and network links.	Aggregation of several servers and links to an (analytic) submodel is not possible without significant errors in results. Aggregation is only possible within single servers and links, as there is usually a loose coupling to their environment.
Different user types and different load scenarios shall be analysed. Therefor, several distributions of input and service processes (not only Markovian) are necessary.	Networks of G/G/n stations shall be modelled.
Output analysis shall exceed mean value analysis. Coefficient of variation, quantiles, histograms are analysed.	(Aggregated) simulation model has to provide relevant output values.

Table 3-4: **Features of Mobile Agent System Analysis and Consequences for Simulation**

4 Efficient Simulation of Mobile Agent Systems

As described in the previous chapter, most of the common approaches to increase simulation efficiency are not suitable with regard to the specific features of mobile agent systems and to the goals of this dissertation. Hence, new approaches have to be developed, resp., existing ones have to be extended or modified.

One important goal of this dissertation is to use real agent code with a minimum of modifications in simulation. Agent code has to be considered as black box from the sight of the simulation environment. Agents' behaviour like, e.g., route selection through the agent system evolves during simulation. Another important goal is to obtain quality of service measures beyond mean values (e.g. distribution of round trip times). Hence, simulations have to provide variance measures, quantiles etc.. These features of the simulation models have to be preserved if methods to increase model efficiency are used.

As mentioned before, one feature of mobile agents is that their routes are usually loop-free. Thus, no loose coupling can be identified between parts of the agent system, composed of several servers and network links. This restricts the possibilities to aggregate parts of the agent system by analytic submodels. Cycling activities can only be found within agent servers, hence, it is feasible to aggregate processes at single agent servers. In fact, this technique is used to increase model efficiency as shown later on.

This chapter describes new approaches to model file and compute servers in mobile agent systems more efficiently while preserving model's expressiveness. Instead of the detailed simulation of processes at agent servers, alternative, more efficient substitutes are used. Several scenarios modelled with these approaches are analysed empirically. Thereby, it is shown that the methods developed increase efficiency by preserving validity and expressiveness of the performance results.

4.1 Experimental Environment

Sections 4.4 and 4.6 describe several experiments which demonstrate the validity of the developed methods and show their efficiency gains. These experiments are executed using the following hardware equipment and evaluation and analysis techniques:

The simulation environment *JaDEMAS* (see section 2.4) is used to run performance simulation on one of the following PCs:

- *goldeneye* with an Intel Pentium IV processor with 2.6 GHz, operating system Windows XP, 1 GByte RAM,
- *trinity* with an Intel Pentium IV processor with 2.0 GHz, operating system Windows XP, 512 MByte RAM,
- *goldfinger* with an AMD Athlon processor with 1.2 GHz, operating system Windows 2000, 512 MByte RAM,
- *blofeld* with an Intel Pentium III processor with 1.0 GHz, operating system Windows 2000, 256 MByte RAM.

To show the wide applicability of the approaches, steady state and finite horizon analyses have been executed. Both analysis methods are implemented using the *sequential simulation* method [47]: A 90% confidence interval is calculated for mobile agent's round trip time. The simulation stops when the relative statistical error of the round trip time is smaller than or equal to a certain threshold. This threshold is set to 0.15. The relative statistical error is defined by the ratio of the half-width of the confidence interval and the point estimate. The steady state analysis is implemented using the *batch means* method [29] as implemented in *JaDEMAS* (see section 2.4.8, page 26). Independent from the reaching of the relative statistical error, at least 10 batches are built. Accordingly, the finite horizon analysis executes several replication runs which all provide mean values for the round trip time. The confidence interval is calculated from these mean values. The number of replication runs depends on the number which are necessary to achieve the threshold for the relative statistical error.

To decide whether results of two different model types match satisfactorily a *paired-t confidence interval* is calculated, a method for the comparison of two alternative systems described by Law and Kelton ([36] page 557 et seqq.). In case of batch means analysis, the approach is modified: The mean value of round trip time per batch is recorded. Then, differences between the batches' mean values of both models are computed and a 90% confidence interval is calculated for these differences. The number of batches to compute differences is determined by the model with the smaller number of batches. This number results from the number of batches which are necessary to achieve the threshold for the relative statistical error of the round trip time.

In case of finite horizon analysis, results are compared along the time axis. Thereby, it is necessary to collect result values at the same timestamps. This is not the case if, e. g., single round trip times are surveyed at the return of mobile agents at their home server. This problem can be solved by calculating averages over result values within moving time windows of fixed size. For the experiments in the following sections a window size of 4000 seconds has proven to be suitable. To calculate the paired-t confidence interval the mean value of round trip time is computed for each time window in each simulation run. Then, the difference between both models is built per time window, per simulation run. Finally, a 90% confidence interval is calculated for these differences. Thus, there results a paired-t confidence interval for each time window. Similar to the batch means analysis, the number of runs to compute differences is determined by the model with the smaller number of runs.

4.2 Concatenated Servers

A way to increase efficiency of process oriented, discrete event simulation is to omit events and, thus, to omit overhead of process management. The approach of concatenating agent servers implements this idea by accumulating time consumption which can be calculated analytically to a single delay value. Instead of generating events at the end of each single delay, one single event is produced at the end of the accumulated delay.

4.2.1 Concatenating Mobile Agent Servers

As described in section 2.4, the network delay of a mobile agent is calculated analytically, as well in the simple as in the extended network model. The network delay follows the end-of-service event of an agent at a server. If the residence time of a mobile agent can be calculated analytically, it can be accumulated with the network delay, so that the first event at an agent server is the arrival of a mobile agent and the next event is the arrival of this agent at the next server (after residence time plus network delay have elapsed). Thus, the departure event of a mobile agent from an agent server can be omitted. Figure 4-1 explains this approach. Figuratively, servers are directly "concatenated" with each other by omitting modelling of the network delay separately.

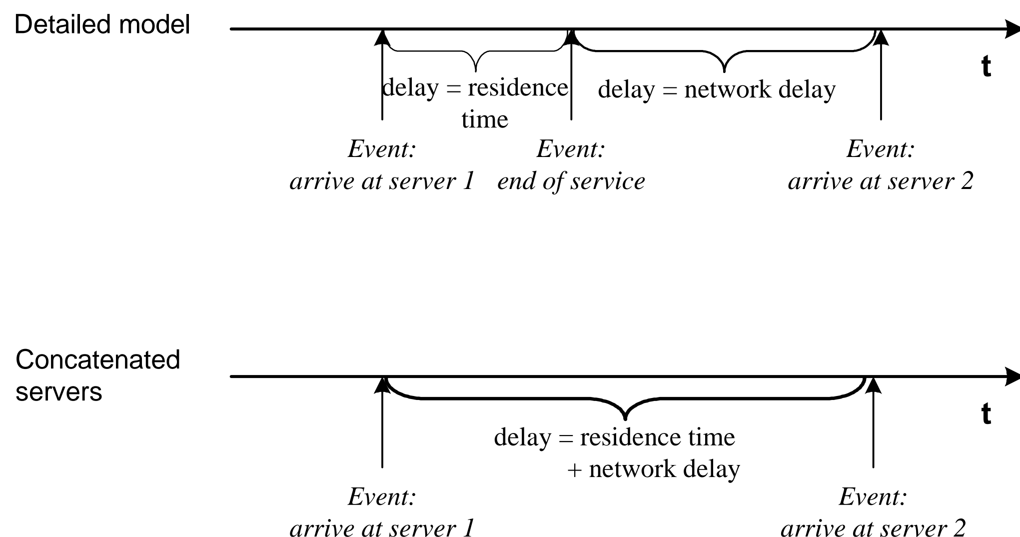

Figure 4-1: **The concept of directly concatenating agent servers**

Hence, the challenge is to calculate mobile agents' residence time at an agent server analytically. Therefor, it is necessary to take a closer look at mobile agents' activities at an agent server. In consequence of the security concept, mobile agents are allowed to do hardly anything at a server. According to the assumed scenarios (see section 2.1, page 8), they send requests to stationary service agents which utilise server resources to provide a service. Figure 4-2 describes this process. Mobile agents directly request a specific service agent via a message addressed to its name. A mobile agent M_j passively waits until the desired service agent S_i is ready to take its request. Next, the service agent acquires server resources to fulfil its service and sends the result back to the waiting mobile agent. It is assumed that a service agent releases server resources after sending the result to the mobile agent (even if he has to serve further mobile agents) and again com-

petes with the other service agents for server resources. Actually, mobile agents' messages are put into the message queue of their service agent. When *Tracy's* blackboards are used, no waiting time arises.

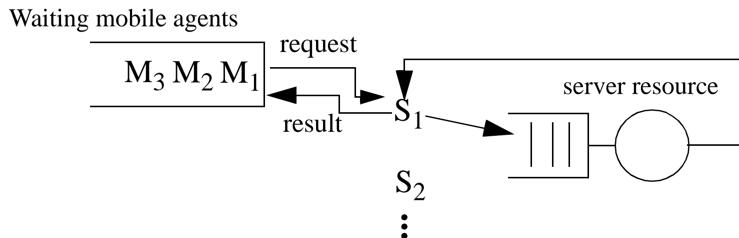

Figure 4-2: **Detailed process of serving mobile agents by service agents**

Figure 4-3 describes the time consumption during the residence of a mobile agent at an agent server and its migration afterwards. The residence time of a mobile agent is composed of two phases:

1. Waiting time for the desired service agent to be idle,
2. Residence time of the service agent at the server resource.

Afterwards, the mobile agent consumes time for its migration to the next server.

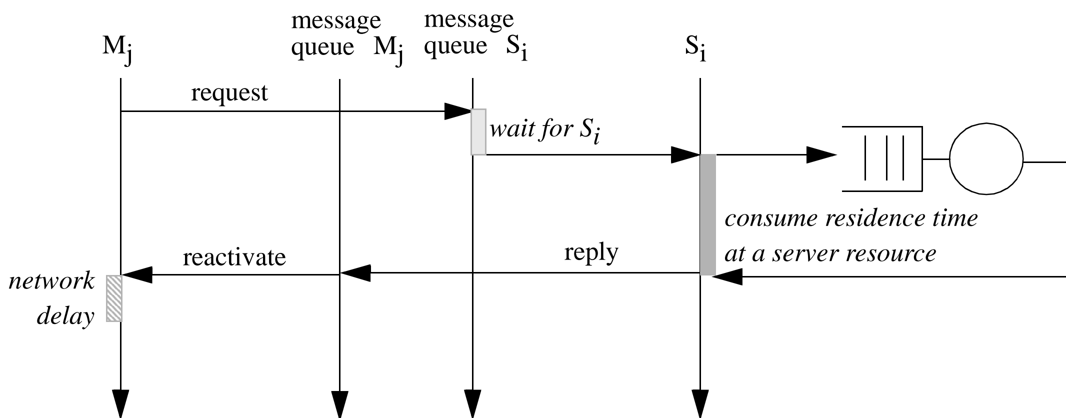

Figure 4-3: **Time consumption with serving mobile agents by service agents**

The time consumption of the mobile agent (M_j) at the server is indirect, because it waits for the service agent (S_i) which directly consumes time at the server resource. M_j directly consumes time for its transfer through the network.

According to the considerations above, it is useful to calculate the residence time from the mobile agent's point of view to accumulate it with the following network delay. Hence, the mobile agent can consume the residence time at the server plus the network delay directly in one step. Figure 4-4 shows the resulting modelling of time consumption. The mobile agent M_j waits for its service agent S_i to be idle. When this is the case, S_i does not consume its residence time at the server resource, but just calculates this delay. It directly sends the reply message back to M_j which then is delayed for the calculated residence time of S_i plus its own calculated network

delay. Hence, M_j consumes the residence time at the server resource for S_i representatively. This way of modelling is sensible because anyway, M_j waits until S_i's residence time has elapsed.

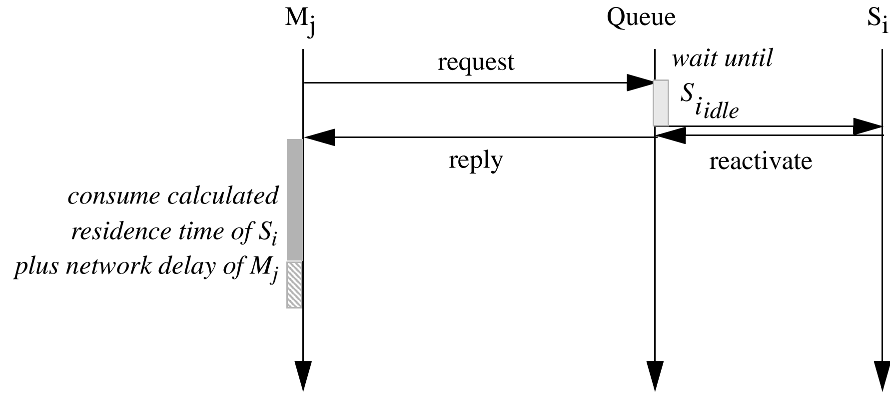

Figure 4-4: **Modelling of time consumption with concatenated agent servers**

Furthermore, analytic calculation of S_i's residence time has the advantage that management processes at the server side (e.g. administration of agents in queues, scheduling) do not have to be simulated in detail, which increases efficiency significantly.

4.2.2 Calculation of the Accumulated Delay

The complete residence time of a mobile agent M_j is composed of the sum of the waiting time until S_i is idle plus S_i's residence time at the server resource. The timestamp when S_i is idle shall be named $S_{i_{\text{idle}}}$. $S_{i_{\text{idle}}}$ is stored in a data structure for each service agent. The residence time of a mobile agent M_j can then be calculated in two steps:

1. Wait until $S_{i_{\text{idle}}}$. The value for $S_{i_{\text{idle}}}$ is read from the data structure. S_i can take new requests of mobile agents not until it is idle, i.e. not until it has finished serving previous mobile agents.
   At M_j's arrival (t_{M_j}) the residence time R_{S_i} of S_i is not calculable if S_i is currently busy: It could happen that, e.g., while M_j is waiting for $S_{i_{\text{idle}}}$ other mobile agents arrive and request other idle service agents which are then put into the resource queue earlier than S_i again. This means, the mobile agent has to be delayed in a first step until $S_{i_{\text{idle}}}$, then the residence time of S_i can be calculated depending on the number of agents currently residing at the server resource.
2. At $S_{i_{\text{idle}}}$ the residence time R_{S_i} of S_i at the server resource is calculated. Sections 4.3 and 4.5 show how R_{S_i} is computed depending on the type of server resource.

The residence time of a mobile agent M_j can be defined as: $R_{M_j} = S_{i_{\text{idle}}} - t_{M_j} + R_{S_i}$. The accumulated delay of M_j then results in $d_{acc} = R_{M_j} + d_{net}$, where d_{net} = calculated network delay.

4.3 Concatenated FIFO Servers (CFS)

As already mentioned, the calculation of the residence time R_{S_i} of a service agent S_i at a server resource depends on the type of the resource. The approach of *concatenated FIFO servers* (*CFS*) implements the calculation of R_{S_i} on file servers, which primary resource, the I/O device, schedules requests in a first-in-first-out manner.

4.3.1 Calculation of Service Agents' Residence Time at the I/O Device

It is well known that the residence time of a job arriving at a FIFO server consists of the sum of its own service time, the service times of jobs already waiting and the residual service time of the job in service.

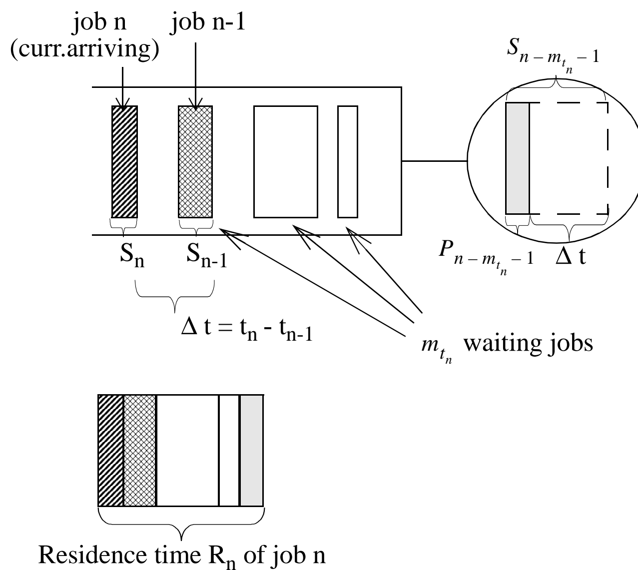

Figure 4-5: **Composition of the residence time of job n**

The following definitions assume that jobs are enumerated according to their arrival order at the server. Figure 4-5 illustrates the calculation of a job's residence time with the following notations:

R_k = residence time of job k,

S_k = service time of job k,

t_k = arrival time of job k,

P_k = residual service time of job k,

m_{t_k} = number of jobs in waiting queue at t_k.

The gaps between the jobs in the queue display the interarrival times. Actually, jobs are arranged successively without gaps in the queue. The residence time of a job at a FIFO server can be computed by the following formula:

$$R_n = S_n + \sum_{i = n - m_{t_n}}^{n-1} S_i + P_{n - m_{t_n} - 1} \qquad \text{(Eq. 1)}$$

If indices get smaller than zero, the corresponding terms are assumed to be zero.

Figure 4-5 shows how (Eq.1) can be defined recursively. The upper half of the figure shows the scenario that job *n* currently arrives at the queuing station. The previous job *n - 1* just arrived when job $n - m_{t_n} - 1$ entered the server. The time Δt elapsed between the arrival of job *n-1* and the arrival of job *n*. Thus, at t_n the residual service time of job $n - m_{t_n} - 1$ is its service time reduced by Δt. If Δt was larger than $S_{n - m_{t_n} - 1}$, one of the following jobs, e.g. job $n - m_{t_n} + 1$, would be in service and its residual service time could be calculated by

$$P_{n - m_{t_n} + 1} = S_{n - m_{t_n} + 1} - \left(\Delta t - P_{n - m_{t_n} - 1} - S_{n - m_{t_n}}\right).$$

(Eq.1) can be defined regressively as:

$$R_{n+1} = S_{n+1} + S_n + \sum_{i = n - m_{t_n}}^{n-1} S_i + P_{n - m_{t_n} - 1} - \Delta t$$

$$\Leftrightarrow R_{n+1} = S_{n+1} + R_n - \Delta t$$

Obviously, $R_n \geq S_n \quad \forall n \in \mathrm{N}$.

This leads to the following definition which is equivalent to (Eq.1):

$$R_1 = S_1$$

$$R_{n+1} = \begin{cases} S_{n+1} + R_n - t_{n+1} + t_n & \text{if} \quad R_n - t_{n+1} + t_n \geq 0 \\ S_{n+1} & \text{otherwise} \end{cases}, n \in \mathrm{N} \qquad \text{(Eq. 2)}$$

In simulation, (Eq.2) allows for the calculation of the residence time R_{n+1} by only storing R_n and t_n (s_{n+1} and t_{n+1} are known for the current job number $n+1$). Scheduling of jobs and handling of the waiting queue is not necessary. Additionally, this means that allocation of resources (e.g. `Res` objects in *JavaDEMOS*, which manage own waiting queues) is not necessary. Transferred to the mobile agent system simulation, the residence time of a service agent S_i at the processor can be calculated as:

$$R_{S_1} = S_{S_1}$$

$$R_{S_i} = \begin{cases} S_{S_i} + R_{S_{i-1}} - t_{S_i} + t_{S_{i-1}} & \text{if} \quad R_{S_{i-1}} - t_{S_i} + t_{S_{i-1}} \geq 0 \\ S_{S_i} & \text{otherwise} \end{cases}, i \in \mathrm{N} \qquad \text{(Eq. 3)}$$

The usage of (Eq.2) shall be illustrated by a simple example which is shown in figure 4-6: Job 1 may be the first job which arrives at the queuing station at $t_1 = 0$. Its service time may be $S_1 = 10$. It goes directly into the idle server, thus, its residence time is $R_1 = S_1 = 10$. Next job 2 arrives at $t_2 = 15$, i.e. $\Delta t = 15$.

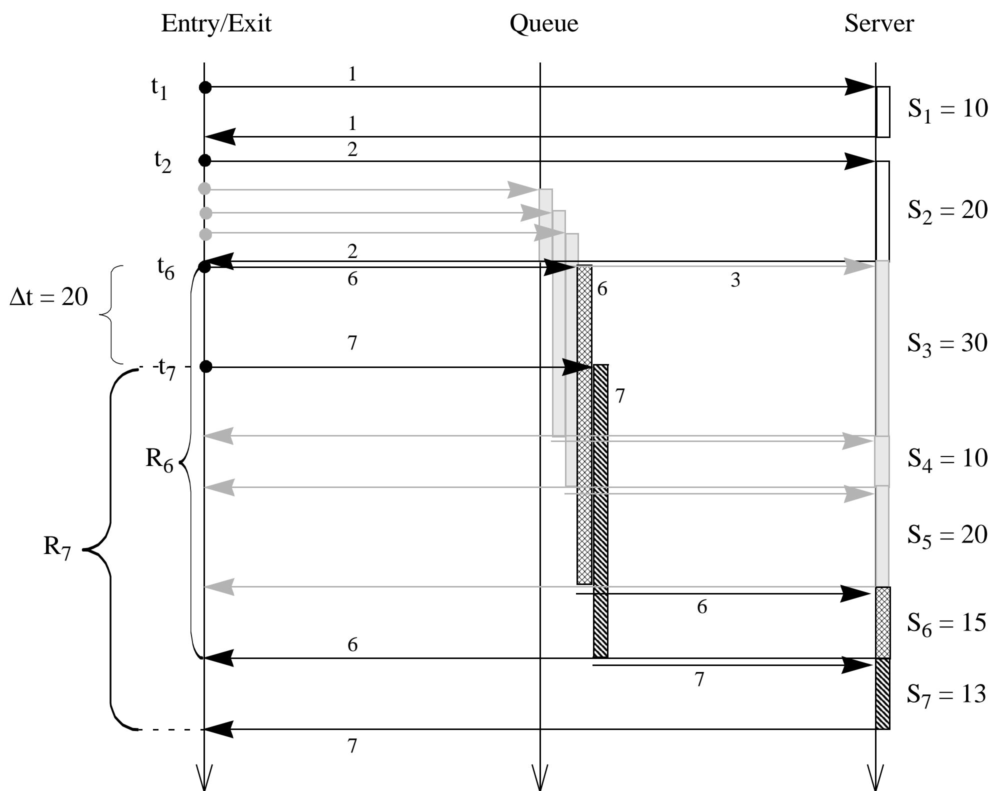

Figure 4-6: **Example of the calculation of FIFO server residence time**

$R_1 < \Delta t$, hence, the residence time of job 2 R_2 is equal S_2. This is logical, since job 2 arrives 15 time units after the arrival of job 1 at an empty system. S_2 may be 20. This scenario may go on until it develops the situation shown in figure 4-5: job n-1 is job 6 which arrives at t_6 just when job 3 moves into the server. It shall be $S_3 = 30$, $S_4 = 10$, $S_5 = 20$ and $S_6 = 15$. So, $R_6 = 75$. Job n (= job 7) arrives at t_7, 20 after t_6, i.e. $\Delta t = 20$. S_7 shall be 13. The residence time of job 7 is computed by $R_7 = S_7 + R_6 - \Delta t = 68$. If Δt was larger than S_3 it would reduce the residual service time of job 4 and perhaps of job 5, etc., respectively. Altogether, it reduces the sum of service times of jobs in the station which R_6 is composed of.

This way, with *CFS* agents are just delayed by the computed residence times. No resources are acquired. This means in terms of *JavaDEMOS*, that agents which arrive at such a server call the method `hold(`R_n`)`. No acquire of `Res` objects is done which would increase administration overhead.

Furthermore, it is possible to specify *CFS* as load dependent servers. This can e.g. be used to model overhead at the servers which arises when multiple agents reside. Service rates determine the speed the residing agents are served with. The service time of an agent is calculated by its service amount divided by the service rate according to the number of agents which reside at the server at its arrival.

4.3.2 Error Estimation

CFS preserves the residence times of mobile agents at agent servers and network, thus it preserves the round trip time. Furthermore, the throughput of mobile agents through the whole agent system remains unchanged.

But, as processes at the agent server and network are not simulated, server utilisation (i.e. I/O device utilisation) during a time period T cannot be observed as in the detailed model. The utilisation law [29], p. 556 et seq., helps here. Utilisation can be calculated as: $U_{A_i} = X_{A_i} \cdot S_{A_i}$, where

U_{A_i} = utilisation of agent server A_i,

X_{A_i} = throughput at agent server A_i,

S_{A_i} = mean service time at agent server A_i.

The mean service time and the throughput of mobile agents have to be surveyed just at the end-of-service of an agent. But, as described above, with *CFS,* end-of-service events are not modelled. In fact, the end-of-service timestamps lie within the accumulated delay for server and network residence time of the mobile agent (see figure 4-7).

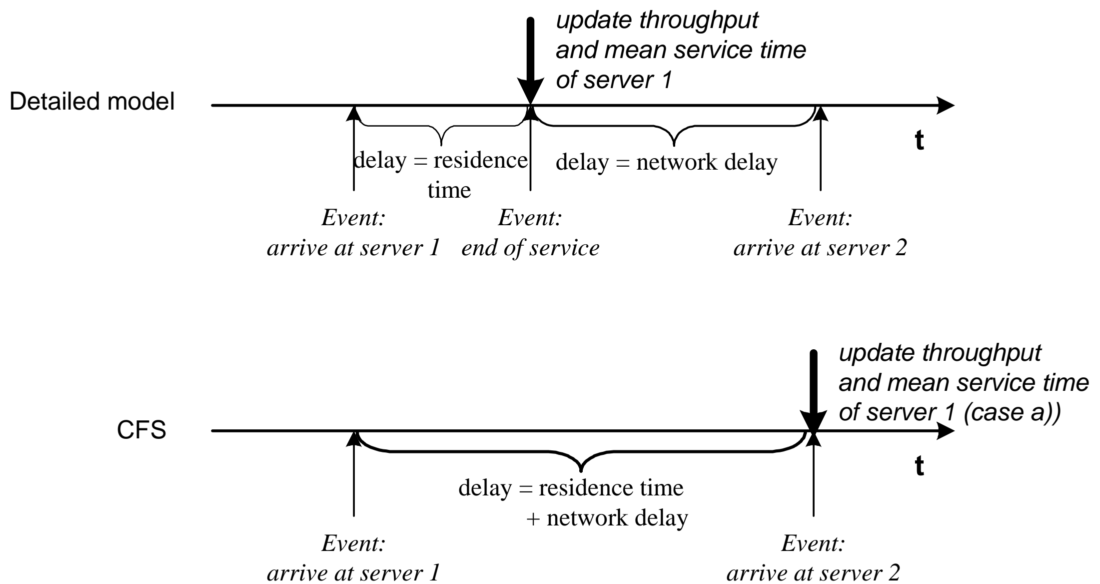

Figure 4-7: **Update of throughput and mean service in detailed model and CFS**

In *CFS*, the service end can be noticed at one of the following events:

a) When the mobile agent arrives at the next server,

b) or if the same service agent is directly requested by another mobile agent after its service for the previous one.

With a) service time and throughput are updated too late (delayed by the network delay). Thus, the server utilisation can only be calculated approximately. Depending on the bandwidth of the network and on the data volume of the mobile agent, i.e. depending on the network delay, the approximation error is bigger or smaller. Since we assume that mobile agents' data volume is not very large, we therefore assume that usually, network delays are quite small. This should minimise the error. With b) service time and the number of finished agents are registered in time, thus there is no approximation error concerning utilisation.

When modelling load dependent service rates, it has to be noticed that the service rate is selected according to the current number of agents. This number is only a snapshot at agent's arrival at the server. One could expect that the mean number of agents during an agent's residence at the server is used. This is not the case, because this value is not known when the agent's service time is calculated. The service time and, thus, its residence time is calculated directly at its arrival at the server.

4.4 Empirical Evaluation of CFS

4.4.1 An Introducing Example

CFS saves accesses to the event list, server queues and to the queue for waiting mobile agents. The corresponding states of a detailed and a *CFS* model shall be shown by an example with two service agents at an agent server (see figure 4-8 and figure 4-9). M_{ij} describes a mobile agent which requests service agent S_i and is numbered by j (to distinguish several mobile agents which request the same service agent S_i). The example starts at t = 0 where mobile agent M_{11} has sent its request to service agent S_1 which allocates the I/O device. Hence, M_{11} waits for response from S_1. Service agent S_2 is still idle. S_1 is served until t = 16. In the detailed model this state results in an event list consisting of the end-of-service event for S_1. M_{11} resides in the mobile agent queue. In *CFS* the time when M_{11} arrives the next server can already be calculated (t = 16+, t = 16 plus network delay of M_{11}). It is composed of the service time of S_1 and the network delay of M_{11}. Thus, the event list gets an entry for the arrival of M_{11} at the next agent server. Data structure $S_{i_{\text{idle}}}$ is updated. M_{11} immediately gets its result from S_1, hence, it does not have to wait for response in the queue.

At t = 1 M_{21} arrives at the server. The corresponding service agent S_2 is put in the queue of the device. As S_2 is currently idle it is sure, that it will be the next agent served. The service amount of S_2 is 9. Hence, its end-of-service time and thus the departure time of M_{21} is computable in *CFS*. The event list gets a corresponding entry at t = 25+ (departure time of M_{21} plus network delay of M_{21}). Data structure $S_{i_{\text{idle}}}$ is updated: S_2 is now busy until t = 25.

At t = 2 M_{22} arrives. The corresponding service agent S_2 is still in the device queue. Thus, the departure time of M_{22} cannot be calculated, yet. This means for *CFS*, that M_{22} first has to wait for S_2 being idle again. Hence, S_2 is put into the event list at its service end. This activity does not chance the data structures in *CFS*.

	System description	*Detail model*	*CFS*
t = 0	wait for response: M_{11} device: S_1 idle S_i: S_2	wait for response: M_{11} event list: 16 / S_1 device queue: (empty)	wait for response: (empty) event list: 16+ / M_{11} S_{iidle}: $S_1 = 16$, $S_2 = 0$
t = 1	wait for response: $\mathbf{M_{21}}$ M_{11} device: $\mathbf{S_2}$, S_1 idle S_i: (empty)	wait for response: $\mathbf{M_{21}}$ M_{11} event list: 16 / S_1 device queue: $\mathbf{S_2}$	wait for response: (empty) event list: 16+ / M_{11} → **25+** / $\mathbf{M_{21}}$ S_{iidle}: $S_1 = 16$, S2 = **25**
t = 2	wait for response: $\mathbf{M_{22}}$ M_{21} M_{11} device: S_2, S_1 idle S_i: (empty)	wait for response: $\mathbf{M_{22}}$ M_{21} M_{11} event list: 16 / S_1 device queue: S_2	wait for response: $\mathbf{M_{22}}$ event list: 16+ / M_{11} → **25** / $\mathbf{S_2}$ → 25+ / M_{21} S_{iidle}: $S_1 = 16$, $S_2 = 25$
t = 3	wait for response: $\mathbf{M_{12}}$ M_{22} M_{21} M_{11} device: S_2, S_1 idle S_i: (empty)	wait for response: $\mathbf{M_{12}}$ M_{22} M_{21} M_{11} event list: 16 / S_1 device queue: S_2	wait for response: $\mathbf{M_{12}}$ M_{22} event list: **16** / $\mathbf{S_1}$ → 16+ / M_{11} → 25 / S_2 → 25+ / M_{21} S_{iidle}: $S_1 = 16$, $S_2 = 25$

Figure 4-8: **Detailed model versus CFS (1)**

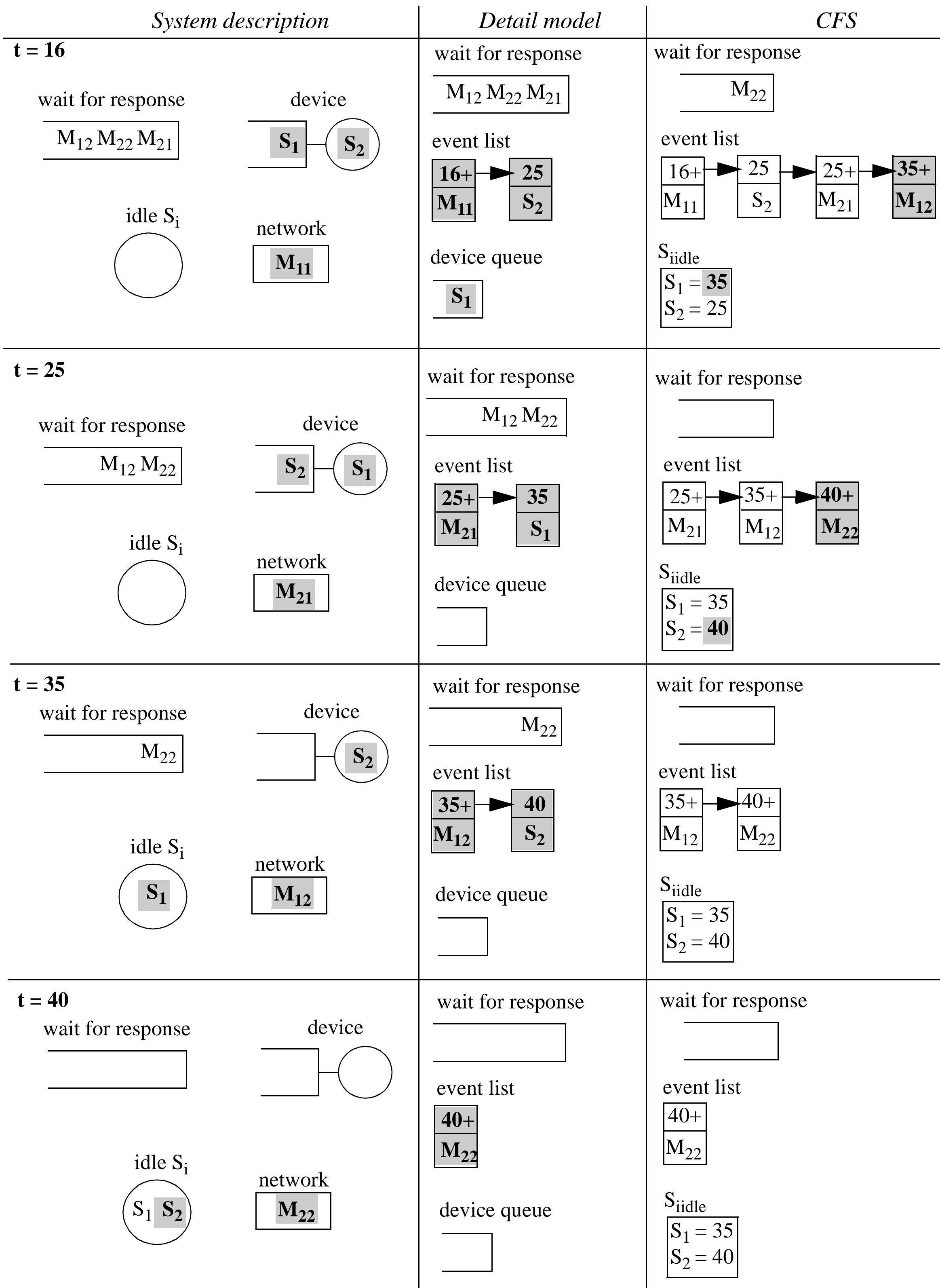

Figure 4-9: **Detailed model versus CFS (2)**

At t = 3 M_{12} arrives. The corresponding service agent S_1 is still in service. Thus, the departure time of M_{12} cannot be calculated, yet. This means for *CFS*, that M_{12} first has to wait for S_1 being idle again. Hence, S_1 is put into the event list at its service end. This activity again does not change the data structures in *CFS*.

At t = 16 the service time of S_1 ends. S_2 is now served. The detailed model gets a corresponding end-of-service entry into the event list. S_1 is put again into the device queue, because it is requested by M_{12}. Further, the arrival of M_{11} at the next agent server is inserted at t = 16+ (t = 16 plus network delay of M_{11}). S_1 now has - according to the request of M_{12} - a service time of 10. This sets the time when S_1 is idle to t = 35. In *CFS* the departure time for M_{12} is now computable, thus, at t = 35+ its arrival at the next agent server is inserted into the event list. Further, M_{12} gets its result and is deleted from the mobile agent wait queue. Data structure $S_{i_{\text{idle}}}$ is updated.

At t = 25 the service time of S_2 ends, M_{21} moves to the network and S_1 is served at the device. S_2 is requested by M_{22}, so it moves into the device queue. Respectively, in the detailed model M_{21} is put into the event list at its arrival at the next server (t = 25+) and the end-of-service event of S_1 is inserted. In *CFS* the residence time of can M_{22} be calculated, so its arrival at the next server at t = 40+ is inserted into the event list. Data structure $S_{i_{\text{idle}}}$ is updated.

At t = 35 the service time of S_1 ends and M_{12} moves to the network. The device is now utilised by S_2. S_1 is idle again. In the detailed model M_{12} is put into the event list at its arrival at the next server (t = 35+) and the end-of-service event of S_2 is inserted. The state of *CFS* is only changed in the case that the event at t = 25+ has been deleted from the event list.

At t = 40 the service time of S_2 ends and M_{22} moves to network. The device is idle and the agent waiting queue is empty. S_2 is idle as well. In the detailed model M_{22} is put into the event list at its arrival at the next server (t = 40+). The state of *CFS* is only changed in that case that the event at t = 35+ has been deleted from the event list.

This example shows the potential of *CFS* to increase simulation performance: the queue for the waiting mobile agents is hardly accessed, the device queue is not necessary, and even if the event list is usually larger than the one in the detailed model, the number of accesses to the list in *CFS* is 26.7% less than in the detailed model (11 accesses versus 15 accesses for insert and delete of events).

In the following sections, the *CFS* approach is compared empirically with detailed simulation models in several experiments. A detailed file server is modelled by a `Res` object of *JavaDEMOS*. Detailed models of all agent servers are compared to models where all agent servers are *CFS*. First, a steady state analysis is done. Then, the dynamic aspect of an agent system is analysed by a finite horizon simulation terminating after 80,000 seconds simulated time under varying workload. Here, performance values are observed along the time axis. In addition to the performance gains, it is shown how *CFS* results comply with the detail model results.

4.4.2 Steady State Analysis

Figure 4-10 shows the modelled agent system. One workload source generates mobile agents at a home server. The mobile agents travel through the agent system consisting of 19 further agent servers. Each mobile agent visits each of the agent servers in random order and requests service

from 1 of 4 service agents which reside at the agent servers. The service agents have mean service times of 1.0, 2.0, 4.0 and 5.0 seconds. It is assumed that there is no delay at agent home.

First, the system is analysed with a Poisson arrival stream of mobile agents. The coefficient of variation of service times is 1.0, the distribution of service times is negative exponential. The number of accesses to the several service agents is uniformly distributed. Summarising, this agent system corresponds to a network consisting of $M/H_4/1$ stations with FIFO scheduling discipline for service agents. Mobile agents are not necessarily served FIFO. This depends if their service agent is idle at their arrival and, thus, can be put into the device queue according to their arrival order.

In the next experiments, interarrival times of mobile agents follow a Cox-2 distribution[1] with a coefficient of variation (c.o.v.[A]) of 3.0 and service times per service agent are set deterministic. Thus, the service time distribution at the agent servers is similar to H_4, with the difference that the each of the 4 service phases are not negatively exponential distributed but are deterministic. This kind of service time distribution shall be abbreviated D_4. The agent system corresponds to a network consisting of $G/D_4/1$ stations with FIFO scheduling discipline for service agents.

Finally, interarrival times of mobile agents have a coefficient of variation (c.o.v.[A]) of 3.0 and the coefficient of variation of service times per service agent i (c.o.v.[S_i]) is set to 5.0 (implemented again by the Cox-2 distribution). Summarising, this agent system corresponds network consisting of G/G/1 stations with FIFO scheduling discipline for service agents.

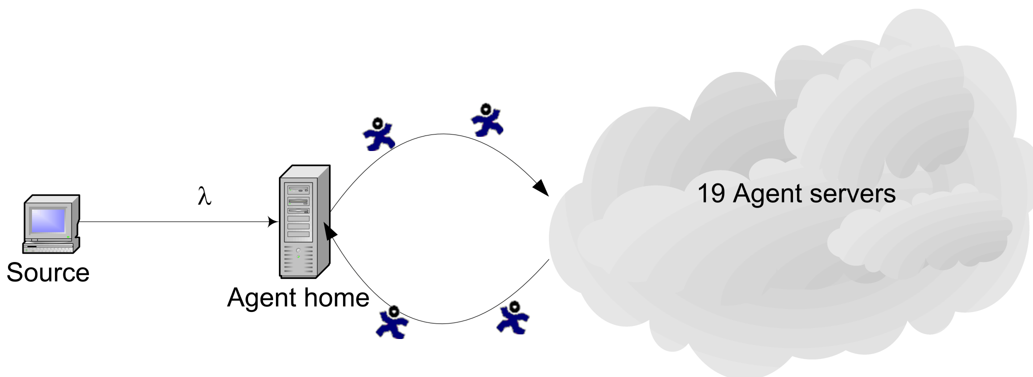

Figure 4-10: **Modelled agent system**

Validity of CFS

Results of detailed and *CFS* models are very close, because the algorithms are functionally equal and in most cases the models use common random numbers (see [36], p. 582 et seq.)

Figure 4-11 shows the utilisation of a dedicated agent server "Bond". Utilisation coincides in all scenarios between detailed and *CFS* models, i.e. approximation errors are not recognisable. Figure 4-12 shows the 90% confidence intervals of the round trip time (*CI RTT*) in the G/G/1 network[2], and table 4-1 illustrates the 90% paired-t confidence intervals of differences between detailed and *CFS* models at several mean mobile agent arrival rates λ. Differences are very small

1. Assumption: both phases are evenly utilised, i. e., $\mu_2 = a_1 \cdot \mu_1$, where μ_1 and μ_2 are the service rates of the two phases and a_1 is the probability to pass through phase 2 after phase 1.

with $G/D_4/1$ and $M/H_4/1$ scenarios. The corresponding confidence intervals include zero in most cases, i.e., the hypothesis that detailed and *CFS* models represent the behaviour of the same system cannot be rejected. With G/G/1 stations, *paired-t* confidence intervals of differences sometimes are very large. This is due to the high variance in differences and the relatively small sample size resulting from the sequential simulation method which is adjusted to the round trip time, not to the batch differences. But again, zero is always included in the confidence interval, thus, models may represent the same system. Furthermore, the half widths of the large confidence intervals with $\lambda = 0.20$ and $\lambda = 0.27$ are 4.6%, resp. 5.1% of the mean value of the corresponding round trip times which is not very high.

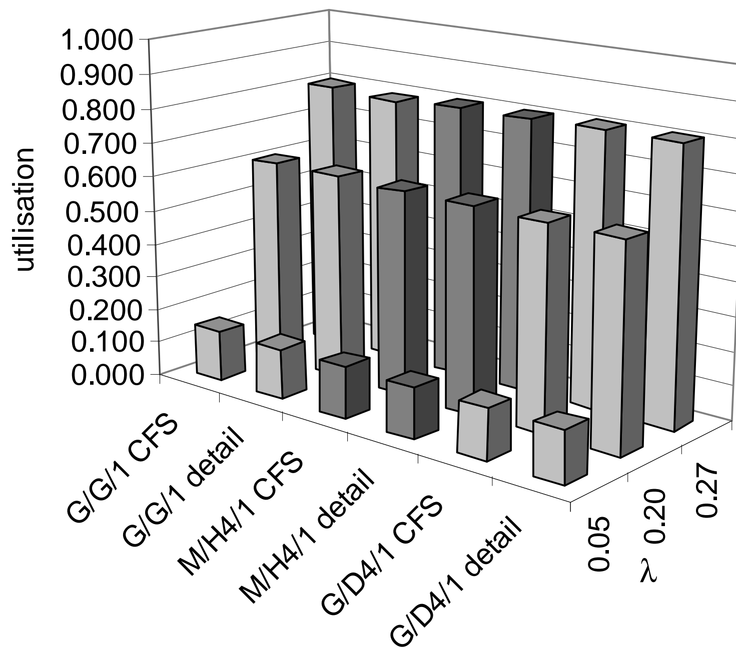

Figure 4-11: **Utilisation of agent server "Bond" in detailed and CFS models**

2. Confidence intervals of round trip times of all model types can be found in Appendix A.2.

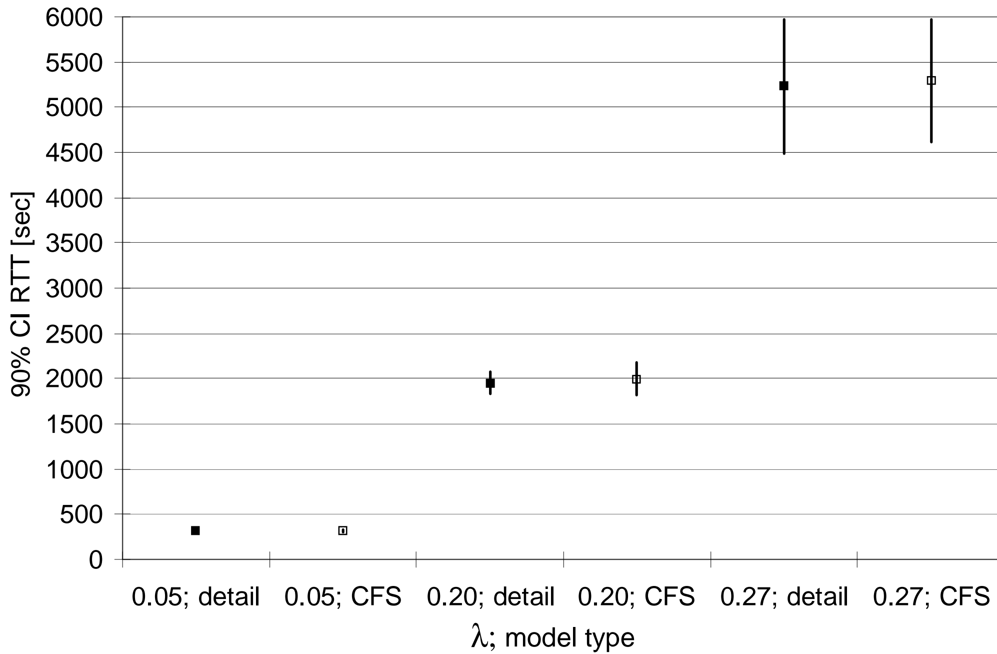

Figure 4-12: **90% confidence intervals of round trip times of detailed and CFS G/G/1 models**

Station types	λ [ma/sec]	paired-t confidence interval (90%)
$G/D_4/1$,	0.05	[-0.009, 0.001]
c.o.v.[A] = 3.0	0.20	[-0.223, 0.564]
	0.27	[1.387, 3.615]
$M/H_4/1$	0.05	[0.000, 0.000]
	0.20	[0.000, 0.000]
	0.27	[0.000, 0.000]
G/G/1,	0.05	[-4.913, 2.673]
c.o.v.[A] = 3.0,	0.20	[-49.887, 130.079]
c.o.v.[S_i] = 5.0	0.27	[-202.128, 334.840]

Table 4-1: **Paired-t confidence intervals of differences between CFS and detailed model**

Efficiency Gains

Figure 4-13 compares the model efficiency at several mobile agent arrival rates λ. It shows absolute values of CPU time consumption. These absolute values are converted into efficiency gains in table 4-2. It is evident, that the approach of *CFS* decreases the CPU time consumption significantly with varying workload. Values are averaged over 10 simulation runs.

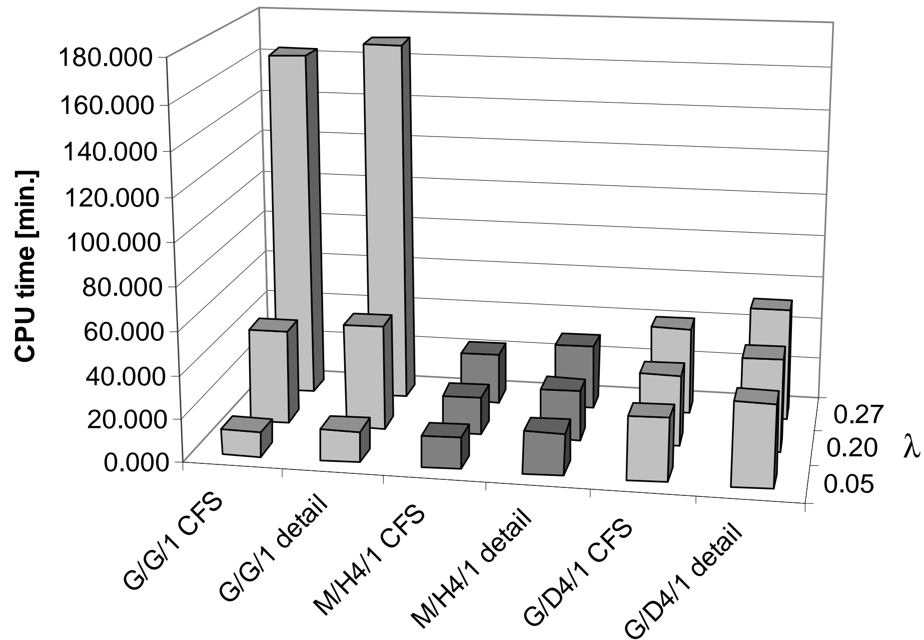

Figure 4-13: **CPU time consumption of detailed and CFS models**

In the G/G/1 network, efficiency gains decrease noticeably with increasing λ. Nevertheless, gains by *CFS* are still significant. In the $M/H_4/1$ and $G/D_4/1$ scenarios (which possess a smaller c.o.v. of service time at the agent servers) this effect is not observable such clearly. This is due to the fact that in the less efficient scenarios the number of agents which first have to wait for their service agent to be idle is higher than in the scenarios where *CFS* is more efficient. Thus, the accesses to the event list for the first phase of mobile agents' residence time (see section 4.2.2) overlay the performance gains reached by the efficient modelling of service agent processing.

Station types on PC	λ [ma/sec]	Gains by CFS
$G/D_4/1$, c.o.v.[A] = 3.0	0.05	23.8%
on *blofeld*	0.20	23.2%
	0.27	21.6%
$M/H_4/1$	0.05	25.9%
on *goldfinger*	0.20	24.8%
	0.27	22.1%
G/G/1, c.o.v.[A] = 3.0,	0.05	20.5%
c.o.v.[S_i] = 5.0	0.20	9.8%
on *trinity*	0.27	3.9%

Table 4-2: **Comparison of efficiency of CFS and detailed model**

4.4.3 Finite Horizon Analysis

The finite horizon analysis investigates the dynamic behaviour of mobile agent systems, i.e. the changes of performance values along the time axis. 80,000 seconds (ca. 28 hours) are modelled. Until simulated time of 20,000 seconds a single source generates mobile agents with rate λ = 0.05 agents per second. Then, 5 further sources are activated which send out agents with the same rate each, until 60,000 seconds of simulated time. Finally, 5 sources are switched off, thus, there is again a single source which generates mobile agent with a rate of 0.05 agents per second. All sources use the same agent home server. It is assumed that there is no delay at agent home. Simulation starts with an empty system. Figure 4-14 shows the modelled agent system.

As with the steady state analysis, 4 service agents reside at each agent server. The service agents possess mean service times of 1.0, 2.0, 4.0 and 5.0 seconds. First, a network of $M/H_4/1$ stations with FIFO scheduling discipline is modelled.

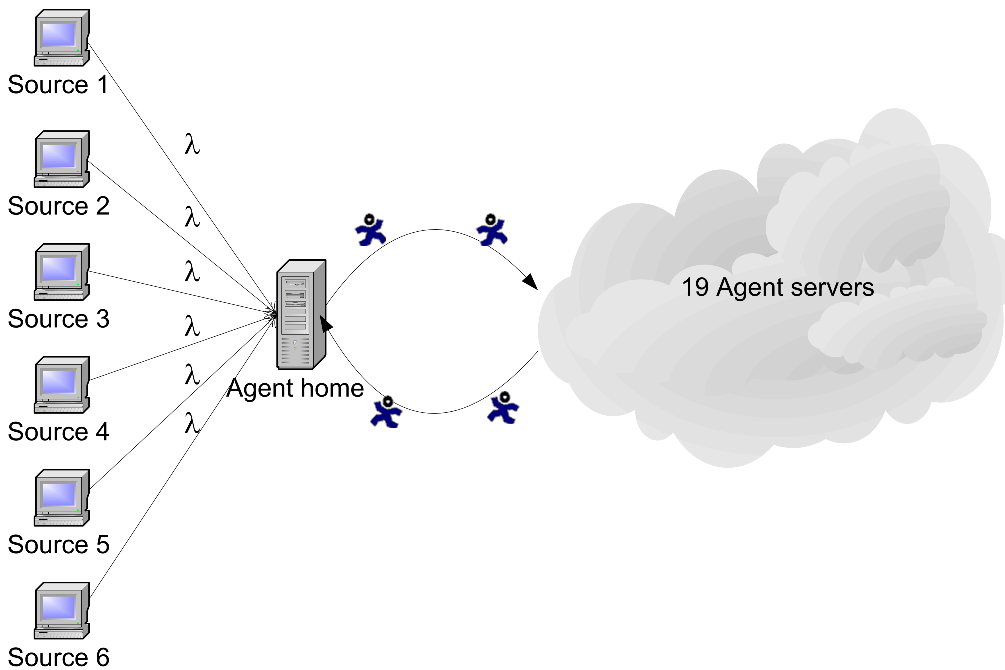

Figure 4-14: **Modelled agent system**

Next, a network of $G/D_4/1$ stations with FIFO scheduling discipline is analysed. Finally, a network of G/G/1 stations with FIFO scheduling discipline is investigated. For a detailed description of the scenarios refer to section 4.4.2, page 59.

Validity of CFS

100 replication runs are necessary for the G/G/1 and G/D/1 network to achieve the threshold for the relative statistical error of 0.15 of mean round trip times with all confidence intervals (all but one outlier). 10 replications are necessary for the $M/H_4/1$ network. The method of common random numbers (see [36], p. 582 et seq.) is used, thus indeed, results of detailed and *CFS* models are very close.

Figure 4-15 shows the variation of round trip times in the G/G/1 network. Figure 4-16 shows the development of the throughput, i.e. the number of agents within 4000 seconds which arrive back home. Both figures show significantly the impact of the workload enhancement between 20,000 and 60,000 seconds. Interestingly, at the end of the heavy workload phase throughput increases once again significantly. This can be explained by the crowded server sources, which still send out many jobs after the end of the heavy workload phase. Differences between the behaviour of the two model types are hardly recognisable.

Also, the empirical distribution of round trip times (see figure 4-17) shows the same behaviour with the detailed and *CFS* model. To confirm the apparent validity of *CFS*, table 4-3 shows the 90% paired-t confidence intervals of differences of mean round trip times resulting from the G/G/1 network model. The confidence intervals all include zero, thus, the hypothesis that both models represent the behaviour of the same system cannot be rejected. Furthermore, table 4-3 includes the differences of mean values of both model types in percent. These do not clearly exceed 3%.

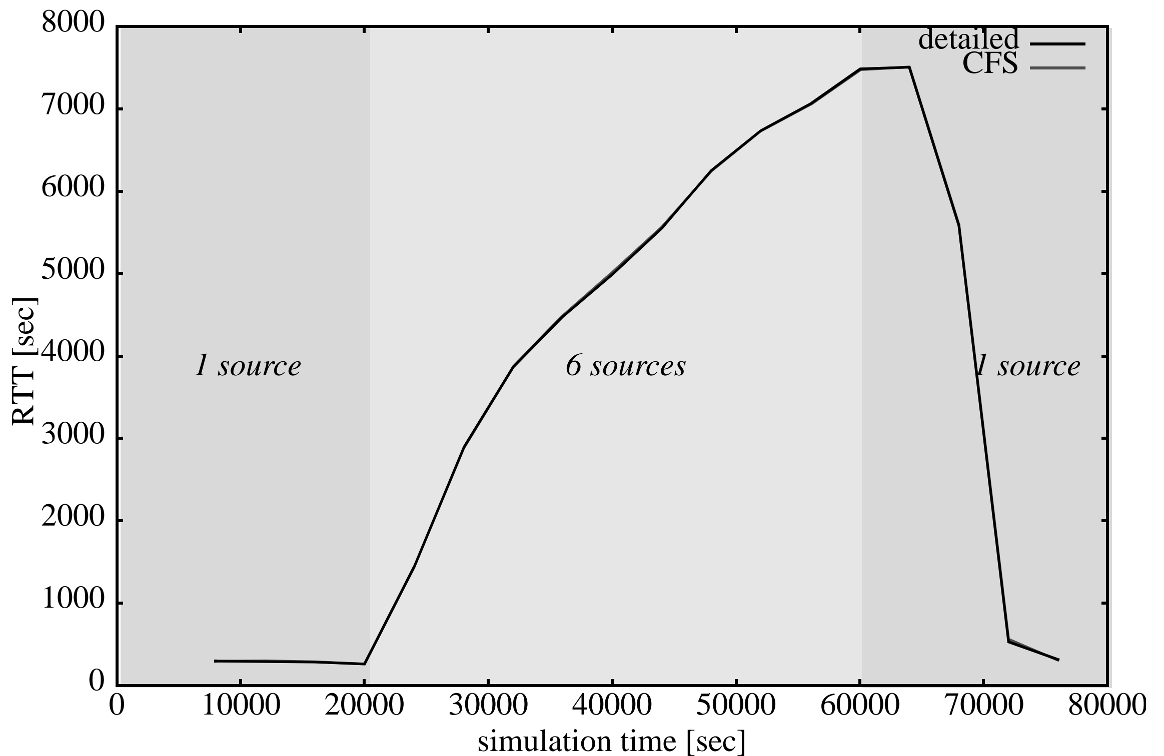

Figure 4-15: **Round trip times in an G/G/1 network (mean values over 100 runs)**

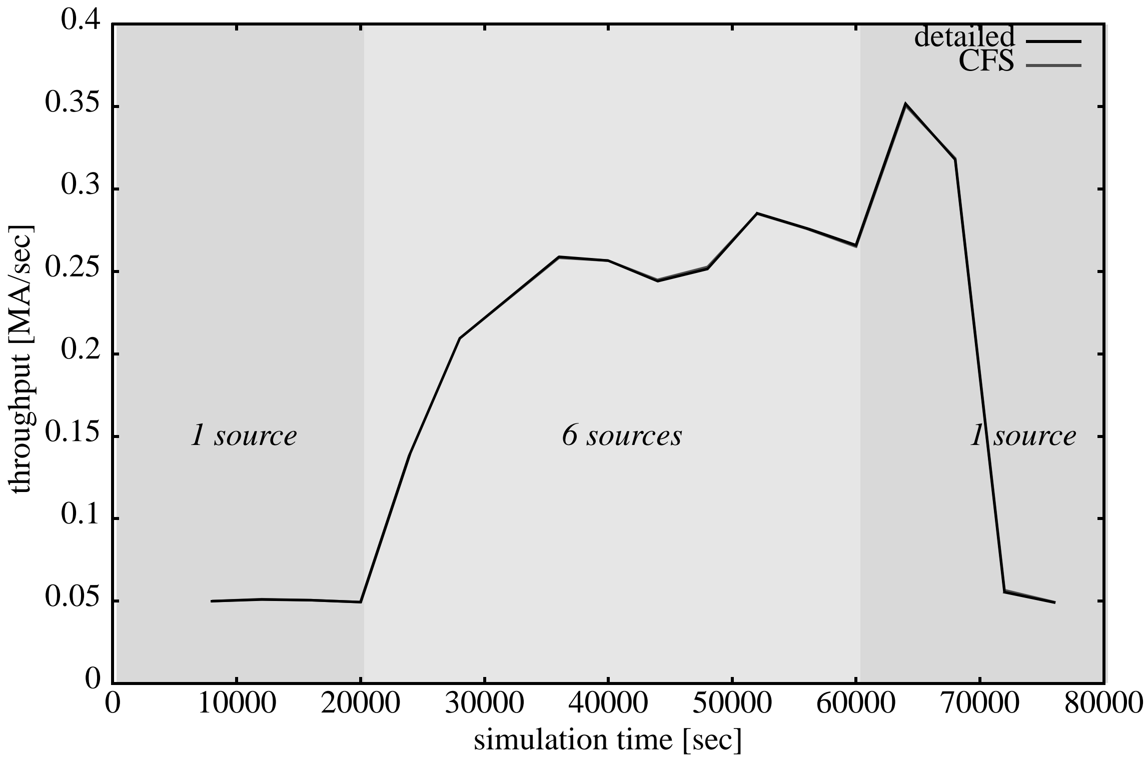

Figure 4-16: **Throughput in an G/G/1 network (mean values over 100 runs)**

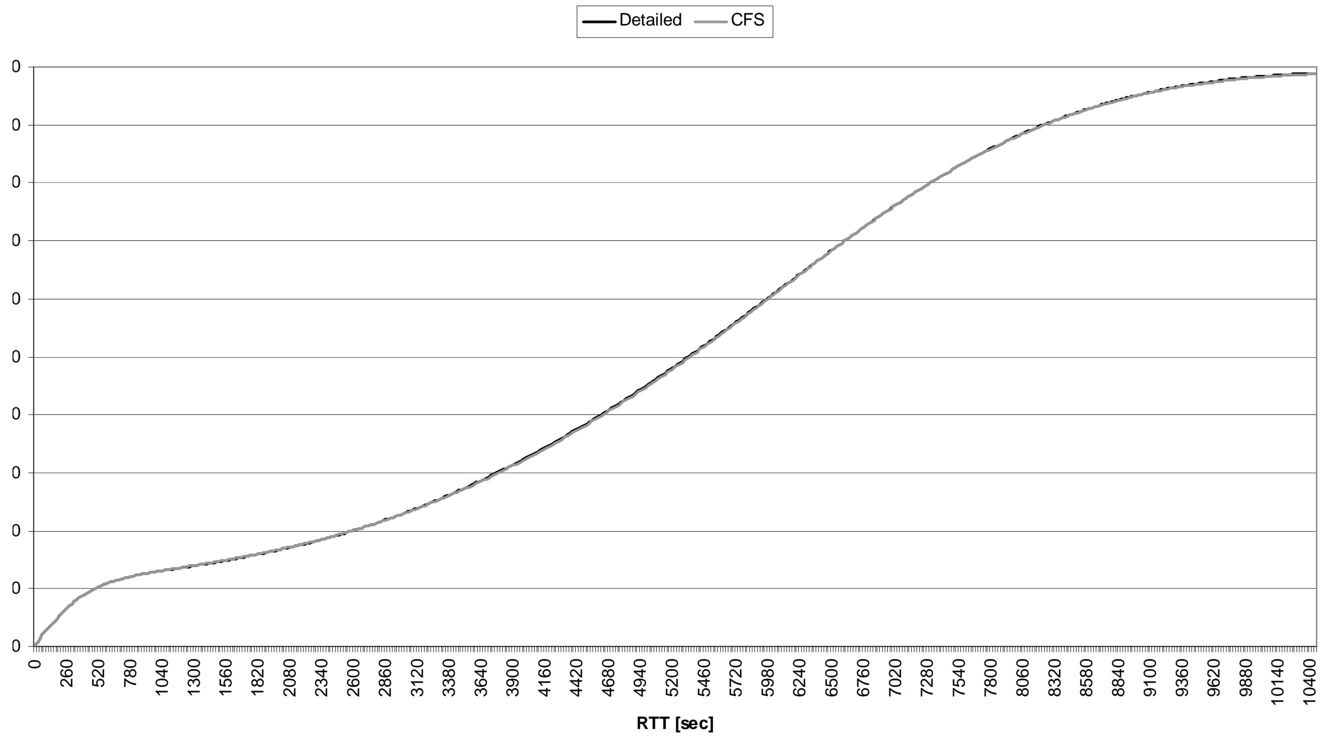

Figure 4-17: **Relative frequency of RTT in an G/G/1 network (mean values over 100 runs)**

model time [sec]	paired-t confidence interval (90%)		% differences of mean values
8000	-4.676	3.274	-0.238
12000	2.199	15.521	3.062
16000	-2.718	9.237	1.148
20000	-6.586	3.137	-0.663
24000	-16.629	2.059	-0.505
28000	-21.514	11.039	-0.181
32000	-23.710	23.506	-0.003
36000	-22.647	44.701	0.246
40000	-13.091	60.632	0.476
44000	-23.629	56.782	0.299
48000	-49.536	31.866	-0.141
52000	-46.674	38.468	-0.061
56000	-51.268	34.715	-0.117
60000	-64.605	39.790	-0.166
64000	-55.873	62.922	0.047
68000	-58.752	87.379	0.257
72000	-35.490	99.181	6.017
76000	-16.995	0.743	-2.598

Table 4-3: **Paired-t confidence intervals of differences between CFS and detailed model (based on 100 replication runs)**

Figure 4-18 through figure 4-20 show the development of utilisation of the exemplary agent server "Bond". Mobile agents are modelled with different data volumes to analyse the impact to the approximated calculation of utilisation. Results are again averaged over 100 replication runs. 150 KB is an average size for mobile agents which are usually several kilobytes large. The variation of the utilisation of server "Bond" maps sufficiently in both model types (see figure 4-18). Even with very large mobile agents of 2 MB (see figure 4-19) and 5 MB (see figure 4-20) the values of utilisation are very similar. Mobile agents with a size of 5 MB are quite unrealistic, as the concept of mobile agents was developed originally to save network bandwidth and to shift calculation and data processing to the agent servers.

Results for the finite horizon analysis of the $M/H_4/1$ and $G/D_4/1$ networks can be found in Appendix A.3 and Appendix A.4. Generally, they show the same match between detailed models and *CFS* as it is the case with the G/G/1 network.

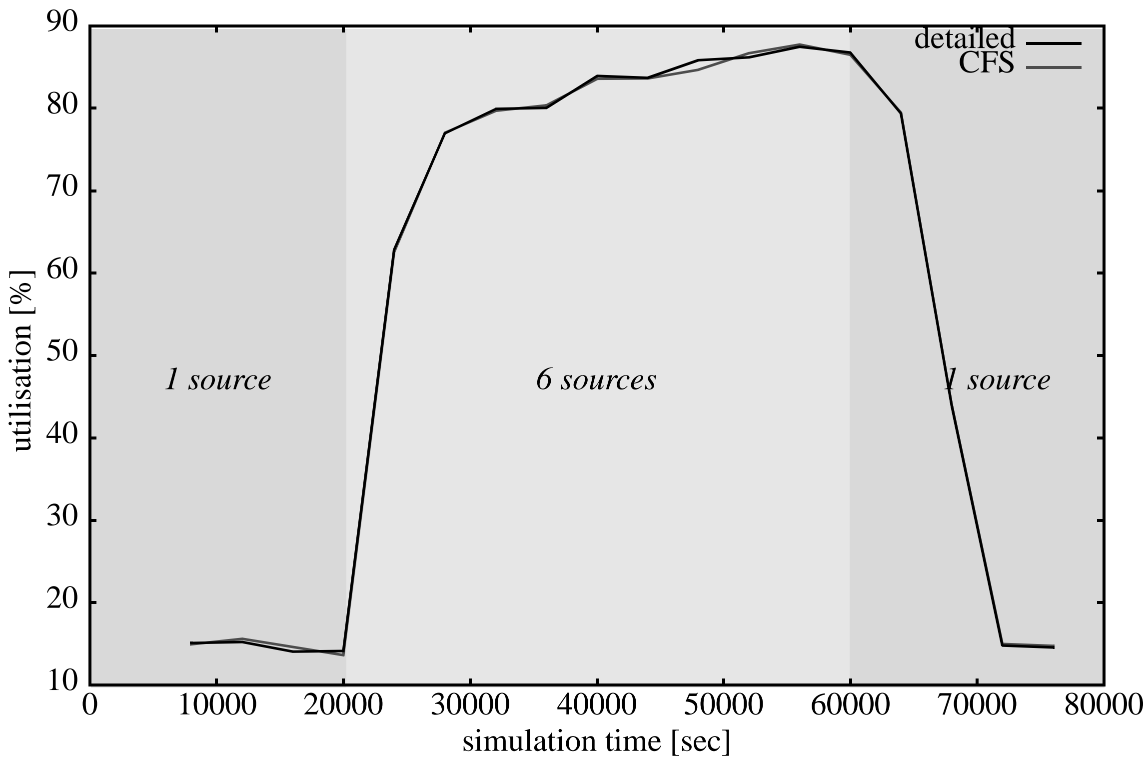

Figure 4-18: **Utilisation of server "Bond" with mobile agents of 150 KB**

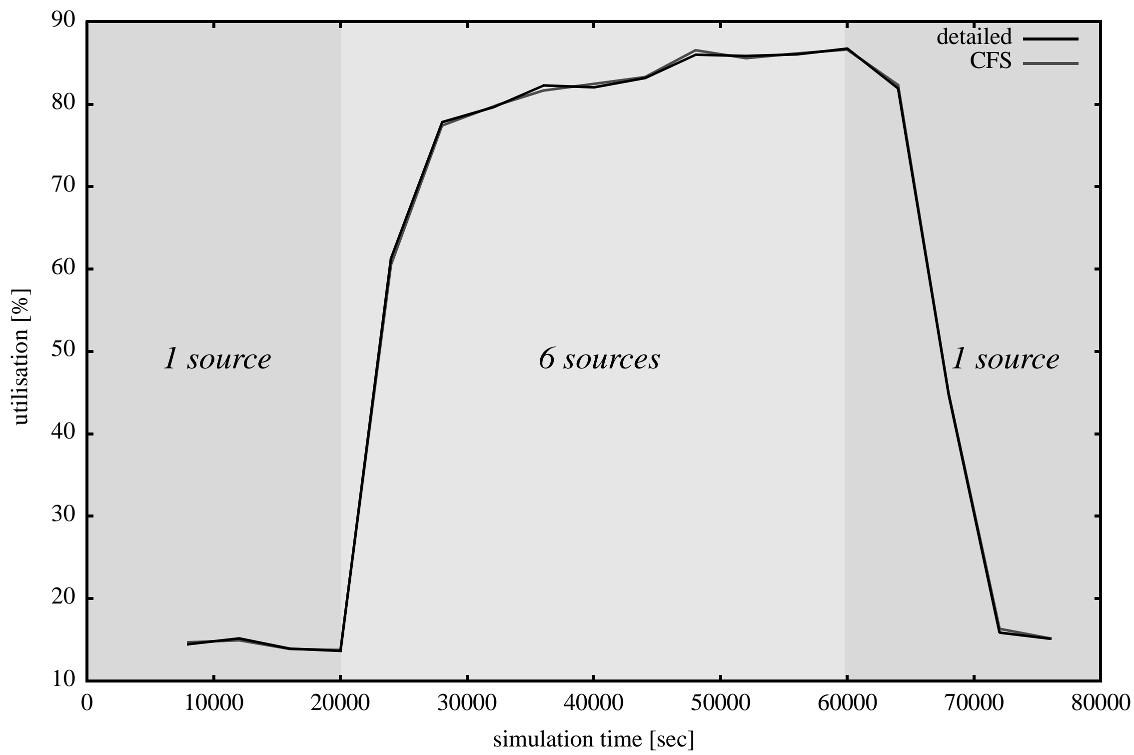

Figure 4-19: **Utilisation of server "Bond" with mobile agents of 2 MB**

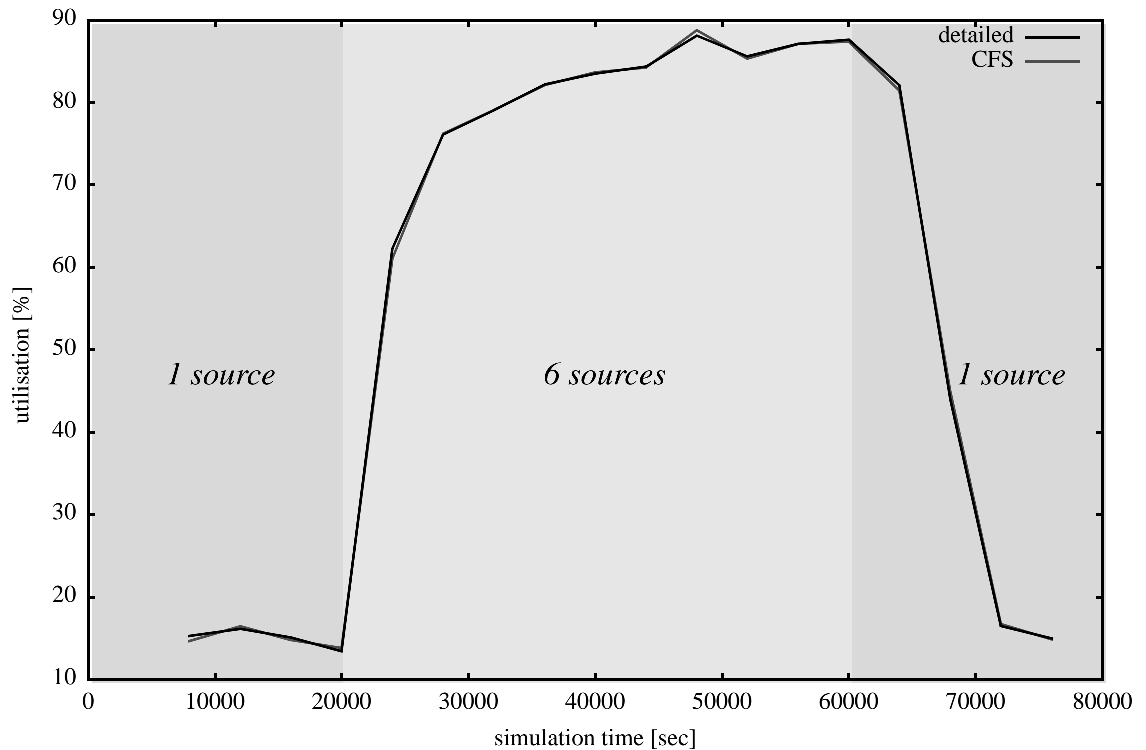

Figure 4-20: **Utilisation of server "Bond" with mobile agents of 5 MB**

Efficiency Gains

Figure 4-21 and table 4-4 compare the duration of the execution time for a simulation run (CPU time). Values are averaged over 100 replication runs for the G/D_4/1 and G/G/1 networks and averaged over 10 replication runs for the M/H_4/1 network. Figure 4-21 shows absolute values of CPU time consumption. In table 4-4, these values are converted into efficiency gains of CFS compared to the detailed models.

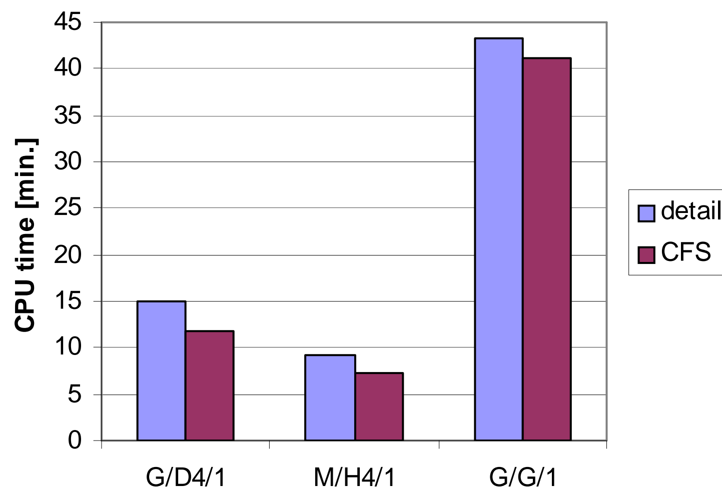

Figure 4-21: **Average amount of CPU time per simulation run**

It is evident, that the *CFS* system improves simulation efficiency significantly, i.e. up to 21.7%. Similar to the results of the steady state analysis, the efficiency gains of *CFS* in the G/G/1 network are quite small. The decreasing efficiency gains can again be explained by the increasing number of mobile agents which first have to wait for their service agent to be idle and, thus, have to be inserted into the event list. This overlays the efficiency gains achieved by the efficient modelling of service agent processing.

Station types on PC	Efficiency gains of CFS (decrease of CPU time)
$G/D_4/1$, c.o.v.[A_i] = 3.0 on *blofeld*	21.7%
$M/H_4/1$ on *goldfinger*	21.6%
G/G/1, c.o.v.[A_i] = 3.0, c.o.v.[S_i] = 5.0 on *trinity*	5.2%

Table 4-4: **Efficiency gains of CFS**

4.5 Concatenated Round Robin Servers (CRRS)

Another type of dedicated agent servers are compute servers which mainly provide CPU power for the agents. CPUs usually schedule jobs in round robin mode. All residing jobs are served cyclic and are given a fixed piece of processing time (time slice or quantum) one after another. A job has finished if the sum of slices it got accumulates to its service time. Waiting time arises when the processor serves other jobs. A typical value for a time slice is 100 milliseconds. This means, e.g., a job with a service time of 10 seconds has to get 100 slices before it has finished. One can imagine that modelling the round robin process in detail with its changing assignments of time slices is very inefficient.

Other ways have to be developed to model compute servers more efficiently. Huh [27][1] presents approaches to calculate residence time at round robin servers which traces back to Welch et al. [60] and Audley et al. [2]. These assume a number of applications which are allocated to a server, and which are repeated periodically. It is assumed that the length of periods for each application class is known. Analogue, in mobile agent systems there is a known number of applications allocated to a server, the service agents. But, they do not have to be executed periodically, furthermore, their execution depends on the requests of arriving mobile agents. Periods or arrival rates of mobile agents per server cannot be assumed apriori. Hence, this approach is not applicable to the modelling of mobile agent systems.

Another common method to simulate round robin servers more efficiently, is to calculate the effective speed at every arrival or departure of a job, i. e. every time the number of jobs changes.

1. The publication of Huh, available from http://zen.ece.ohiou.edu/john-thesis.html has to be read carefully and critically. Some inconsistencies can be found, e.g., between the formula for $D_{pred1}(A)$ on page 22 and the referring example. In case of doubt, [2] and [60] should be taken into account.

The effective speed for a job is defined by the speed of the processor divided by the mean number of jobs residing while the job is served. Hence, effective speed can as well be understood as a rate of progress for job execution. The residence time of job i is calculated by $\text{residence time (i)} = \frac{\text{service time (i)}}{\text{effective speed}}$. Whensoever the effective speed changes, i.e. when a new job arrives or if a job departs, the residence times of all jobs in the processor are calculated anew and the jobs are rescheduled according to their changing residence times. This method shall be called *rescheduling* technique.

The technique described below shows similarities to the *rescheduling* technique. However, it reduces the frequency of rescheduling of jobs and thus, it is more efficient: It does not only consider the current number of jobs at the processor, but also their service amounts and their estimated residence times. Rescheduling of jobs is not necessary at every arrival or departure of a job. This technique shall be called *reduced rescheduling*. The algorithm and efficiency gains are as well described in [17]. *Reduced rescheduling* is a hybrid approach. Scheduling of jobs by the processor is not simulated, but residence times of jobs are calculated and stored in a data structure.

The approach of *concatenated round robin servers (CRRS)* uses the *reduced rescheduling* technique to model the residence time of service agents on concatenated round robin servers.

4.5.1 Calculation of Service Agents' Residence Time at the Processor

According to the *reduced rescheduling* technique the residence time R_{S_i} of service agent S_i is first estimated based on the service agents which currently reside at the server. The ID's, service amounts and estimated departure times of these agents are stored in a data structure called *job list*. They are sorted with increasing (residual) service amounts. Arrivals of succeeding service agents during the residence of S_i are not taken into account within this first estimation. A requesting mobile agent is delayed for the estimation of R_{S_i} which is correct or too small, depending on the number of successively arriving service agents at the processor and their service time. After the first delay, the mobile agent checks if R_{S_i} has been corrected in the meantime. The mobile agent is then iteratively delayed until R_{S_i} is estimated correctly.

In detail, the *reduced rescheduling technique* proceeds as follows:

1. At every arrival of a service agent at the processor the departure times of all currently residing service agents (stored in *job list*) are calculated according to the following algorithm:

   (a) Calculate effective speed: $effectiveSpeed = \frac{speed}{n}$,

   where *speed* = speed of processor and *n* = number of service agents in the processor, i.e. in *job list* (including the agent which has just arrived).

The first service agent from *job list* (index 0) leaves the processor at $timeDep_0 = \text{current time} + \frac{resSA_0}{effectiveSpeed}$, where $resSA_0$ = residual service amount of the first service agent.

(b) For each further service agent in *job list* (indexes j, j = 1, ..., n) calculate the departure time by $timeDep_j = timeDep_{j-1} + \frac{resSA_j - resSA_{j-1}}{effectiveSpeed_j}$, where $effectiveSpeed_j = \frac{speed}{n-j}$.

When service agent S_i arrives at the processor, the requesting mobile agent is delayed by the residence time of S_i which is calculated as described above plus the network delay. After this first delay is elapsed, the mobile agent is again delayed if S_i's residence time has been corrected in the meantime.

2. When an agent arrives at or departs from the processor, the residual service amounts have to be corrected in *job list*. This is only an access to the data structure *job list*, no scheduling process in simulation. Of course, an efficient implementation of *job list* is essential, not to overlay the efficiency gains achieved by less scheduling activity.

The *reduced rescheduling technique* for calculating the residence time of service agents in the processor can also be applied to networks consisting of general round robin stations. Instead of service agents, arbitrary jobs can be served by the processor. [17] shows results of the simulation of a general queuing network.

Furthermore, servers modelled with *reduced rescheduling technique* can be specified as load dependent servers. This can be used e.g. to model overhead at the servers which arises when multiple agents reside. Service rates determine the speed of the server. At the calculation of residual service times of agents the server speed is set to the service rate according to the number of agents which currently reside.

4.5.2 Multi Processors

Compute servers with multiple processors are modelled as in the detailed model: multiple instances of *CRRS* are built. The service agents are allocated to a *CRRS* processor by a dispatcher which selects the processor with the currently lowest number of agents in service.

4.5.3 Error Estimation

For part 2. of the *reduced rescheduling* algorithm it is important to recognise the event when an agent (or in general, a job) leaves the processor. With concatenated servers, the departure of a service agent from the processor is not easy to recognise, because there exists no departure event in the simulation (compare figure 4-7). As known from section 4.3.2 with *CFS*, the service end is recognisable at one of the following events:

a) The corresponding mobile agent arrives at the next server,

b) the same service agent is directly requested by another mobile agent after its service for the previous one.

This implies certain approximation errors.

a) occurs especially with low utilisation. With a) the service end is noticed too late by the network delay. This does not affect the round trip time of mobile agents. It slightly influences utilisation, but, due to the small amount of network delay compared to the residence time at servers, it can be assumed that this impact in utilisation will not be recognisable in experiment results.

b) occurs especially with high utilisation. With b) the service end is noticed in time, but, the cumulative delay of the mobile agents (server residence time plus network delay) can be calculated too small. Consider the following example which is expressed in figure 4-22:

The estimated departure time of service agent S_1 and thus, for the corresponding mobile agent M_1 is t = 5.0. The network delay for M_1 is calculated to be 0.5 seconds. Hence, M_1 is cumulatively delayed until t = 5.5.

At time t = 5.3, S_1 definitively departs the processor and is requested directly by another mobile agent M_2. Thus, S_1 gets a new estimated departure time according to the service request of M_2. The new departure time now is bind to M_2. The correct cumulative delay of M_1 would be t = 5.3 + 0.5 = 5.8. But, M_1 checks the departure time of S_1 for updates not until t = 5.5. At this timestamp the modified departure time of S_1 is lost for M_1, because it is overwritten by a new value for M_2. Thus, M_1 reaches the next server too early by 0.3 time units.

A history list with last departure times of service agents could help to correct this error. For efficiency reasons, this is not implemented.

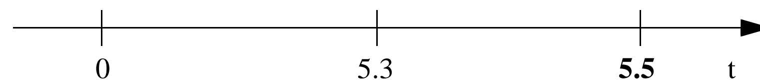

Figure 4-22: **Example for a too small cumulative delay**

Hence, the residence time $R_{M_j}(i)$ of mobile agent M_j at server i plus succeeding network delay is possibly too small by E, $0 \le E < \text{network delay}$. On the other hand, this increases the arrival

rate λ_{i+1} at the next server *i+1*, which results in a higher population at this server and, thus, it results in a higher residence time there. Hence in total, approximation error b) is neutral concerning round trip time. It slightly increases server utilisation. But, as very small network delays are assumed, it can be expected that this impact in utilisation will not be recognisable in experiment results.

A further approximation error dues to the fact that the order which jobs get the processor quantum in is not considered. Thereby, estimation of residence time is a bit too pessimistic with the *reduced rescheduling* technique. Server utilisation is not influenced. Consider the following example (figure 4-23):

At time t = 0, jobs A, B, C and D reside at the processor with service amounts of 3, 1, 1 and 2 time slices. Actually, the quantum is allocated to D first. After the time slice is elapsed, half of D is served, after 2 time slices C is finished, after 3 time slices B is finished and so on. For simplicity reasons, the time slice may correspond to 1 time unit. Finally, D gets the second quantum at t = 4, thus D is finished at t = 5. In the same way, A is finished at t = 7. With *reduced rescheduling*, the service order of the jobs is not recognised. The calculation again begins at t = 0. B and C have the smallest service amounts of 1, i. e. they will leave the processor first. During their residence time, processor capacity has to be shared with 4 jobs, thus, their departure time is calculated as $t + (1 \cdot 4) = 4$.

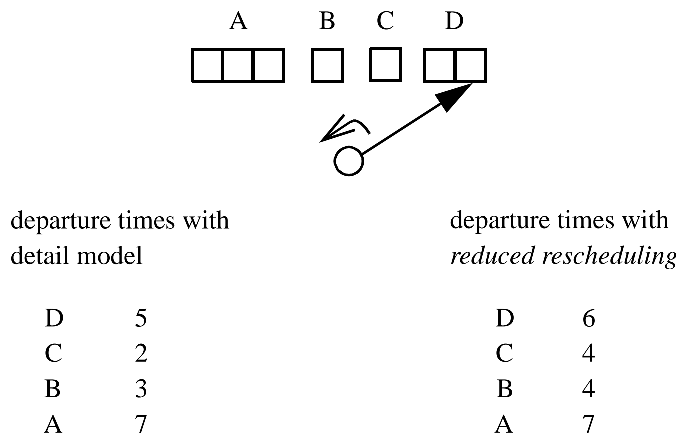

Figure 4-23: **Approximation error if not recognising quantum allocation**

Next, D is the job with the smallest service amount. At t = 4 there is 1 time unit of service amount of D left, this is served with the presence of 1 further job (A). Hence, D's departure time is calculated as $t + (1 \cdot 2) = 6$. In the same way, the departure time of A is calculated to be 7. Apart from A, the departure time of all jobs is overestimated. This way, the departure time is estimated slightly too high or correct with *reduced rescheduling*. An underestimation is not possible.

Summarising, the quality of the results for mobile agents' residence times and, thus, round trip times directly depends on the quality of the calculation of the service agents' residence times, as the mobile agents' residence time consists of $R_{M_j} = S_{i_{\text{idle}}} - t_{M_j} + R_{S_i}$. The process of waiting for an idle service agent (which results in the value $S_{i_{\text{idle}}}$) is modelled exactly and implies no further errors (see section 4.2.2). Based on the experiences with *CFS* and the additional considerations in this section, it can be assumed that approximation errors will affect significantly only

round trip time of mobile agents, because the allocation of the processor quantum is not recognised. This sometimes leads to a slightly pessimistic estimation of residence times at servers.

4.6 Empirical Evaluation of CRRS

To demonstrate efficiency gains of the *CRRS* approach it is compared with detailed simulation in several experiments which correspond to the scenarios modelled to analyse the efficiency of *CFS*. A detailed compute server is modelled by the `CPU` building block of *JaDEMAS* (see Chapter 2, section *Compute Servers*). Models with detailed simulated agent servers are compared to models where all agent servers are *CRRS*. First, a steady state analysis is done with the agent system. Then, dynamic aspects are analysed by a finite simulation terminating after 80,000 seconds simulated time under varying workload. Here, performance values are observed along the time axis. In addition to the performance gains, it is shown how *CRRS* results comply with the detail model results.

4.6.1 Steady State Analysis

Figure 4-10 on page 60 shows the modelled agent system. The workload scenarios are the same as described in section 4.4.2, page 59.

Validity of CRRS

Figure 4-24 shows the utilisation of a dedicated agent server "Bond". Utilisation coincides in all scenarios between detailed and *CRRS* models. This verifies the prediction that the approximation error of utilisation (see section 4.5.3) is negligible. Figure 4-25 shows the 90% confidence intervals of the round trip time (*CI RTT*) in the G/G/1 network[1], and table 4-5 illustrates the 90% paired-t confidence intervals of differences between the detailed and *CFS* models at several mean mobile agent arrival rates λ.

Differences are very small with $G/D_4/1$ and $M/H_4/1$ scenarios, although the corresponding confidence intervals do not always include zero. With G/G/1 stations, *paired-t* confidence intervals sometimes are very large. This is due to the high variance in differences and the quite small sample size resulting from the sequential simulation method which is adjusted to the round trip time, not to the batch differences. But with G/G/1 models, zero is always included in the confidence intervals, thus, models may represent the same system. Furthermore, the half widths of the large confidence intervals with $\lambda = 0.20$ and $\lambda = 0.27$ are 6.2%, resp. 9.6% of the mean value of the corresponding round trip times. This is not a very big difference.

1. Confidence intervals of round trip times of all model types can be found in Appendix A.7.

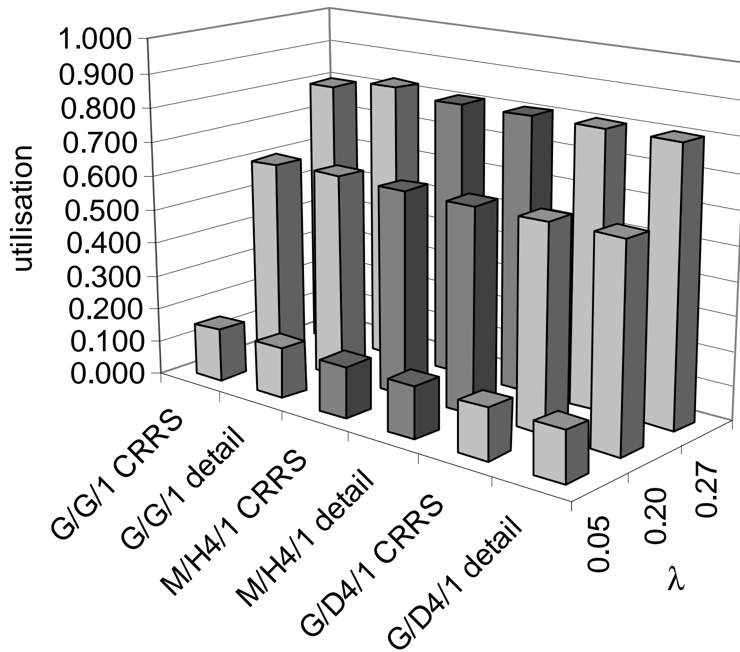

Figure 4-24: **Utilisation of detailed and CRRS models**

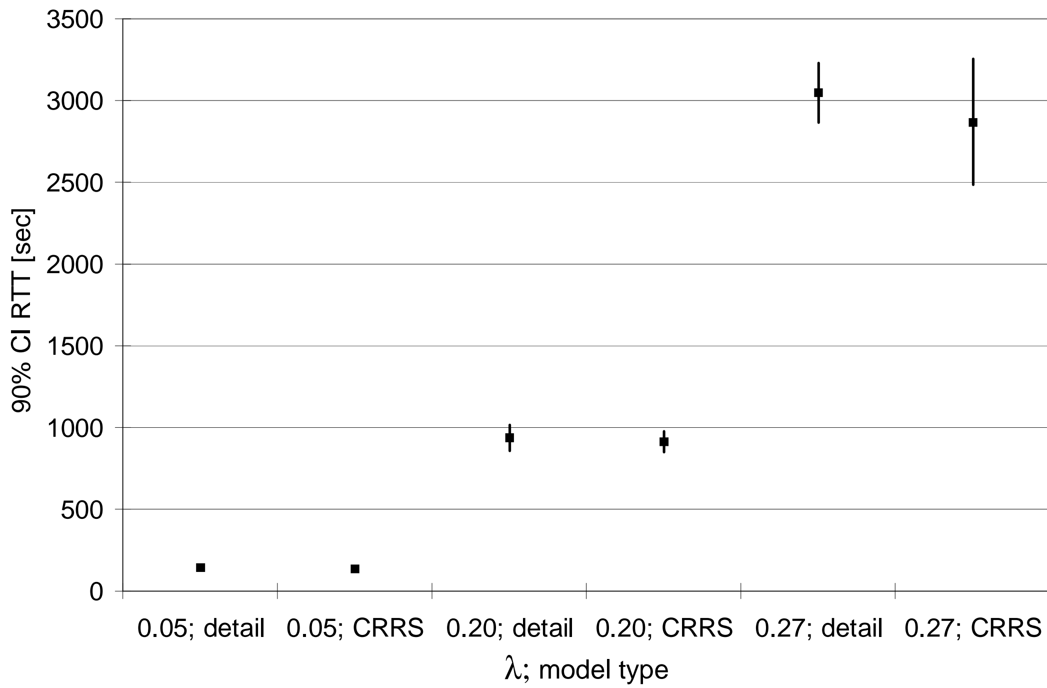

Figure 4-25: **90% confidence intervals of round trip times of detailed and CRRS G/G/1 models**

Station types	λ [ma/sec]	paired-t confidence interval (90%)
$G/D_4/1$, c.o.v.[A] = 3.0	0.05	[0.093, 0.168]
	0.20	[0.798, 1.762]
	0.27	[1.848, 4.347]
$M/H_4/1$	0.05	[-0.004, 0.229]
	0.20	[1.189, 2.962]
	0.27	[1.349, 7.189]
G/G/1, c.o.v.[A] = 3.0, c.o.v.[S_i] = 5.0	0.05	[-6.619, 1.542]
	0.20	[-86.528, 30.726]
	0.27	[-472.526, 111.475]

Table 4-5: **Paired-t confidence intervals of differences between CRRS and detailed model**

Efficiency Gains

Figure 4-26 compares the model efficiency with several mobile agent arrival rates λ. It shows absolute values of CPU time consumption. These values are converted into efficiency gains as described in table 4-6. It is evident, that the approach of *CRRS* decreases the CPU time consumption significantly with different workload scenarios. Values are averaged over 10 simulation runs. In all scenarios, the efficiency gains with *CRRS* are evident.

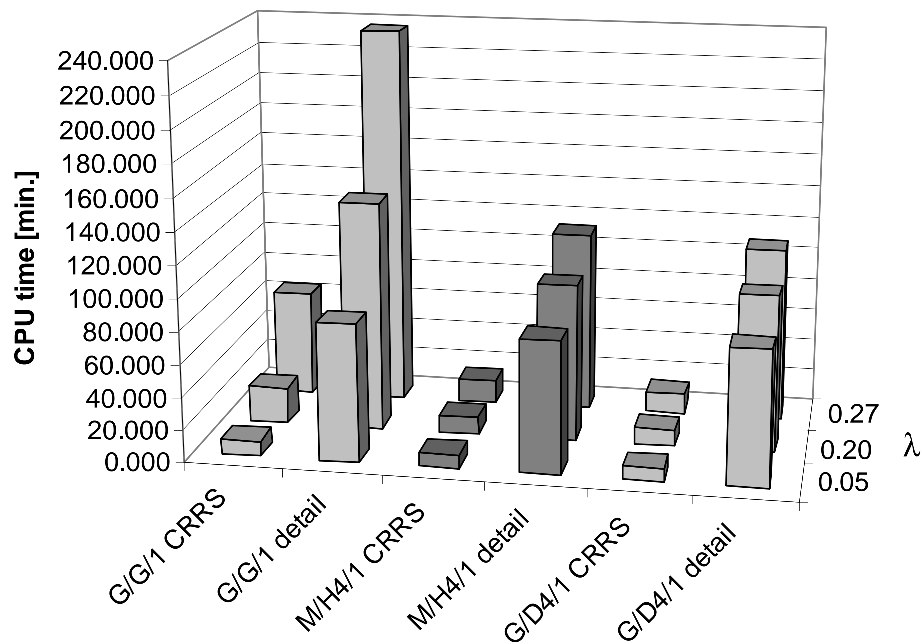

Figure 4-26: **CPU time consumption of detailed and CRRS models**

Station types on PC	λ [ma/sec]	Gains by CRRS
$G/D_4/1$, c.o.v.[A] = 3.0 on *goldeneye*	0.05	90.3%
	0.20	89.5%
	0.27	88.2%
$M/H_4/1$ on *goldeneye*	0.05	89.3%
	0.20	88.7%
	0.27	87.5%
G/G/1, c.o.v.[A] = 3.0, c.o.v.[S_i] = 5.0 on *goldeneye*	0.05	89.0%
	0.20	84.3%
	0.27	72.4%

Table 4-6: **Comparison of efficiency of CRRS and detailed model**

4.6.2 Finite Horizon Analysis

The dynamic behaviour of the mobile agent system is analysed, i.e., the changes of performance values along the time axis are observed. 80,000 seconds (approx. 28 hours) are simulated. Until simulated time of 20,000 seconds a single source generates mobile agents with a rate of $\lambda = 0.05$ agents per second. Then, 5 further sources are activated which send out agents with the same rate, each, until 60,000 seconds simulated time. Finally, 5 sources are switched off, thus, there is again a single source which generates mobile agents with a rate of 0.05 agents per second until 80,000 seconds. All sources use the same agent home server. It is assumed that there is no delay at agent home. Simulation starts with an empty system. Figure 4-14 on page 64 shows the modelled agent system.

The modelled scenarios are the same as described in section 4.4.3, page 64.

Validity of CRRS

For the $G/D_4/1$ and G/G/1 networks, 100 runs are necessary, for the $M/H_4/1$ network only 15 runs are necessary to achieve the threshold for the relative statistical error of 0.15.

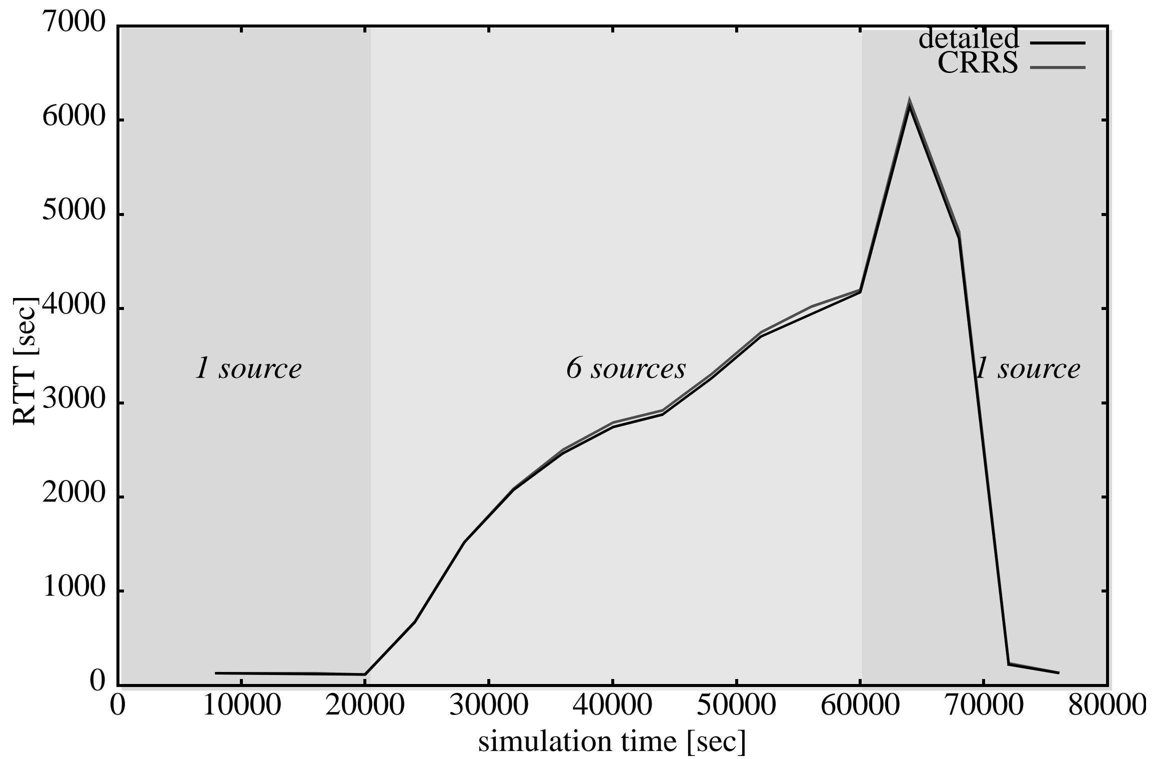

Figure 4-27: **RTTs in an G/G/1 network (averaged over 100 runs)**

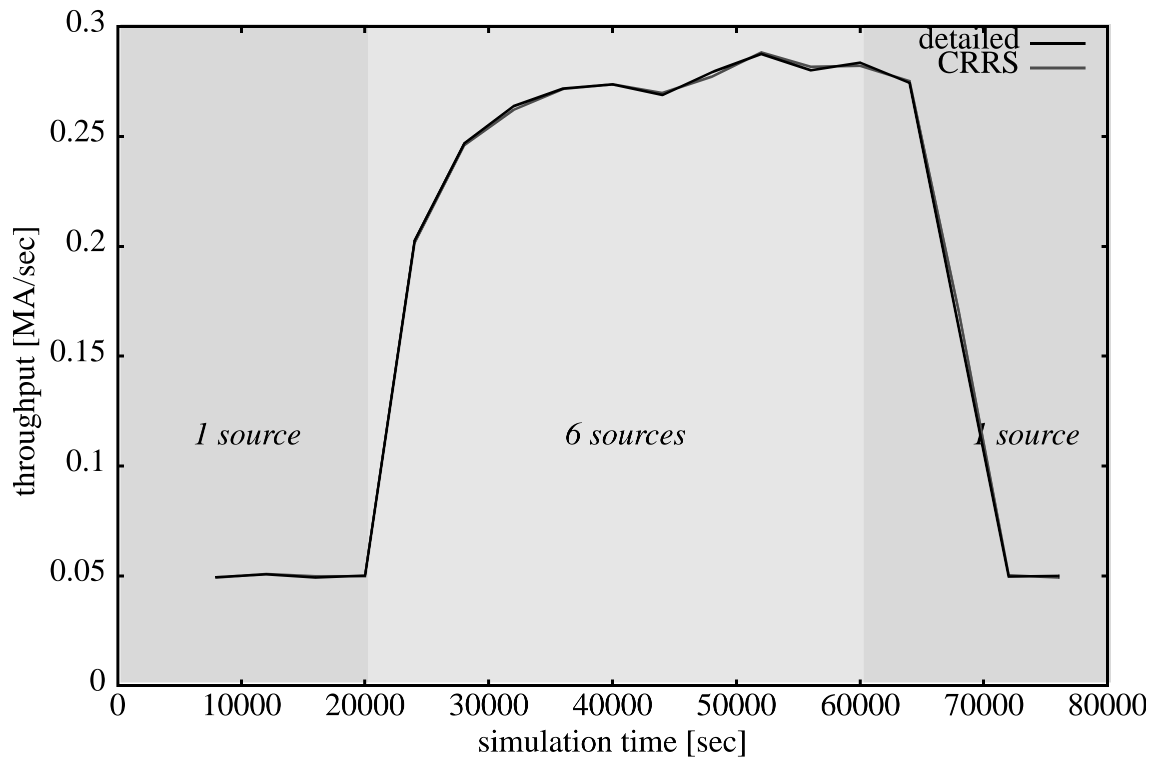

Figure 4-28: **Throughput in an G/G/1 network (averaged over 100 runs)**

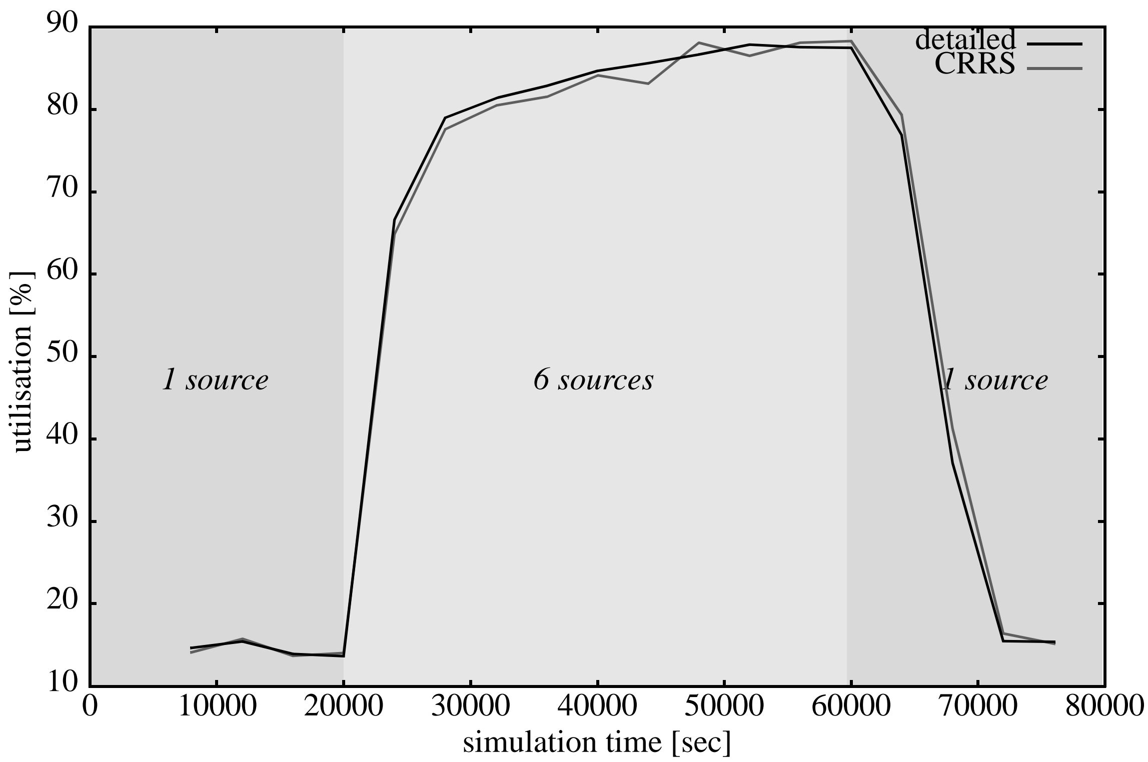

Figure 4-29: **Utilisation of server "Bond" (averaged over 100 runs)**

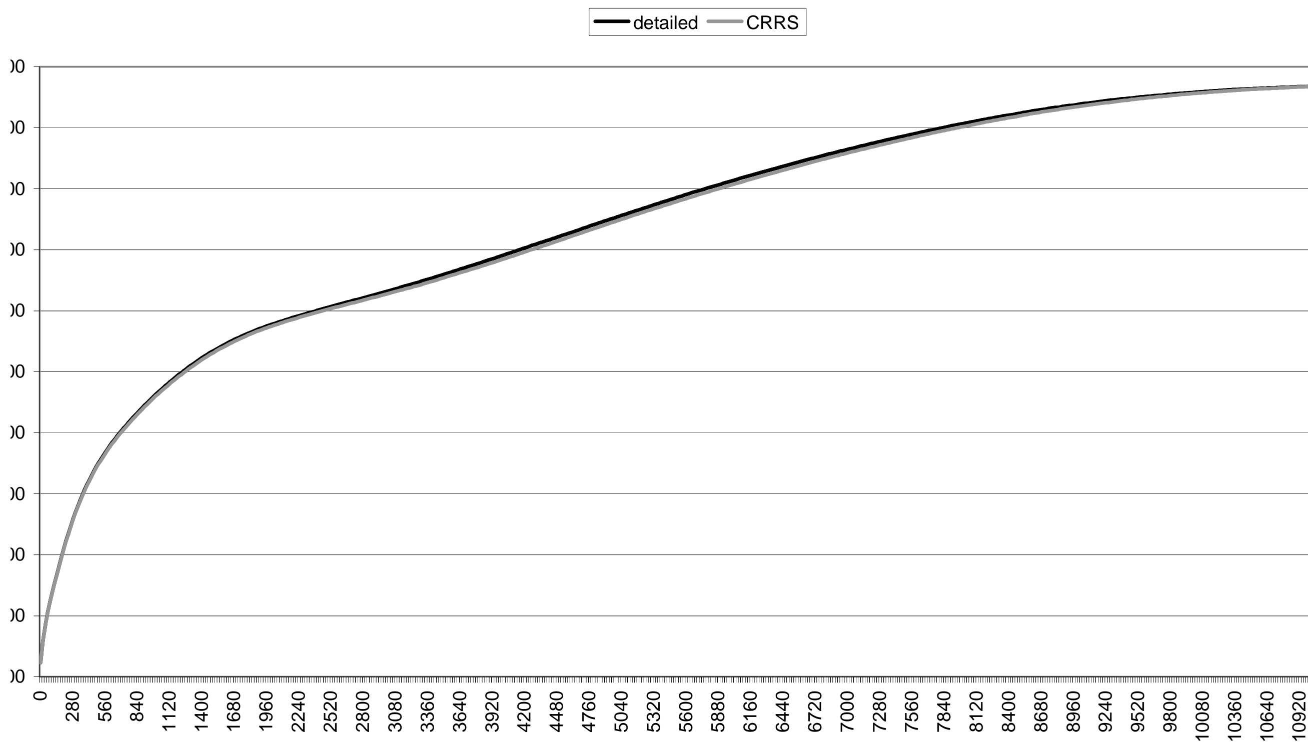

Figure 4-30: **Empirical distribution of RTT in an G/G/1 network (averaged over 100 runs)**

Figure 4-27 shows the changes of round trip times in the G/G/1 network. Figure 4-28 shows the throughput, i.e. the number of agents within 4000 seconds which arrive back home. Similar to the *CFS* experiments, at the end of the heavy workload phase round trip times and throughput increase once again significantly. This can be explained by the overloaded servers, which still need a certain time until they are empty again. Figure 4-29 illustrates the utilisation of the exemplary agent server "Bond". There are several reasons why the values in the figures of detailed and *CRRS* models are such similar:

- The same random number streams were used per replication run for detailed model and *CRRS*. Thus, interarrival times of mobile agents are the same. Service times of service agents are allocated when these agents are requested by mobile agents. Hence, if service agents are requested in a different order (due to a different order of arriving mobile agents), service times are different from one model type to another. This happens, because of partly different delays of mobile agents, caused by approximation errors (see section 4.5.3). Summarising, the method of common random numbers is not applied, but random numbers in whole are quite similar in both model types.
- Results are averaged over 100 replications and each single data value in the figures is averaged over a time window of 4000 seconds. Thus, outliers are widely eliminated.

model time [sec]	paired-t confidence interval (90%)		% differences of mean values
8000	-2.230	3.527	0.504
12000	-1.197	5.127	1.513
16000	-0.590	4.586	1.587
20000	-2.343	2.986	0.284
24000	-0.235	15.103	1.110
28000	-12.489	23.332	0.357
32000	-14.254	36.355	0.532
36000	3.795	75.949	1.618
40000	10.625	83.966	1.726
44000	4.117	78.158	1.431
48000	-3.928	89.920	1.318
52000	1.241	87.799	1.201
56000	35.096	125.845	2.044
60000	-26.000	77.063	0.612
64000	1.638	142.066	1.170
68000	-40.973	191.659	1.590
72000	-15.587	45.714	6.894
76000	0.842	7.221	3.037

Table 4-7: **Paired-t confidence intervals of differences between CRRS and detailed model (based on 100 replication runs)**

Also, the empirical distribution of round trip times (see figure 4-30) shows a behaviour similar to the detailed and *CRRS* model. Due to the approximation error of the *reduced rescheduling* technique, smaller round trip times appear more frequently with the detailed model, higher values appear more frequently with *CRRS*. To confirm the apparent validity of *CRRS*, table 4-7 shows the 90% *paired-t* confidence intervals of differences between detailed and *CRRS* model which represent the G/G/1 network. The confidence intervals do not always include zero, but nevertheless, *CRRS* is a good approximation. This can be concluded from the differences in percent between the mean round trip times of both model types. In most cases, mean values of *CRRS* are a bit higher than those of the detailed model, but they do not exceed 7% of the values of the

detailed model. It can be assumed that the approximation error is caused by the *reduced rescheduling* technique (which does not consider the allocation of the quantum in round robin servers, see section 4.5.3).

Results for finite horizon analysis of the $M/H_4/1$ and $G/D_4/1$ networks can be found in Appendix A.8 and Appendix A.9. They show in general the same match between detailed models and *CRRS* as with the G/G/1 network.

Efficiency Gains

Figure 4-31 and table 4-8 compare the duration of the execution time for a single simulation run (CPU time). Values are averaged over 100 replication runs for the $G/D_4/1$ and G/G/1 networks and averaged over 15 replication runs for the $M/H_4/1$ network. It is evident, that the *CRRS* system improves simulation efficiency significantly, i.e. up to 92.1% in the observed scenarios.

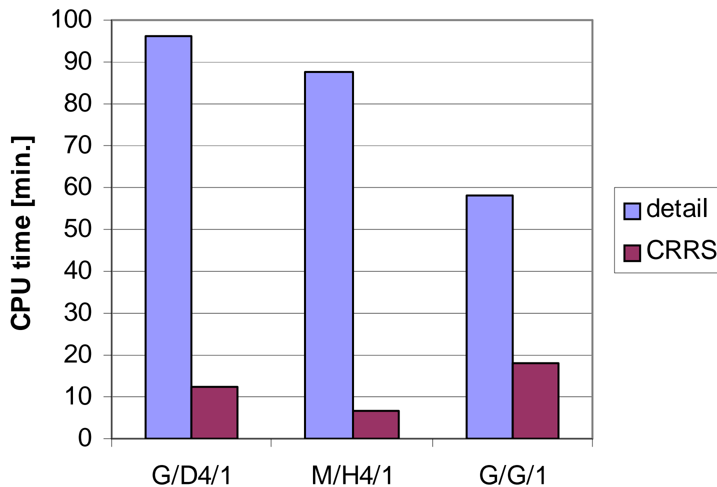

Figure 4-31: **Average amount of CPU time per simulation run**

Station types on PC	Efficiency gains of CFS (decrease of CPU time)
$G/D_4/1$, c.o.v.[A_i] = 3.0 on *blofeld*	87.2%
$M/H_4/1$ on *goldfinger*	92.1%
G/G/1, c.o.v.[A_i] = 3.0, c.o.v.[S_i] = 5.0 on *goldeneye*	69.1%

Table 4-8: **Amount of CPU time**

4.7 Summary

This chapter describes hybrid modelling techniques to increase the efficiency of simulation models of mobile agent systems by "concatenating" agent servers. Thereby, the residence time of a mobile agent at a server is accumulated with its network delay which arises when migrating to the next server. Instead of delaying the agent first for the residence time at a server and afterwards for the network delay, there is only a single cumulative delay. When a mobile agent arrives at a server, it is delayed for this value. The next event in simulation is the arrival at the next agent server. Thus, the end-of-service event at the server is omitted. This reduces the number of events which are simulated and, thus, increases simulation efficiency.

To accumulate residence time and network delay which is calculated analytically, it is necessary to calculate residence time analytically, too. The kind of this calculation depends on the type of agent server. The approaches *CFS* and *CRRS* allow for the calculation of agents' residence time on file resp. compute servers. Simulation efficiency is increased further by calculating residence time instead of simulating scheduling processes in detail.

The techniques and algorithms of the new approaches are described, efficiency gains and approximation errors are discussed. Finally, validity and efficiency gains are demonstrated by an empirical evaluation of the approaches.

The basic idea of concatenating servers is applicable to all networks where network delay can be calculated analytically. Furthermore, the algorithm used with *CFS* to directly calculate residence time of incoming jobs is applicable with arbitrary FIFO servers. The wide range of applicability of the *reduced rescheduling* technique (which is the basis of *CRRS*) has been demonstrated in [17].

The advantage of *CFS* and *CRRS* compared to other algorithms which increase efficiency is, that it is not necessary that the modeller has knowledge about the stochastic characteristics of the modelled systems. Furthermore, general stochastic patterns can be modelled. Beyond this, multiple output analysis techniques are possible. Analyses of performance results may include transient and steady state analysis. Point estimators as well as interval estimators are provided. Beyond mean values, the second central moment and histograms of performance values can be output, which allows for the investigation of service level agreements. In case of transient analysis the development of performance values along the time axis is observable.

5 Capacity Planning of Mobile Agent Systems

Previous chapters describe the modelling method, implemented with the simulation environment *JaDEMAS*. Furthermore, they describe approaches for efficient simulation. So far, it is assumed that proper model parameters exist. Validity of model results, compared to real agent systems, has not been addressed. But, these issues are part of the capacity planning process. Hence, they are dealt with in this chapter.

This chapter describes the phases of the capacity planning process. In addition to performance modelling, it shows how measurements have to be planned and executed, so that results directly can be transferred to simulation models. Measurement results are used as input parameters or they are compared to model results to evaluate models' validity. The methodology of capacity planning is based on [6], pp. I-31 - I-38, and is transferred to the context of mobile agent systems where *JaDEMAS* is used for performance modelling.

Figure 5-1 illustrates the process of capacity planning. Based on a real system implementation a trustworthy baseline model is built, which is the foundation for the prediction model. The prediction model represents the future system of interest, which performance shall be analysed. If the model forecasts that the performance of the planned system is not satisfying, the model should be modified until the performance requirements are met. Then, the real system can be implemented according to the configuration of the prediction model.

The following sections describe the single phases of the performance capacity planning process. It starts with the building of a baseline model, which has to be validated and calibrated. The process ends with the analysis, resp. the "tuning" of the prediction model which represents the planned real system.

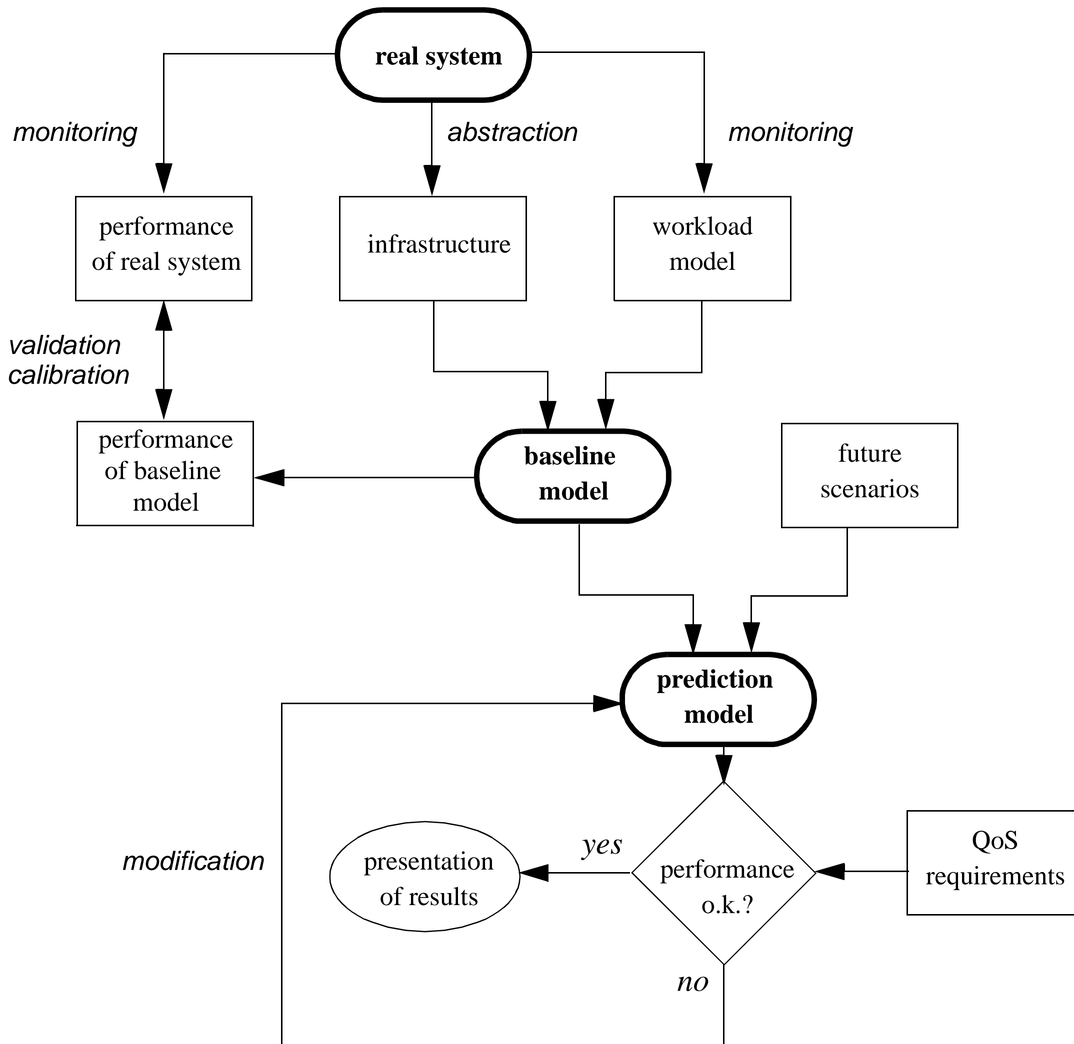

Figure 5-1: **Process of capacity planning ([6], p. I-32, modified)**

5.1 Baseline Modelling

Models of real systems are either built to reduce complexity of existing systems to easier understand processes or they are built to analyse and tune planned systems which are not yet implemented. This dissertation concentrates on the latter case. Instead of costly implementing the real system and identifying errors or performance bottlenecks afterwards, a more competitive model is analysed.

When developing models, it is always necessary to have a reference implementation of a real system to adjust the model to. This does not mean, that the whole planned system has to be implemented. Rather, a smaller, similar system can be used to build a model of. If this model is validated it can be modified according to the planned system later on. The model which represents the reference implementation is called *baseline model*. It is assumed that, if the baseline model maps the real system correctly, then the extended prediction model will map the planned system correctly.

5.1.1 Specification of the Infrastructure in the Model

The infrastructure of the model is given by the infrastructure of the real system. According to *JaDEMAS*, agent servers and network links have to be specified concerning:

- Agent server's DNS names and resources as there are
  - type (file or compute server),
  - number of CPUs and time slice in case of compute servers.

  Calibration parameters:
  - service rates for serving user load, system and migration overhead (should be set to 1.0 before calibration and can be changed to calibrate the model),
  - mean ghost migration delay and mean duration (in number of agents) of on and off phases of "ghost server" (should be set to 0 before calibration and can be changed to calibrate the model),
  - fixed system overhead which arises per server when starting or migrating an agent (should be set to 0 before calibration and can be changed to calibrate the model),
  - Service amount for DNS look up and service rates of DNS server.
- Routing table which specifies the characteristics of links between servers. Simple model: bandwidth. Extended model: bandwidth, portion of bandwidth which cannot be used by application which is due to background load or overhead, TCP round trip time per link, service rates of nodes on each link.
- With import of external generator and/or initialisation classes: pathes to this classes and necessary parameters.

If a server is of type compute server and shall be modelled in detail, the time slice of its CPU has to be specified. If this cannot be taken from the operating system parameters, the time slice has to be estimated. *JaDEMAS* proposes 100 milliseconds as default. If the compute server is modelled by *CRRS*, the time slice does not have to be specified.

Calibration parameters initially should by set to values 0 or 1 and can be modified to calibrate the model.

TCP round trip time, which is used in the extended network model, can be estimated by measuring the round trip time when executing the *ping* command. Parameters "portion of bandwidth which cannot be used by application which is due to background load or overhead" and "service rates of nodes on each link" should be used as calibration parameters. This means, they should initially be set to 0, resp. 1 and should be adjusted afterwards.

5.1.2 Workload Specification

After modelling the infrastructure of the mobile agent system, the workload which utilises the system has to be modelled. According to *JaDEMAS*, the following workload parameters have to be itemised:

- Arrival rates, and burstiness and variation of requests,
- agent classes (path) and their first names in the agent system,
- home server(s) per agent,
- data volume of mobile agents,

- service times of agents at server resources.

Arrival rates, service times and data volume of agents have to be measured or estimated. A good estimation for the data volume of mobile agents is the size of their class file. Arrival rates and service times are not easy to obtain. They usually have to be measured in a real system, currently in use, or in a testbed. The next two sections deal with the obtaining of service times and arrival rates.

Measurement of Service Times

Figure 5-2 shows the workload types as defined in *JaDEMAS* and it describes possibilities to measure the time consumption dued to these workload types (as well see section 2.4.1).

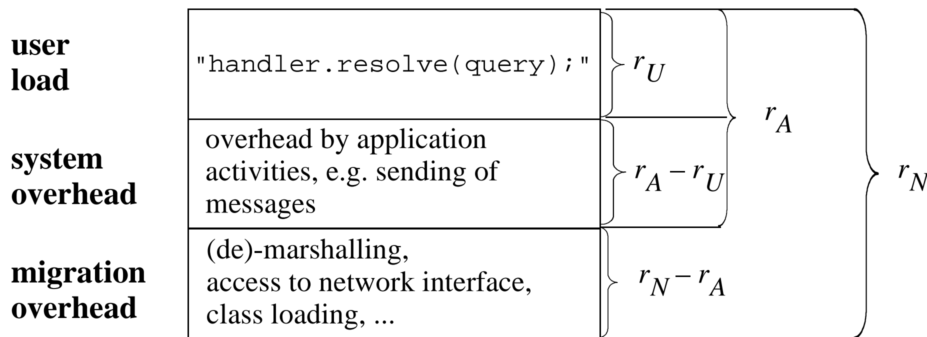

Figure 5-2: **Measurement of time consumption**

r_A describes the time which elapses from the beginning until the end of the execution of a mobile agent's program code at a server. This duration can be measured by instrumenting the agent code, i.e. by implementing the measurement of timestamps at the beginning and at the end of the code. *User load* can be measured in the program code of a service agent by setting one measuring point before and one after the code which provides the main service, e.g. a function called `resolve(...)`. As long as the service agent allocates server resources, the requesting mobile agent is passively waiting. *System overhead* describes the time consumption of the mobile agent from the applicational point of view which is beyond waiting for the service agent. *Migration overhead* can be calculated by measuring the mobile agent's residence time from the network interface (r_N) and substracting r_A.

Generally, these measures describe residence times at single server resources. But, if contention at the servers is avoided, i.e. if these values are measured if only a single agent resides at a server, residence times are equal to service times, because no waiting times arise. This means, that service times can be obtained by measuring the residence times in an "empty" system. This is easy to achieve by using a laboratory environment, i.e. a testbed. If a system in a production environment is used, measurements should take place when no usual business proceeds. E.g. measurements could be done after hours when a single user creates single requests, i.e. mobile agents.

JaDEMAS supports trace driven simulation [36], p. 296, for service times. Measured service times can be stored in a trace file for each workload type at each server. Files are read from the beginning to the end. If the end of a file is reached, reading of service times again starts from the beginning of the file. Alternatively, the modeller can specify a stochastic distribution for the service times.

Arrival Rates

Arrival rates of mobile agents determine the workload intensity. By specifying arrival rates, different scenarios are defined where real system and baseline model shall be compared for validation purposes.

Usually, those workload intensities which are of interest for the planned system should be analysed to assure the confidence in the baseline model. Arrival rates of requests, i.e. of mobile agents, which are necessary to produce the desired level of intensity can be estimated by estimating the utilisation of server resources: $\rho = \frac{\lambda}{n} \cdot \bar{s}$, where n = number of processors, λ = arrival rate, $\bar{s}$ = mean service time. Referring to a single agent server, a feasible range for low traffic intensity would be $0 < \rho \leq 0.3$, for middle traffic intensity $0.3 < \rho \leq 0.7$, and for high traffic intensity $0.7 < \rho < 1.0$.

The desired arrival rates can either be achieved in a running system by systematic measurements in known phases with low, middle and high user activity, or arrival rates can be produced by a workload generator. The former case has the advantage that measurement can take place while the system is used as usual without additional effort, and measured values are quite realistic. The latter case can be preferred because arrival rates can be adjusted more precisely. Beyond the arrival rate, the modeller has to specify the burstiness of the expected workload. A realistic level of burstiness should be estimated based on measurements in a real system.

To effectively compare measurement results of the real system with simulation results of the baseline model, it is advisable to set interarrival times of requests in the real system as similar as possible to interarrival times in the baseline model.

5.1.3 Modification of Agent Program Code

After the specification of the infrastructure and the workload model, the agent program code has to be prepared to run in *JaDEMAS*. For a detailed description of the parts of the code which usually have to be modified see section 2.4.7.

5.1.4 Execution of Experiments and Preparation of Results

JaDEMAS models provide all result values, which are necessary to validate and calibrate models: Residence times r_A and r_N of agents at single servers, network delay, round trip times, utilisation and throughput. These values are provided as mean values, partly as histograms or even as single values per agent.

In the real system, results have to be measured. The measures r_A and r_N have been introduced in section 5.1.2. Figure 5-3 illustrates how to obtain these measures from an alternative point of view. It shows the process of mobile agent execution and migration in *Tracy* as known from section 2.3.1, page 12. Furthermore, it describes the measurement of residence times and additionally of round trip times. The term rtt_N names the round trip time of mobile agents which can be observed from the network interface. It describes the duration from the occurrence of the first packet of the mobile agent in the network (sent from the home server) until the arrival of the last

packet of the mobile agent back at the network interface of the home server. rtt_A defines the round trip time which can be measured by instrumentation of the mobile agent code, i.e. the duration from the execution of the first statement at the home server until the last statement at the home server (after having returned home again).

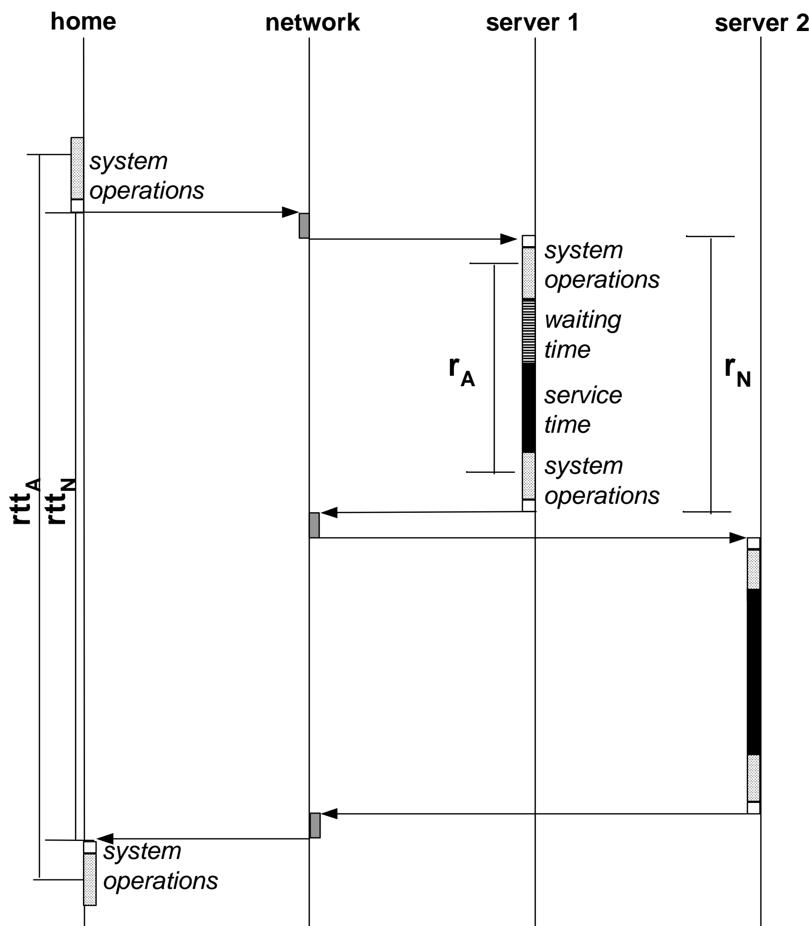

Figure 5-3: **Measurement of performance values**

Unfortunately, utilisation of resources per agent server cannot be measured with sufficient accuracy. Usually, monitoring tools provide measurement of utilisation of CPUs and perhaps of I/O devices. But, a correlation to the workload types which cause the utilisation is hardly possible. Besides, measurement of utilisation had to be synchronised with measurement of residence and round trip times to compare results. Because of these difficulties, it is advisable to validate and calibrate the baseline model on the basis of round trip time, residence times and throughput. To get an idea of resource utilisation in the real system, it can be calculated according to the utilisation law [29], p. 556 et seq.: $U = X \cdot S$, where U = utilisation, X = throughput, S = mean service time.

To evaluate the baseline model, differences to the real system have to be analysed. A *paired-t confidence interval* of the performance results should be calculated ([36] page 287 et seqq., resp. page 557 et seqq.) to compare baseline model and real system. This approach was introduced previously in chapter 4.

Baseline model and real system should be executed several times with independent workload streams (concerning interarrival times and service times). To improve comparability of results, the *common random number* approach ([36], p. 582 et seq.) can be applied: The same arrival streams are used for one execution of the baseline model and the real system. This demands the usage of a workload generator in the real system or it demands a trace driven simulation where interarrival times and service times are obtained from the real system. Differences between results of the baseline model and real system are calculated. A confidence interval is then calculated for these differences. The baseline model and the real system are executed as frequently, resp. as long as necessary to get below a prespecified width of the confidence interval. Usually, the threshold for the width is specified as relative statistical error (ratio of the half-width of the confidence interval and the point estimate), see [47]. Reasonable values for the confidence level and the relative statistical error are 0.9 and 0.15.

The method for the output analysis (steady state or finite horizon analysis) should be chosen according to the scenarios which are planned for the prediction model. For an explanation and an example of steady state and finite horizon analyses see chapter 4.

5.1.5 Model Validation and Calibration

After the execution of measurements and model runs, the next step is the analysis of the observed differences between baseline model and real system.

Law and Kelton state: "Clearly, the decision as to whether the difference between a model and a system is practically significant is a subjective one, depending on such factors as the purpose of the model and the utility function of the person who is going to use the model." [36], p. 288. Nevertheless, there exist certain rules of thumb proposing tolerable differences. Lazowska et al. set the following ranges for tolerable differences between mean result values of real system and queuing model results:

system throughput	system response time	device utilisation
5 to 10%	10 to 30%	5 to 10%

Table 5-1: **Reasonable tolerances in validation (from [37], table 12.3, p. 292)**

Although these values refer to mathematically solved queuing networks (with multiple job classes), they shall be assumed to be reasonable for simulation, too. Hence, mean result values can be compared and be evaluated according to table 5-1. Alternatively, borders of the *paired-t* confidence intervals can be analysed to get best or worst case scenarios. If differences lie within the tolerance ranges the baseline model is regarded to be valid and the performance prediction can start. Otherwise, the baseline model has to be calibrated, i.e. it has to be modified until differences are sufficiently small. This should be done using the calibration parameters, described in section 5.1.1. Probably, modelled residence times are too small compared to the real system because overhead usually is not regarded in the first model. Then, service rates of system resources have to be decreased with increasing number of agents at a server.

5.2 Performance Prediction

After gaining a reliable baseline model, it has to be extended according to the planned system which performance shall be analysed. Relevant system scenarios, which include workload and

capacity of servers and networks have to be specified. Furthermore, performance requirements, resp. thresholds of performance values have to be set to evaluate model results.

5.2.1 Specification of Future Scenarios

In this phase the design of the planned system has to be specified: infrastructure and workload scenarios have to be defined.

The infrastructure is usually an extension of the initial real system, e.g. agent systems with more agent servers with different capacities are analysed. Workload parameters depend on the scenarios which are planned to be analysed. The variation of requests during the simulated time, the stochastic characteristics of the arrivals of these requests have to be estimated. If systems similar to the planned system exist, such parameters can probably be measured there. Otherwise, parameters have to be estimated according to the modeller's experience.

Furthermore, techniques for the analysis of model results depend on the specified scenarios. If long-term characteristics of the agent system are of interest, results should be analysed by steady state analysis. Otherwise, if evolution of performance values shall be analysed during a certain time duration, a finite horizon analysis should be executed.

Finally, the baseline model has to be extended to the prediction model according to the scenarios of interest.

5.2.2 Quality of Service Requirements

To evaluate results of the prediction model, quality of service requirements for the system performance have to be specified. Depending on the demands on performance, these requirements can be expressed as thresholds in terms of mean values or as quantiles. Examples are "the mean round trip time of mobile agents should not exceed 20 seconds" or "80% of round trip times should be below 25 seconds". *JaDEMAS* supports the examination of both types of requirements.

Furthermore, the modeller has to check if the requirements are specified completely, realistically and consistently. Priorities of requirements could be defined, resp. thresholds should be weighted as weak or strong. Additionally, Jain states that performance requirements have to be "SMART": specific, measurable, acceptable, realisable and thorough. "Specificity precludes the use of words like 'low probability' and 'rare'. Measurability requires verification that a given system meets the requirements. Acceptability and realizability demand new configuration limits or architectural decisions so that the requirements are high enough to be achievable. Thoroughness includes all possible outcomes and failure modes." [29], p. 42.

5.2.3 Execution of Experiments

Depending on the specified future scenarios a steady state or finite horizon analysis has to be executed. In case of steady state analysis, a single long simulation run is sufficient. *JaDEMAS* automatically controls the duration of the simulation according to the specified level of confidence and the relative statistical error of the round trip time. In case of finite horizon analysis, multiple independent simulation runs have to be executed until the threshold for the relative statistical error at the desired confidence level is reached.

Reasonable values for the confidence level and the relative statistical error are 0.9 and 0.15. For a more detailed explanation and for examples of steady state and finite horizon analysis see chapter 4.

5.2.4 Analysis of Results

Performance results of the prediction model have to be evaluated according to the specified quality of service requirements. If results do not meet the requirements, the model should be modified (which would correspond to the tuning of a real system) until performance is satisfying. Finally, the real agent system can be implemented according to the prediction model.

5.3 Mobile Agent Laboratory MOLAB

As described in the previous sections, models have to be parameterised with realistic input values to provide realistic results. Furthermore, to verify model results these have to be compared to measurements of real agent systems. Thus, real agent systems have to be measured for two purposes:

1. gaining realistic model input parameters,
2. validation of model results.

As already mentioned, it is useful to have a testbed, resp. a laboratory to measure the real system. In this laboratory it should be controllable exclusively which applications load the system to identify cause and effect correlations. The mobile agent laboratory *MOLAB is* an example for such a laboratory.

5.3.1 Infrastructure

Figure 5-4 shows the infrastructure of the mobile agent laboratory *MOLAB* at the University of Essen. All computers are single processor machines, and they are located in a switched Fast-Ethernet segment with a bandwidth of 100 MBit/s. Furthermore, the computers are described by the following parameters:

Computer	Processor Type	Operating System	No of Processors	Memory
goldeneye	Intel Pentium IV, 2.6 GHz	SUSE Linux 9.0	1 (Hyper-Threading Technology)	1 GByte
bond	Intel Pentium III, 1.2 GHz	Windows XP	1	512 MByte
goldfinger	AMD Athlon, 1.2 GHz	SUSE Linux 8.0	1	512 MByte
bender	Intel Pentium III, 500 MHz	Solaris 8	1	256 MByte

TABLE 1: Computers in MOLAB

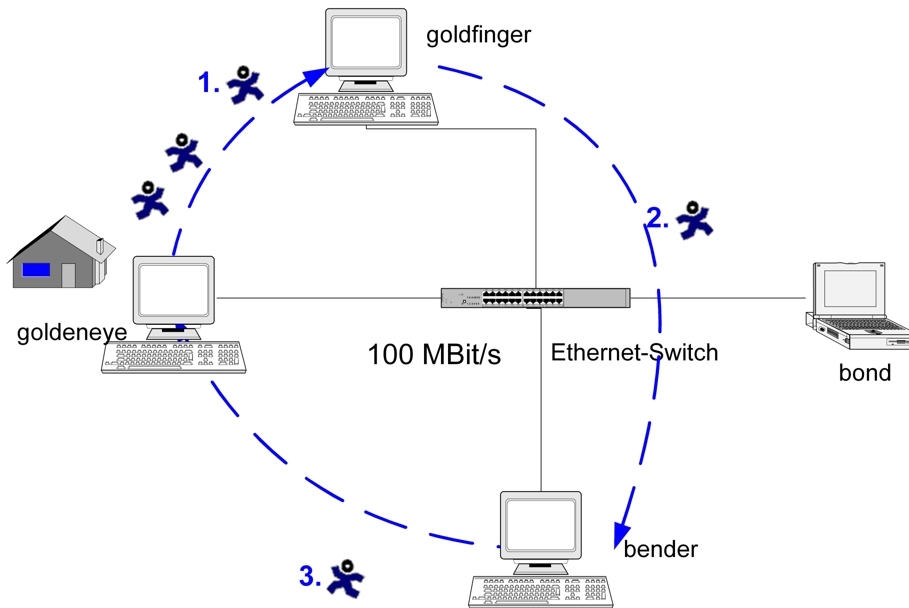

Figure 5-4: **Mobile agent laboratory MOLAB with an exemplary agent route**

goldeneye is the mobile agent's home server, i.e. the network access point for users of the mobile agent system. All computers of *MOLAB* are running the mobile agent platform *Tracy* [4], [10]. By default, mobile agents migrate by *Push-All-To-Next* strategy. This means that the complete agent as well as its data is sent to the next destination server at once. Agents migrate via the SATP/TCP strategy: *Tracy's* specific *Simple Agent Transfer Protocol* is used on top of the TCP/IP network protocol to transmit mobile agents.

5.3.2 Workload Generation

MOLAB contains a workload generator which generates *Tracy* mobile agents at the network access point, i.e. at the mobile agents' home server. Several distributions can be chosen for the interarrival times. Thus, smoother or burstier traffic can be generated.

5.3.3 Monitoring

In *MOLAB,* the mobile agent monitoring tool *GrepAg* [22] is installed. *GrepAg* is a Java2 based network analysing tool to monitor mobile agents. *GrepAg* runs on agent servers. It measures round trip times, residence times and throughput of mobile agents from the network perspective. *GrepAg* monitors these measures as single values and calculates averages, variances and histograms.

5.4 Summary

In this chapter a method for capacity planning is described. The methodology refers to mobile agent systems, but is transferable to general IT systems. The phases of the capacity planning process are described and advises are given to help system developers during the capacity planning process. The testbed *MOLAB* is described. *MOLAB* can be regarded as a reference installation for a measurement environment for mobile agent systems. It can easy be replicated at any other location with a similar infrastructure.

6 Case Study

This chapter describes a case study for the capacity planning of an agent system which implements an information retrieval application. Modelling techniques which are explained in previous chapters, are applied within the process of capacity planning. Thus, this chapter combines the developed modelling approaches and the methodology of capacity planning. Furthermore, it shows that the developed methods are applicable with manageable effort.

The case study is designed as realistic as possible. It is assumed that there is only a low budget for this business, hence, the effort shall be minimised. The following sections show the capacity planning process step by step according to the methodology developed in chapter 5.

6.1 Collegiate Timetable Service

The case study investigates an application for a timetable service for students at a university. This application will be implemented by a mobile agent system and it is called *Collegiate Timetable Service*. With this application, students can request automatically generated timetables according to the courses they wish to attend. Figure 6-1 describes the architecture of the application. At the system entry point there is a local agent server with a stationary system entry agent. This one operates with the help of transaction and message handlers. Students (external users) specify their desired courses via a web interface. The web server delivers a request to the local agent server. The transaction handler receives the request and forwards it to *SystemEntryAgent*. This one analyses the request and generates a mobile agent (*QueryAgent*) with the order to visit several faculty servers and to collect information about dates and locations of the specified courses. At each faculty server (remote agent server), the mobile agent asks a system agent (*ServiceAgent*) to give information according to the student's request. *ServiceAgent* calls a service handler which accesses the faculty's data base (XML files) and delivers available information to *ServiceAgent* which forwards it to the mobile agent. This way, the mobile agent travels from server to server until it has all information needed to build a timetable. Finally, it returns home to the local agent server and submits the collected information to *SystemEntryAgent*. This one analyses the information, solves time conflicts if necessary, and advises the message handler to gen-

erate and send an email to the student with the resulting timetable. For further details about the application see [58].

Collegiate Timetable Service shall be implemented with 10 mobile agent servers (faculty servers). They will be located in the university's Intranet and they will be fully mashed with a bandwidth of 100 MBit/sec. Capacity planning starts at a point where agent program code is implemented and only tested functionally with the help of two test computers.

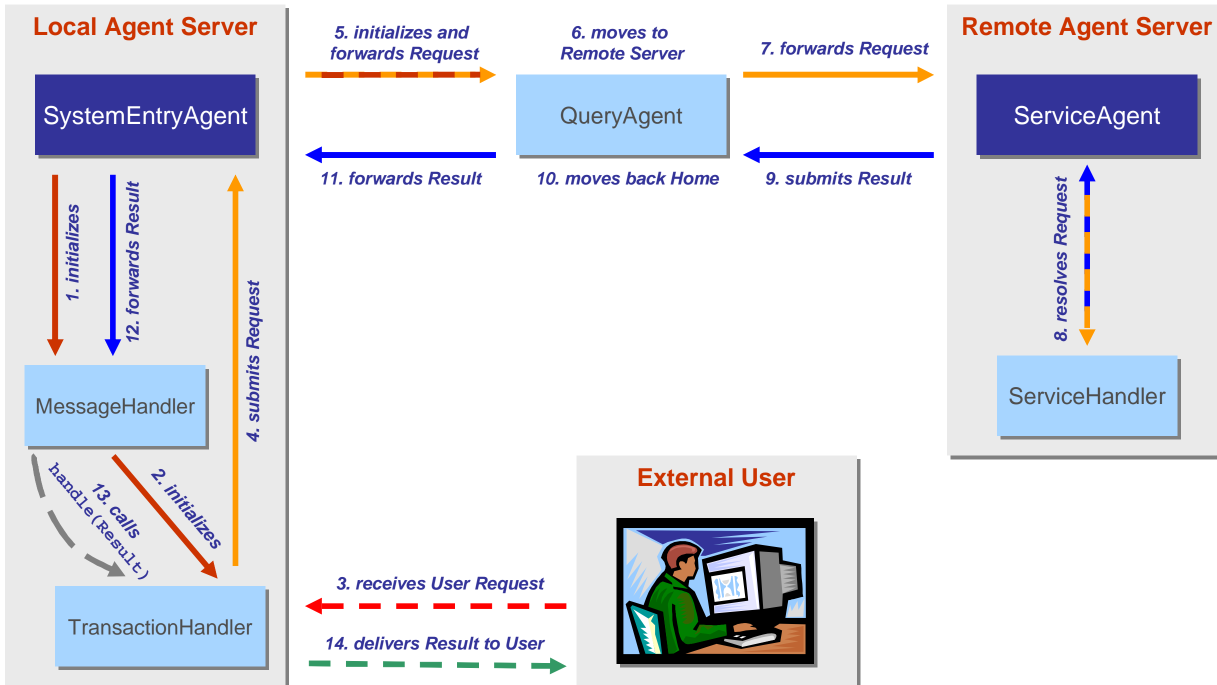

Figure 6-1: **Architecture of Collegiate Timetable Service, according to [58]**

Collegiate Timetable Service is a complex mobile agent system since it has an interface to a web service and does not only contain agents as software entities, but also includes handlers to perform services. Even this complex system can be modelled with *JaDEMAS* quite easy.

6.2 Baseline Modelling

There exists no comparable running system at the university so far, thus, measurements for baseline modelling take place in a testbed. *MOLAB* (see section 5.3 on page 93) is used to implement the real system for this purpose. Servers *goldfinger*, *bond* and *bender* are implemented as remote agent servers with *ServiceAgent* and *ServiceHandler* each and with single faculty data bases. *goldeneye* is the local agent server with *SystemEntryAgent*, *MessageHandler* and *TransactionHandler*. Servers in *MOLAB* are fully mashed by a switched Fast-Ethernet, which corresponds to the planned network structure for *Collegiate Timetable*. Servers' capacities accord to four of the future servers, i.e. the future system is an extension of the implementation in *MOLAB*.

6.2.1 Specification of the Infrastructure in the Model

The main service which is provided at the agent servers is information retrieval, thus, servers are modelled as file servers. Aggregates are used (*CFS*, see section 4.3, page 52) to increase efficiency.

Furthermore, the network links have to be specified by a routing table for an extended network model. It contains network parameters which are used to model TCP pipes between servers. The routing table is a matrix in a simple text file. Figure 6-2 shows a snapshot of the routing table used in the baseline model. Each value in the matrix consist of the four components bandwidth, relative decrease of bandwidth (e.g. by background traffic), the average TCP round trip time for a link between two servers and service rates of nodes at a link. According to *MOLAB*, the bandwidth is set to 100 Mbit/sec for each link. Links are used by the application exclusively and the average TCP round trip time is 0.1 ms (measured as round trip time of *ping*-packets). The service rate for the additional delaying of mobile agents on their trip through a network link is set to 500,000 agents per second at each link. This is a first estimation and can be modified afterwards for means of calibration.

```
0;0;0;0                   100;0;0.0001;500000     100;0;0.0001;500000
100;0;0.0001;500000       0;0;0;0                 100;0;0.0001;500000
100;0;0.0001;500000       100;0;0.0001;500000     0;0;0;0
100;0;0.0001;500000       100;0;0.0001;500000     100;0;0.0001;500000
100;0;0.0001;500000       100;0;0.0001;500000     100;0;0.0001;500000
100;0;0.0001;500000       100;0;0.0001;500000     100;0;0.0001;500000
100;0;0.0001;500000       100;0;0.0001;500000     100;0;0.0001;500000
100;0;0.0001;500000       100;0;0.0001;500000     100;0;0.0001;500000
100;0;0.0001;500000       100;0;0.0001;500000     100;0;0.0001;500000
100;0;0.0001;500000       100;0;0.0001;500000     100;0;0.0001;500000
```

Figure 6-2: **Snapshot of routing table**

The average TCP window size has to be specified via the graphical user interface. Figure 6-3 shows the GUI for the specification of the infrastructure.

The real agent system (mobile agents, system agents and handlers) is initialised by the class `Main` of *Collegiate Timetable Service*. This class is used to initialise the system in simulation, too (with minor modifications). It calls the method `startAgent` which causes *JaDEMAS* to load the corresponding agent byte code.

Initially, calibration parameters are set to default values.

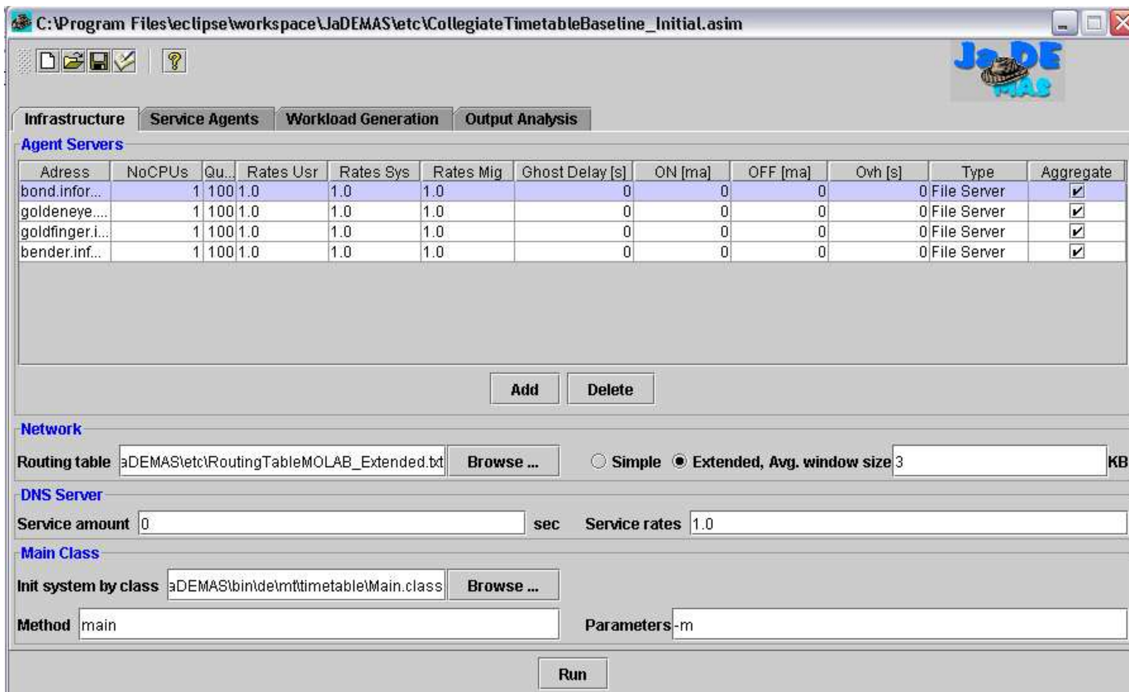

Figure 6-3: **Specification of infrastructure**

6.2.2 Workload Specification

In the simulation model the workload is generated by an individual-implemented generator class which is as well used (with slight modifications) to generate workload in the real system. The mean data volume of the mobile agents is set to 6 KByte according to the size of their class file.

Measurement of Service Times

The program code of mobile and service agents has been instrumented to measure time consumption which is due to user load, system and migration overhead[1]. Timestamps are measured at start and at end points of corresponding operations and differences are calculated. Results are written to trace files (one for each workload type at each agent server).

For the measurement of service times, the workload generator in *MOLAB* generates requests with a constant interarrival time of 2 seconds. Agent's round trip times are less than 1 second with low workload, thus, it can be guaranteed that no contention arises at the agent servers, i.e. that measured time consumption is equal to agents' service time. 1000 mobile agents are generated, hence, the trace files contain entries of 1000 different agents.

1. For the specification of the different workload types, see section 2.4.1, page 18.

Arrival Rates

Next, scenarios have to be specified to validate the baseline model. Thus, the arrival rates have to be determined which produce the desired workload intensities. Scenarios with low and high workload intensity are of major interest. To determine the corresponding arrival rates, resp. inter-arrival times, the expected utilisation of server resources is calculated: $\rho = \frac{\lambda}{n} \cdot \bar{s}$, where n = number of processors, λ = arrival rate, $\bar{s}$ = mean service time. Each resource (system, migration and user component) are modelled with a single processor, i.e. $n = 1$. $\bar{s}$ is calculated as average of measured service times at the user, system or migration component.

Hence, two workload scenarios are defined:

- Low workload intensity with interarrival time of 0.2 seconds, which corresponds to a calculated utilisation of the expected slowest resource (migration component of server *bender*) of 20%,
- high workload intensity with interarrival time of 0.05 seconds, which corresponds to a calculated utilisation of the migration component of *bender* of 80%.

6.2.3 Modification of Agent Program Code

To transfer the real agents into *JaDEMAS*, the program code had to be modified in the following way:

- Imports of *Tracy* packages have to be changed to the corresponding *JaDEMAS* packages.
- Methods `startAgent` and `addAgent` get the additional parameter <DNS name> of the agent's home server.
- In the initial class `Main`: The method `startAgent` has to be called 3 times to start each service agent at every remote agent server and once to start the system entry agent at the local agent server.
- In the real agent system the transaction handler listens to a socket for incoming requests. This would block the simulation. Thus, the *TransactionHandler* has to be modified to receive incoming requests from a *JavaDEMOS* internal buffer.
- *ServiceAgent* consumes time to fulfil its service. Thus, the method call `consume()` is added in *ServiceAgent*. Service amounts are read from trace files.
- The faculty data bases (XLM files) are copied to the computer where the simulation is going to run.

6.2.4 Execution of Experiments and Preparation of Results

As already mentioned, the capacity planning process shall be executed with an effort as small as possible. There is no capacity to run multiple measurements, hence, the c.o.v. of interarrival time is set to 0. This means, a single measurement is executed with constant interarrival times of 0.2 seconds and another with constant interarrival times of 0.05 seconds. Due to this parameterisation, the corresponding performance models contains deterministic arrival processes and service times read from trace files. Hence, there is a single model run necessary per workload intensity. Different model runs would all provide the same results because no random number generators

with varying seeds are used. Thus, the comparison of the two systems (model and real system) by analysing a paired-t confidence interval is not applicable. Rather, average, maximum values and quantiles of performance results of measurements and simulation runs shall be compared to validate the baseline model.

JaDEMAS models provide all result values, which are necessary for model validation and calibration: Residence times r_A and r_N of agents at single servers[1], network delay, round trip times (rtt_A and rtt_N), utilisation and throughput.

In the real system, in *MOLAB*, corresponding values are measured by instrumenting the agent program code and by usage of the tool *GrepAg* [22] as described in section 5.3.3, page 94. Agent code instrumentation provides values for r_A per server and for rtt_A. *GrepAg* measures r_N per server and rtt_N. Network delay per agent can be calculated by $d_{net} = rtt_N - \left(\sum r_N\right)$.

The model is validated by a finite horizon analysis. Measurement, resp. simulation ends after 10,000 seconds.

6.2.5 Model Validation and Calibration

Initially, model results of both workload scenarios are too optimistic, i.e. residence times and round trip times are too low. This is due to the fact that model parameters were extracted from a system with very low workload. With increasing workload intensity, overhead arises at the servers. Hence, parameters "service rates" of system and migration components, "ghost migration delay", "service amount for DNS look up" and "service rates" of DNS server are used to calibrate the model. Figure 6-4 shows the parameterisation of the infrastructure after calibration. It could be observed that agent server *bender* shows delays at the migration component which cannot be modelled by the adjustment of service rates. Approximately every 200 agents there are 10 agents with an additional mean delay of 3 seconds. This effect is modelled by the ghost delay which is introduced in section 2.4.1, page 18.

Figure 6-5 through figure 6-7 show average results of the real system and of simulation for residence times (r_A and r_N), round trip times (rtt_A and rtt_N) and network delay after calibration. Significant differences between r_A and r_N at each server in both workload scenarios point out the high impact of migration overhead on residence time r_N. Migration overhead can be calculated by $r_N - r_A$. In fact, system and migration overhead dominate the resource consumption by user load. Furthermore, it is obvious that server *bender* is the bottleneck with lower workload intensity, whereas *bond* is the bottleneck with higher intensity. Residence times r_N of servers "bender" and "bond" clarify this. This again is due to delays in the migration component.

1. For a definition of types of response times see section 5.1.4, page 90

Infrastructure | Service Agents | Workload Generation | Output Analysis

Agent Servers

Adress	NoCPUs	Qu...	Rates Usr	Rates Sys	Rates Mig	Ghost Delay [s]	ON [ma]	OFF [ma]	Ovh [s]	Type	Aggregate
bond.infor...	1	100	1.0	0.8, 0.16	1.0, 0.75, 0.425	0	0	0	0	File Server	✔
goldeneye....	1	100	1.0	1.0	1.0	0	0	0	0	File Server	✔
goldfinger.i...	1	100	1.0	0.74	1.0, 0.8, 0.19, 0.31, 0.32	0	0	0	0	File Server	✔
bender.inf...	1	100	1.0	1.0, 0.34, 0.48	1.0, 0.85	3	10	200	0	File Server	✔

Add | Delete

Network

Routing table: aDEMAS\etc\RoutingTableMOLAB_Extended.txt | Browse ... | ○ Simple ◉ Extended, Avg. window size 3 KB

DNS Server

Service amount: 0.00025 sec | Service rates: 1.0

Main Class

Init system by class: aDEMAS\bin\de\mt\timetable\Main.class | Browse ...

Method: main | Parameters: -m

Figure 6-4: **Parameters of infrastructure after calibration**

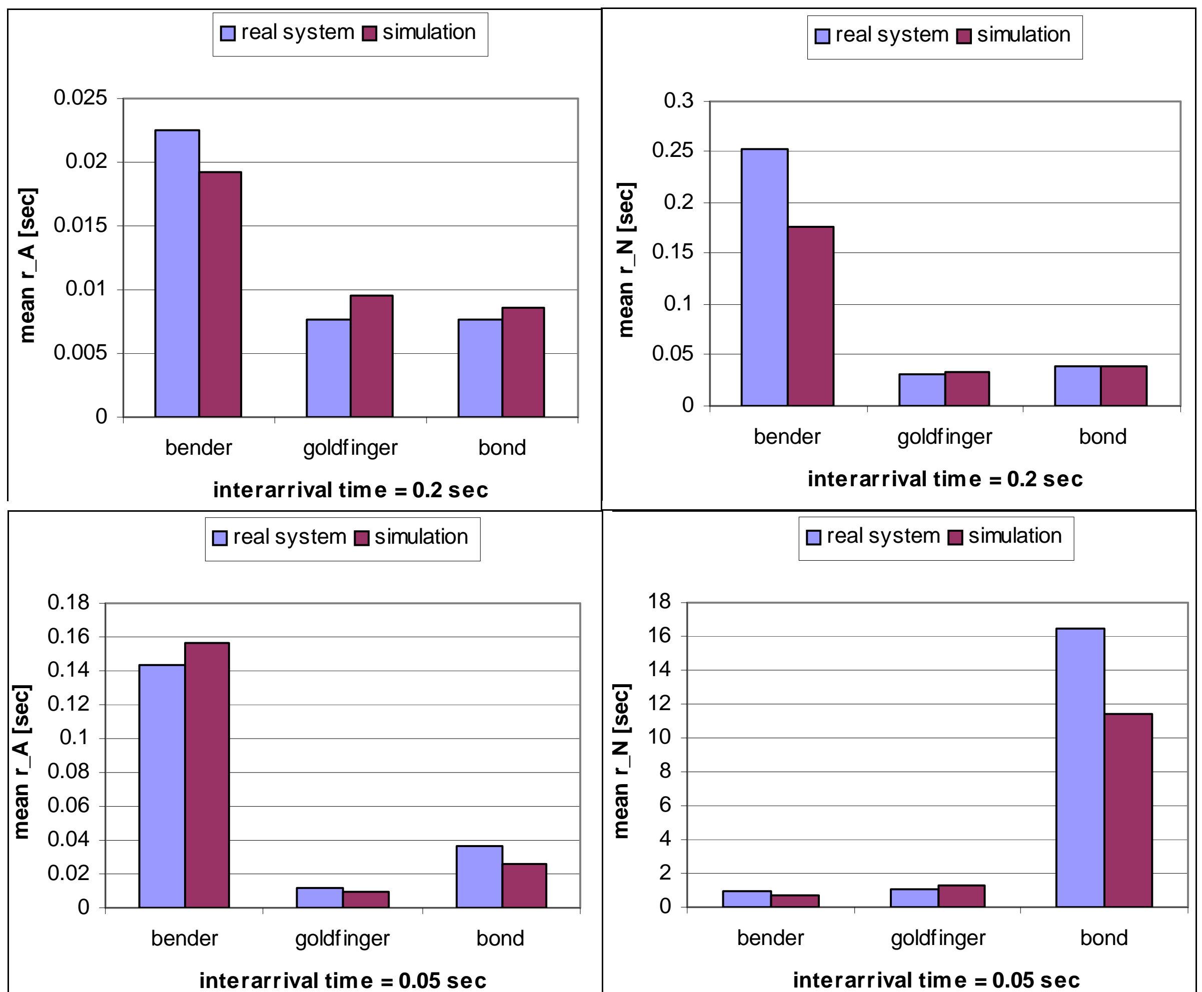

Figure 6-5: **Mean residence times**

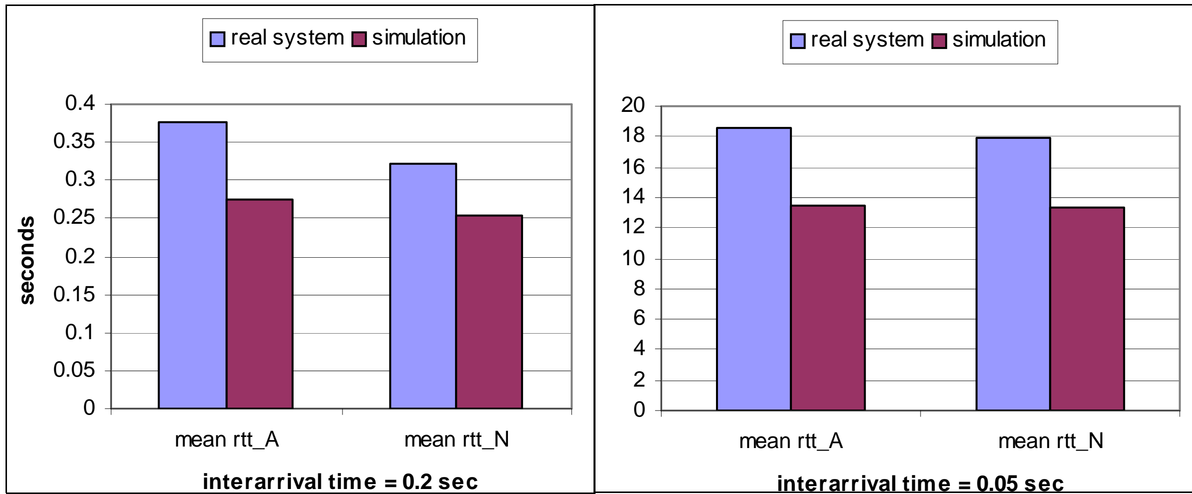

Figure 6-6: **Mean round trip times**

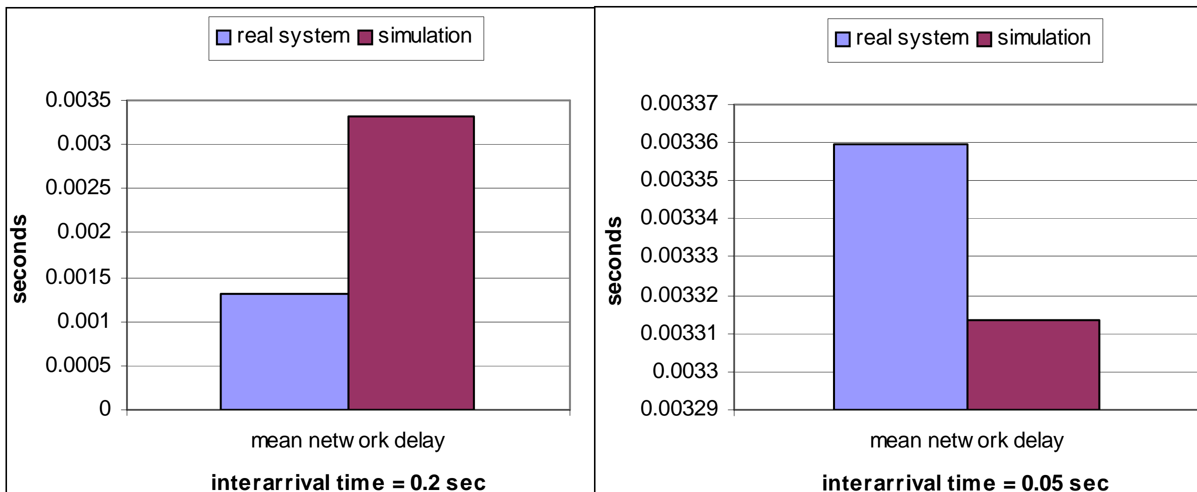

Figure 6-7: **Mean network delays**

Table 6-1 through table 6-3 compare results of real system and baseline model by showing differences in percent. Differences up to ca. 30% concerning mean values of residence time and round trip time are tolerable (compare section 5.1.5, page 91). Thus, the baseline model can be regarded calibrated concerning residence and round trip time.

The difference of 150.93% of network delay in the scenario with lower workload intensity is significant. Network parameters could not be calibrated to fit for both modelled scenarios. A closer look at the measured network delays reveals measurement inaccuracies which have a high impact on the very small network delays with lower workload intensity. Hence, it is feasible to calibrate the baseline model with respect to the higher workload scenario.

	interarrival time = 0.2		interarrival time = 0.05	
	rt_A	rt_N	rt_A	rt_N
bender	-14.48%	-30.48%	9.37%	-25.44%
goldfinger	25.20%	8.25%	-21.63%	21.79%
bond	12.98%	3.37%	-27.60%	-31.04%

Table 6-1: **Differences of residence times**

interarrival time = 0.2		interarrival time = 0.05	
rtt_A	rtt_N	rtt_A	rtt_N
-27.03%	-21.66%	-27.91%	-25.16%

Table 6-2: **Differences of round trip times**

interarrival time = 0.2	interarrival time = 0.05
150.93%	-1.37%

Table 6-3: **Differences of network delay**

6.3 Performance Prediction

The goal of the case study is to investigate if the planned system, i.e. the infrastructure and the application fulfils the recommended performance requirements. To answer this question, the baseline model is extended to a prediction model which accords to the planned system.

6.3.1 Specification of Future Scenarios

The planned system consists of 10 agent servers with one faculty data base, each. 4 servers are of the same type as the ones in *MOLAB*. 6 additional servers are planned to possess the same capacity as server *bender*. Again, servers shall be fully meshed in a Fast-Ethernet with bandwidth of 100 MBit/sec.

The mobile agent system shall be confronted with a stress test. It shall be assumed that 22,500 students will use *Collegiate Timetable Service*. Each student shall averagely access the application twice. It should be assured that the system copes with the worst-case scenario that all students access *Collegiate Timetable Service* twice within one hour. This results in an arrival rate of 45,000 requests per hour. To evaluate the worst-case scenario, a steady state, i.e., a batch means analysis is executed. *JaDEMAS* is parameterised to calculate a confidence interval of round trip times with a confidence level of 0.9 and to stop simulation when a relative statistical error of 0.15 is reached.

In a second scenario, round trip times shall be observed with different workload levels spread on a single day at the beginning of a semester. Workload peaks are assumed between 10 and 12 o'clock a.m. when students will have got up in the morning and after lunch between 2 and 4 o'clock p.m.. Figure 6-8 shows the expected variation of arrival rate λ. A finite horizon simulation is executed to analyse this scenario. The number of necessary simulation runs is determined

by the relative statistical error of round trip time which is again set to 0.15. Requests are expected to be quite bursty in both workload scenarios, i.e. the c.o.v of interarrival times is set to 6.0.

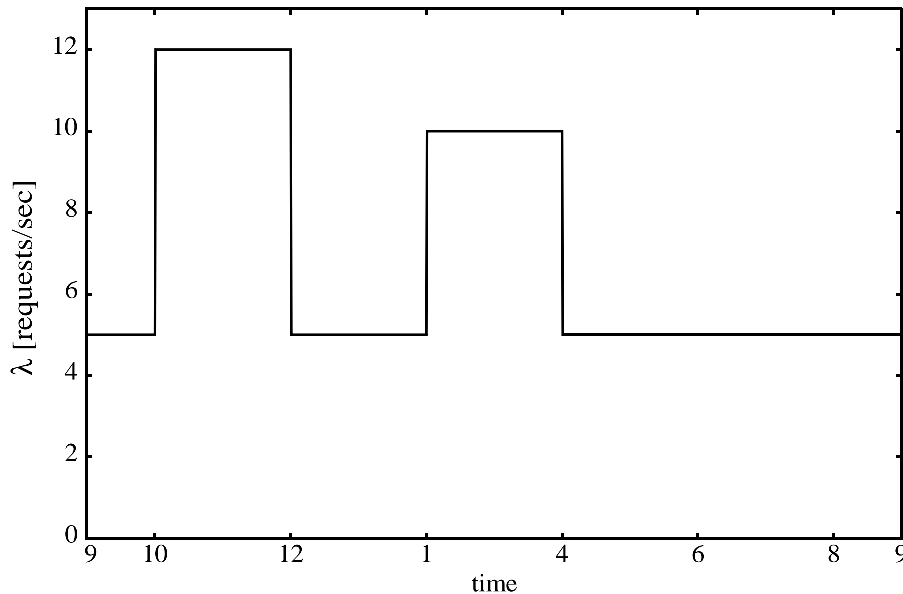

Figure 6-8: **Expected workload variation on a single day**

6.3.2 Quality of Service Requirements

With the worst-case scenario, the university wants to observe mean and maximum round trip time of mobile agents. Since servers probably get into overload situations and because this scenario is not likely to happen very often, service requirements are set not too strictly: The mean round trip time rtt_{mean} shall not be above 8 seconds, the maximum value rtt_{max} shall not exceed 10 seconds.

With the second scenario, mean round trip time shall not be above 8 seconds at all workload levels. Additionally, 80% of the values shall be below 5 seconds.

6.3.3 Execution of Experiments

First, the baseline model has to be extended to a prediction model. Therefor, additional agent servers have to be parameterised in *JaDEMAS*. According to the planned system, their parameters can be copied from server *bender*. Furthermore, the application has to be extended to its full dimension. This means, additional faculty data bases have to be provided, the workload generator and the initialisation classes have to be extended to generate requests to the additional data bases and to initialise service agents at the new servers. Finally, the prediction model is simulated with the workload scenarios previously specified.

6.3.4 Analysis of Results

The batch means analysis of the worst-case scenario results in a confidence interval of round trip time $CI(rtt) = [15.562 \text{ sec}, 21.022 \text{ sec}]$. The maximum round trip time is $rtt_{max} = 69.806 \text{ sec}$. This result is unacceptable compared to the quality of service requirements of $rtt_{mean} \leq 8 \text{ sec}$ and $rtt_{max} \leq 10 \text{ sec}$.

The reason for this bad performance can be found by a closer look at the utilisation of server resources and residence times (see figure 6-9 and figure 6-10).

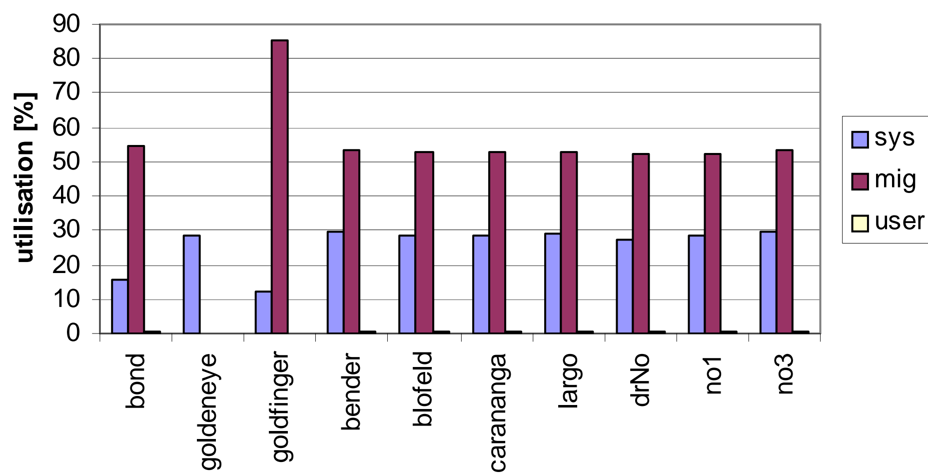

Figure 6-9: **Utilisation of server resources**

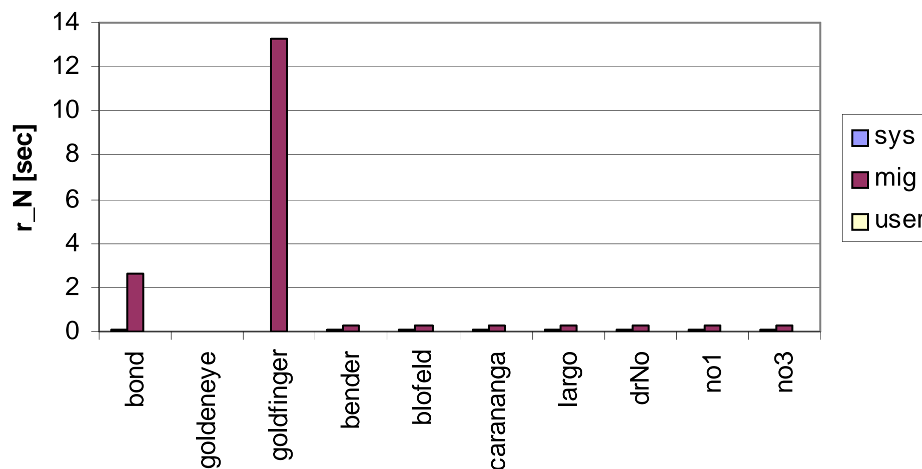

Figure 6-10: **Mean residence times**

goldfinger's resource which is responsible for migration is utilised by 85.5%. This is reflected in the mean residence time (13.269 seconds) of mobile agents at *goldfinger*'s migration component, too. This means, that *goldfinger* is the bottleneck of the planned system.

There are usually two possibilities to solve such a performance problem. Either the infrastructure of the agent system or the application itself has to be changed. A modification of the application could include, e.g., to split the faculty data base of *goldfinger* and transfer a ratio to another server to decrease arrival rates at *goldfinger*. Alternatively, *goldfinger* could be replaced by a faster server. The latter alternative is chosen.

Hence, the agent system in the prediction model is tuned by doubling the capacity of server *goldfinger*. Now, the confidence interval of round trip time is $CI(rtt) = [13.319\text{ sec}, 17.988\text{ sec}]$. The maximum round trip time is $rtt_{max} = 69.200\text{ sec}$. Again, results are not satisfying. What happened by this system tuning, is a bottleneck movement, see figure 6-11 and figure 6-12.

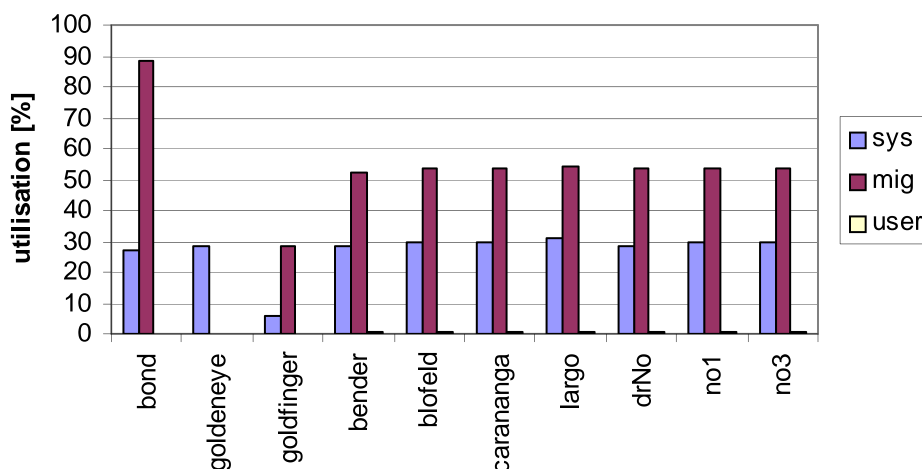

Figure 6-11: **Utilisation of server resources**

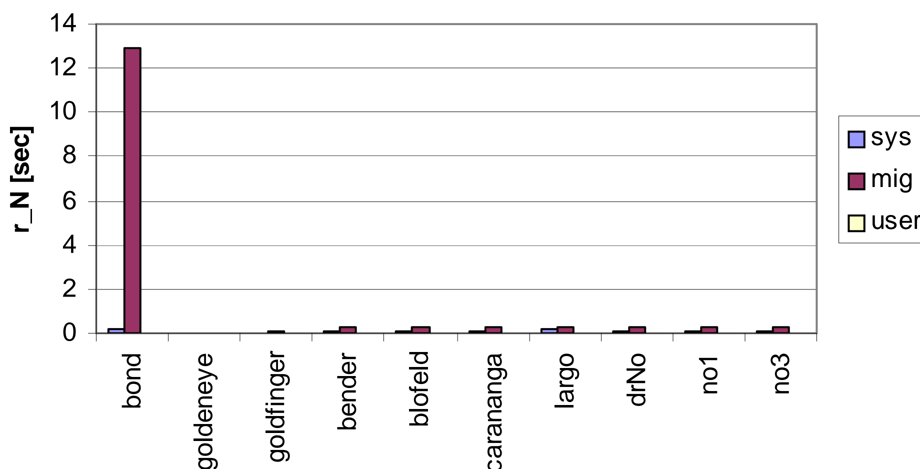

Figure 6-12: **Mean residence times**

Now, the migration component of server *bond* is the bottleneck. It shows a utilisation of 88.5% and a mean residence time of 12.904 seconds.

Hence, the model is again modified, this time by doubling the capacity of server *bond*. With this second tuning the confidence interval of round trip time results in

$CI(rtt) = [4.240 \text{ sec}, 5.344 \text{ sec}]$, mean value $rtt_{mean} = 4.792 \text{ sec}$. The maximum round trip time is $rtt_{max} = 6.563$. These results are satisfying according to the quality of service requirements, i. e. a system configuration which meets the worst-case requirements is found.

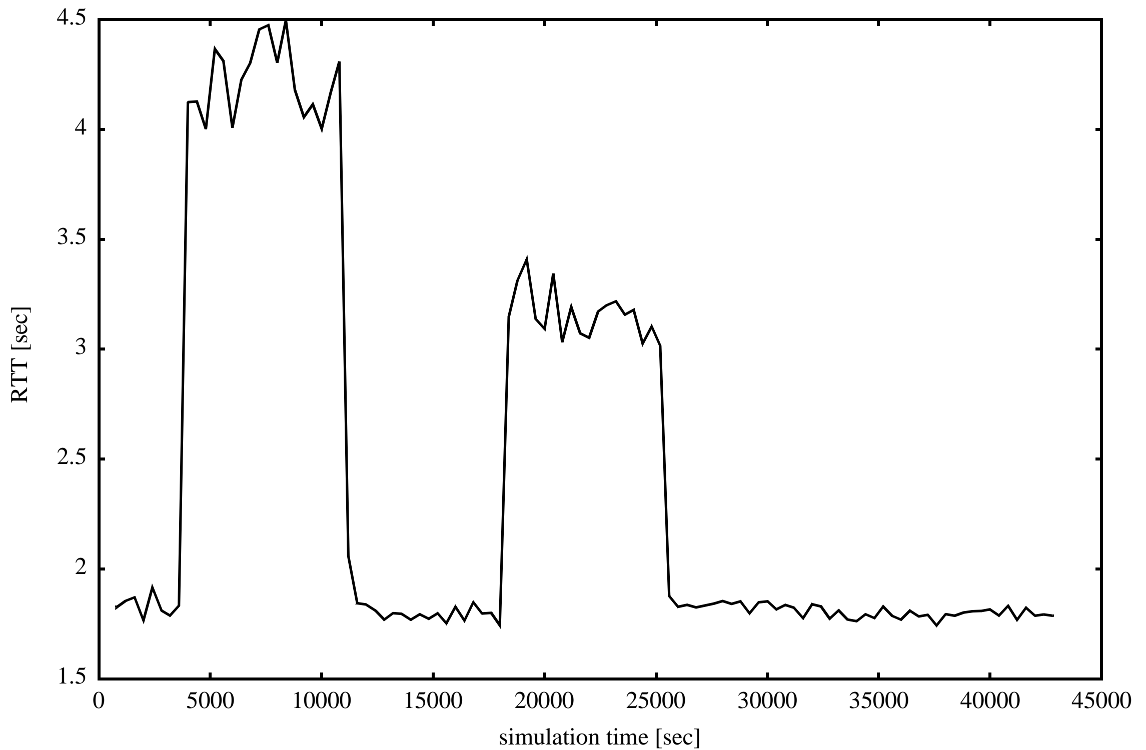

Figure 6-13: **Mean round trip times (averaged over 400 seconds and 10 replication runs)**

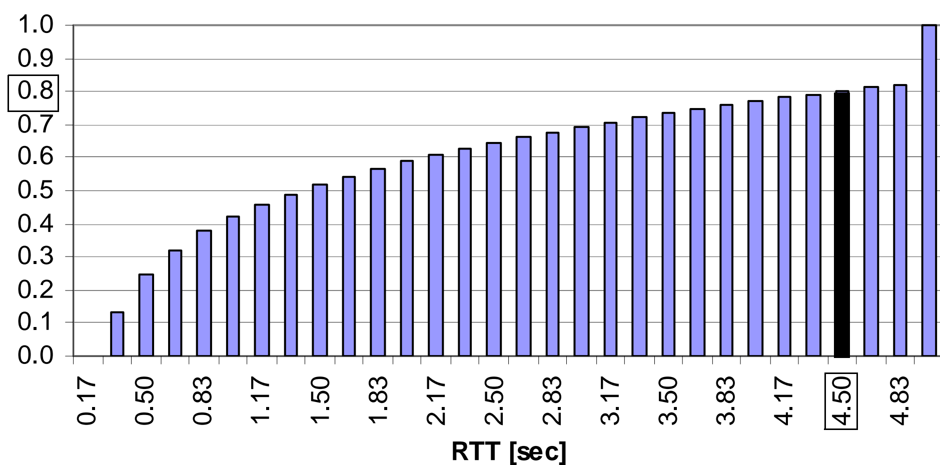

Figure 6-14: **Distribution of round trip times (gained from 10 replication runs)**

Now, the finite horizon analysis is performed assuming the tuned system configuration. In fact, this configuration provides the required performance. Figure 6-13 and figure 6-14 show the fulfilment of the performance requirements: Figure 6-13 clearly describes the reaction of the round

trip times on the workload peaks in the morning and afternoon. Nevertheless, mean round trip time is below 4.5 seconds with all workload levels ($rtt_{mean} \leq 8$ sec is required). Figure 6-14 shows the distribution of round trip times and allows for the evaluation of quantiles. 80% of all round trip times are lower than 4.5 seconds. It is required that 80% of the round trip time should be below 5 seconds. Hence, the current configuration of the prediction model meets the service requirements for the planned system. Thus, the planned real system should be implemented with respect to this configuration.

6.4 Summary

This chapter illustrates the phases of the capacity planning process for mobile agent systems by a case study. The capacity of an information retrieval system for students at a university is planned, considering certain quality of service requirements. A baseline model is built and calibrated according to a real system which is implemented in the laboratory *MOLAB*. This model is extended corresponding to the planned real system. The infrastructure first planned cannot provide the desired performance requirements. Two servers show up to be the bottlenecks. It is decided to exchange these two with faster ones and, actually, quality of service requirements are met. Finally, the system configuration for the planned system is found.

It is shown that the developed techniques and the capacity planning methodology are applicable with a manageable effort. This includes that a modeller should be able to build performance models during the development of the agent system with minor effort.

7 Summary and Outlook

This dissertation presents methods and techniques for capacity planning of mobile agent systems. Mobile agent systems are used to implement distributed applications. One important goal of this dissertation is to enable system developers to plan the capacity of future systems as early as possible. Unfortunately, capacity planning of distributed applications is hardly done in time. This often leads to severe performance problems when the application is already running in a production environment. Thus, problems can be corrected only afterwards, which is a very costly procedure. System developers often avoid to analyse performance aspects before application roll-out because methods and tools for performance modelling fundamentally differ from known methods and tools for system development. This dissertation approaches methods for performance modelling to methods for system development.

7.1 Techniques for Performance Modelling

Chapter 2 presents new techniques for performance modelling of planned mobile agent systems which can be applied when the program code of the application is implemented. Thereby, the basic idea is to transfer real agent's program code directly into the performance model. To follow this approach and due to the specific characteristics of mobile agents, discrete event simulation has been chosen for performance modelling. The simulation environment *JaDEMAS* has been developed to implement this approach.

JaDEMAS is based on the simulation package *JavaDEMOS* [26] and it is designed to model mobile agent systems implemented with the *Tracy* platform [9], version 0.61. *JaDEMAS* internally models communication, waiting processes, contention scenarios and scheduling strategies. The modeller does not have to model these operations himself. *JaDEMAS* provides two types of agent servers: file and compute servers. Moreover, there exist two types of network models: a simple one where only bandwidth between linked servers has to be specified and an extended network model which models a TCP pipe between source and destination server. The modelled mechanisms for communication, migration, scheduling should be similar in most mobile agent

platforms. Hence, mapping *JaDEMAS* to another system should mainly include an adjustment to the API.

The developed methodology for capacity planning and the modelling techniques are designed for mobile agent systems which implement intranet applications. Originally, the paradigm of mobile agents has been developed for the implementation of distributed applications in the Internet. However, for security reasons, its applicability might be limited there. Beside all security precaution of the mobile agent software, operators will hardly risk access to their resources by foreign agents which could be disguised trojaners or another type of malicious code. Hence, it rather can be assumed that mobile agent systems will actually be applied in intranets where system operators and agent developers are from the same organisation. This is the case when mobile agent systems are installed in intranets. Typical scenarios which can be modelled with *JaDEMAS* are as follows: A single network access point is assumed (mobile agents' home server). Upon user requests, mobile agents are sent out from their home server and finally return to their origin. They are sent out to perform service according to user requests. At each agent server they contact service providing, stationary agents.

As commonly known, simulation reaches its limits with very large or complex models. Chapter 3 shows existent approaches to increase the efficiency of performance models. These approaches could be applicable in the context of this dissertation, at a first glance. But, it turns out, that none of them is suitable with regards to the objectives set in this dissertation. Especially, the *SHRiNK* approach [46] has been investigated quite detailed because it first seemed to be applicable. However, experiments show that this approach does not generally provide an improvement of the model efficiency.

Hence, chapter 4, presents new techniques to increase simulation efficiency by using hybrid modelling techniques. In particular, models of large agent systems benefit from these techniques. The so-called *CFS* approach aggregates agent servers with primary file server functions to building blocks which can be analysed analytically. The *CRRS* approach simulates agents servers which mainly provide CPU power with a reduced number of events. It can be shown that valid models can be built with this approaches and that significant efficient gains are provided. *CFS* and *CRRS* can as well be applied to general queuing networks where network delays can be calculated analytically. Moreover, the approaches allow for modelling of general stochastic patterns and for multiple output analysis techniques.

7.2 Methodology for Capacity Planning

Chapter 5 embeds the developed approaches for performance modelling into the capacity planning process. It describes the phases of capacity planning by using the developed modelling approaches. Furthermore, it shows how measurements have to be planned and executed, so that measurement results directly can be transferred to simulation models as input parameters or to compare performance results. The methodology for capacity planning is based on [6], pp. I-31 - I-38. It is transferred to the context of mobile agent systems using *JaDEMAS* for performance modelling.

The basic steps of the capacity planning process are the following: First, a baseline performance model is developed. This model is calibrated by comparing its results with performance values measured in a reference real agent system. This reference system can be an application similar to the future one which is currently running in a production environment or in a testbed implementation. Chapter 5 describes *MOLAB*, a mobile agent laboratory which provides an infrastructure

and measurement tools for testbed applications. Next, the infrastructure of the planned agent system and relevant workload scenarios have to be specified. Then, the baseline model is extended to a prediction model according to this specifications. Furthermore, quality of service requirements for the future system have to be defined. Finally, it is analysed if the prediction model fulfils these requirements. If not, it is modified until the performance meets the specified service level requirements. Then, the necessary system capacity is found and the planned system should be implemented with respect to this configuration.

7.3 Applicability of the Approaches

The applicability of the developed approaches is demonstrated by a case study. A mobile agent system has been developed which implements an information retrieval application. The application is called *Collegiate Timetable Service*. It allows students to request automatically generated timetables according to the courses they wish to attend at the university. Chapter 6 describes a case study which shows that the developed methods are applicable with manageable effort in a realistic environment. Therefor, the methodology presented in Chapter 5 is applied step by step to plan the capacity of the future mobile agent system. It turns out that the system planned initially, is not able to provide the required performance. An agent server is identified as the bottleneck and it is exchanged by a faster one in the model. This results in a movement of the bottleneck to another server. After replacing the second server by a faster one, the service requirements are fulfilled.

7.4 Outlook into Future Research

If mobile agent applications develop in a direction which is not covered by the scenarios modelled in this dissertation, the modelling approaches should be adapted to this development. Furthermore, *JaDEMAS* should be adapted if a real standard for the architecture of mobile agent systems will be specified. This standard should allow for the exchange of mobile agents between different platforms. Then, there should be identical application programming interfaces which had to be transferred to *JaDEMAS*. Furthermore, the migration and network model of *JaDEMAS* could be extended. A concrete extension could be the integration of *Tracy*'s *Kalong* component into *JaDEMAS*.

If a planned mobile agent system is implemented and running in a production environment, it could be checked if predicted performance values are met in reality. Therefor, performance values had to be monitored in the real system and compared to model results to evaluate the quality of the prediction model.

List of Figures

Figure 1-1: Generation of mobile agents with intranet applications 3
Figure 2-1: Typical model scenario 8
Figure 2-2: Concept of agent system simulation model 10
Figure 2-3: Process of agent trip (example: mobile agent visiting two servers) 12
Figure 2-4: Menu of diagram symbols (according to [48] with modifications) 15
Figure 2-5: Simulation of a file server 16
Figure 2-6: Simulation of a CPU 17
Figure 2-7: Modelling of multi processors 17
Figure 2-8: Structure of work load 18
Figure 2-9: Serving of mobile agents by system agents 19
Figure 2-10: Integration of real agent classes in JaDEMAS 20
Figure 2-11: Inheritance hierarchy of agents in JaDEMAS with code fragments of JavaDEMOS 20
Figure 2-12: Simulation of interaction between mobile and service agent 21
Figure 2-13: Simulation of network links 22
Figure 2-14: Infrastructure of the mobile agent system 28
Figure 2-15: Agent classes and allocation to home servers 29
Figure 2-16: Parameterising of workload 29
Figure 2-17: Parameterising of output analysis 30
Figure 2-18: Modifications with the code of the system agent *ServSys1* *30*
Figure 2-19: Mean round trip time (single simulation run) 31
Figure 2-20: Distribution of round trip time (RTT) (single simulation run) 32
Figure 2-21: Utilisation of server "bond" (single simulation run) 32
Figure 2-22: System throughput (single simulation run) 33
Figure 3-1: Process of aggregation 36
Figure 3-2: Detailed M/G/1 model 41
Figure 3-3: SHRiNK M/G/1 model 42
Figure 4-1: The concept of directly concatenating agent servers 49
Figure 4-2: Detailed process of serving mobile agents by service agents 50
Figure 4-3: Time consumption with serving mobile agents by service agents 50
Figure 4-4: Modelling of time consumption with concatenated agent servers 51
Figure 4-5: Composition of the residence time of job n 52
Figure 4-6: Example of the calculation of FIFO server residence time 54
Figure 4-7: Update of throughput and mean service in detailed model and CFS 55
Figure 4-8: Detailed model versus CFS (1) 57
Figure 4-9: Detailed model versus CFS (2) 58
Figure 4-10: Modelled agent system 60
Figure 4-11: Utilisation of agent server "Bond" in detailed and CFS models 61
Figure 4-12: 90% confidence intervals of round trip times of detailed and CFS G/G/1 models 62
Figure 4-13: CPU time consumption of detailed and CFS models 63

Figure 4-14: Modelled agent system 64
Figure 4-15: Round trip times in an G/G/1 network (mean values over 100 runs) 65
Figure 4-16: Throughput in an G/G/1 network (mean values over 100 runs) 66
Figure 4-17: Relative frequency of RTT in an G/G/1 network (mean values over 100 runs) . 66
Figure 4-18: Utilisation of server "Bond" with mobile agents of 150 KB 68
Figure 4-19: Utilisation of server "Bond" with mobile agents of 2 MB 68
Figure 4-20: Utilisation of server "Bond" with mobile agents of 5 MB 69
Figure 4-21: Average amount of CPU time per simulation run 69
Figure 4-22: Example for a too small cumulative delay 73
Figure 4-23: Approximation error if not recognising quantum allocation 74
Figure 4-24: Utilisation of detailed and CRRS models 76
Figure 4-25: 90% confidence intervals of round trip times of detailed and CRRS G/G/1 models 76
Figure 4-26: CPU time consumption of detailed and CRRS models 77
Figure 4-27: RTTs in an G/G/1 network (averaged over 100 runs) 79
Figure 4-28: Throughput in an G/G/1 network (averaged over 100 runs) 79
Figure 4-29: Utilisation of server "Bond" (averaged over 100 runs) 80
Figure 4-30: Empirical distribution of RTT in an G/G/1 network (averaged over 100 runs) . 80
Figure 4-31: Average amount of CPU time per simulation run 82
Figure 5-1: Process of capacity planning ([6], p. I-32, modified) 86
Figure 5-2: Measurement of time consumption 88
Figure 5-3: Measurement of performance values 90
Figure 5-4: Mobile agent laboratory MOLAB with an exemplary agent route 94
Figure 6-1: Architecture of Collegiate Timetable Service, according to [58] 98
Figure 6-2: Snapshot of routing table 99
Figure 6-3: Specification of infrastructure 100
Figure 6-4: Parameters of infrastructure after calibration 103
Figure 6-5: Mean residence times 103
Figure 6-6: Mean round trip times 104
Figure 6-7: Mean network delays 104
Figure 6-8: Expected workload variation on a single day 106
Figure 6-9: Utilisation of server resources 107
Figure 6-10: Mean residence times 107
Figure 6-11: Utilisation of server resources 108
Figure 6-12: Mean residence times 108
Figure 6-13: Mean round trip times (averaged over 400 seconds and 10 replication runs) . 109
Figure 6-14: Distribution of round trip times (gained from 10 replication runs) 109
Figure A-1: Mean residence times in M/G/1 model with c.o.v.[S] = 4.0 126
Figure A-2: Utilisation in M/G/1 model with c.o.v.[S] = 4.0 (averaged over 500 time units) . . 126
Figure A-3: Mean residence times in M/G/1 model with c.o.v.[S] = 6.0 128
Figure A-4: Utilisation in M/G/1 model with c.o.v.[S] = 6.0 (averaged over 500 time units) . . 129
Figure A-5: RTTs in an M/H4/1 network (averages over 10 replications) 135

Figure A-6: Throughput in an M/H4/1 network (averages over 10 replications) 135
Figure A-7: Utilisation of server "Bond" in a 10 Mbit network with mobile agents of 150 KB (averages over 10 replications) 136
Figure A-8: RTTs in an G/D_4/1 network (averages over 100 replications) 138
Figure A-9: Throughput in an G/D_4/1 network (averages over 100 replications) 138
Figure A-10: Utilisation of server "Bond" in a 10 Mbit network with mobile agents of 150 KB (averages over 100 replications) 139
Figure A-11: RTTs in an M/H4/1 network (averages over 15 replications) 143
Figure A-12: Throughput in an M/H4/1 network (averages over 15 replications) 144
Figure A-13: Utilisation of server "Bond" in a 10 Mbit network with mobile agents of 150 KB (averages over 15 replications) 144
Figure A-14: RTTs in an G/D_4/1 network (averages over 100 replications) 146
Figure A-15: Throughput in an G/D_4/1 network (averages over 100 replications) 146
Figure A-16: Utilisation of server "Bond" in a 10 Mbit network with mobile agents of 150 KB (averages over 100 replications) 147

List of Tables

Table 3-1: Comparison of steady state results of SHRiNK and detailed M/G/1 models 42
Table 3-2: Comparison of efficiency of SHRiNK and detailed M/G/1 models 43
Table 3-3: Variance with different a 44
Table 3-4: Features of Mobile Agent System Analysis and Consequences for Simulation 45
Table 4-1: Paired-t confidence intervals of differences between CFS and detailed model 62
Table 4-2: Comparison of efficiency of CFS and detailed model 63
Table 4-3: Paired-t confidence intervals of differences between CFS and detailed model (based on 100 replication runs) 67
Table 4-4: Efficiency gains of CFS 70
Table 4-5: Paired-t confidence intervals of differences between CRRS and detailed model 77
Table 4-6: Comparison of efficiency of CRRS and detailed model 78
Table 4-7: Paired-t confidence intervals of differences between CRRS and detailed model (based on 100 replication runs) 81
Table 4-8: Amount of CPU time 82
Table 5-1: Reasonable tolerances in validation (from [37], table 12.3, p. 292) 91
Table 6-1: Differences of residence times 105
Table 6-2: Differences of round trip times 105
Table 6-3: Differences of network delay 105
Table A-1: Differences of mean residence times of SHRiNK models compared to detailed M/G/1 model (c.o.v.[S] = 4.0) with 70 replication runs 127
Table A-2: Differences of mean residence times of SHRiNK models compared to a detailed M/G/1 model (c.o.v.[S] = 6.0) with 110 replication runs 130
Table A-3: Efficiency gains of SHRiNK vs. detailed M/G/1 model 130
Table A-4: Confidence intervals (CI) of mean residence time of M/G/1 models (c.o.v.[S] = 4.0). 70 replication runs with a = 1, 80 replication runs with a = 0.5, 280 replication runs with a = 0.167 131
Table A-5: Confidence intervals (CI) of mean residence time of M/G/1 model (c.o.v.[S] = 6.0). 110 replication runs with a = 1, 240 runs with a = 0.5, 640 runs with a = 0.167 132
Table A-6: Comparison of steady state results of CFS and detailed model 133
Table A-7: Comparison of efficiency of CFS and detailed model 133
Table A-8: Confidence intervals (CI) of mean residence time (10 replication runs) 134
Table A-9: Differences of mean RTT of CFS model compared to detailed model (10 replication runs) 134
Table A-10: Confidence intervals (CI) of mean residence time (100 replication runs) . . 137
Table A-11: Differences of mean RTT of CFS model compared to detailed model (100 runs) 137
Table A-12: Confidence intervals (CI) of mean residence time (100 replication runs) . . 140
Table A-13: Amount of CPU time per run 140
Table A-14: Confidence intervals of round trip time 141
Table A-15: Amount of CPU time 141

Table A-16: Confidence intervals (CI) of mean residence time (15 replication runs) 142
Table A-17: Differences of mean RTT of CRRS model compared to detailed model (15 replication runs) 143
Table A-18: Confidence intervals (CI) of mean residence time (100 replication runs) . . 145
Table A-19: Differences of mean RTT of CRRS compared to detailed model (100 runs) 145
Table A-20: Confidence intervals (CI) of mean residence time (100 replication runs) . . 147
Table A-21: Amount of CPU time 148

Bibliography

[1] S. C. Agrawall, J. P. Buzen, A. W. Shum: Response Time Preservation: A General Technique for Developing Approximate Algorithms for Queueing Networks. Proceedings of the 1984 ACM SIGMETRICS conference on Measurement and modeling of computer systems, 1984, pp 63 - 77. ISBN:0-89791-141-5.

[2] N. C. Audsley, A. Burns, M. F. Richardson, A. J. Wellings: *Hard Real-Time Scheduling: The Deadline-Monotonic Approach*. Report YCS-90-146, Department of Computer Science, York University, 1990.

[3] F. Baskett, K. M. Chandy, R. R. Muntz, F. G. Palacios: *Open, closed and mixed networks of queues with different classes of customers*. J. ACM, 22, pp. 248-260, 1975.

[4] G.M. Birtwistle: *DEMOS - A System for Discrete Event Modelling on Simula*. Macmillan, 1985.

[5] G. Bolch, S. Greiner, H. de Meer, K. Trivedi: *Queueing Networks and Markov Chains - Modeling and Performance Evaluation with Computer Applications*. John Wiley & Sons, Inc., 1998.

[6] R. Bordewisch, C. Flüs, R. Grabau, J. Hintelmann, K. Hirsch, H. Risthaus: *Methodik und Verfahren*. In: B. Müller-Clostermann: Kursbuch Kapazitätsmanagement, pp. I-26 - I-61. BoD, 2001, ISBN 3-8311-2823-5.

[7] P. Braun: *The Migration Process of Mobile Agents - Implementation, Classification, and Optimization*. Dissertation, Friedrich-Schiller-Universität Jena, May 2003.

[8] P. Braun, I. Müller, S. Geisenhainer, V. Schau, W. Rossak: *A Service-oriented Software Architecture for Mobile Agent Toolkits*. Workshop on Security, Interoperability, and Applications of Mobile Agent Systems (SIAMAS 2004) at the 11th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2004) May 2004, Brno, Czech Republic.

[9] P. Braun, C. Erfurth, W. Rossak: *An Introduction to the Tracy Mobile Agent System*. Technical Report No. 24/2000. Friedrich-Schiller-Universität Jena, Institut für Informatik, September 2000

[10] P. Braun, J. Eismann, C. Erfurth, W. Rossak: *Tracy -- A Prototype of an Architected Middleware to Support Mobile Agents*. Proceedings of the 8th Annual IEEE Conference and Workshop on the Engineering of Computer Based Systems (ECBS), Washington D.C. (USA), April 2001

[11] L. Cardelli: *A Language with Distributed Scope. Computing Systems*, 8(1):27-59, 1995.

[12] A. Carzaniga, G. P. Picco, G. Vigna: *Designing Distributed Applications with Mobile Code Paradigms*. Proceedings of the 19th International Conference on Software Engineering, 1997.

[13] C. H. Chandy, U. Herzog, L. Woo: *Parametric Analysis of Queuing Network Models*. IBM Journal Res. Dev. 19, 1, pp. 36-42, 1975.

[14] P. Courtois: *Decomposability: Queueing and Computer System Applications*. Academic Press, New York, 1977.

[15] E. W. Dijkstra: *A Note on Two Problems in Connection with Graphs*. Numerische Mathematik, Vol. 1, pp. 269-271, Springer-Verlag, Berlin, 1959.

[16] M. Dikaiakos, M. Kyriakou, G. Samaras: *Performance Evaluation of Mobile Agent Middleware*. Proceedings of the 5th International Conference on Mobile Agents (MA 2001), Atlanta (USA), December, 2001.

[17] C. Flüs: *Reduced Rescheduling Technique for Efficient Simulation of Round Robin Servers*. In: P. Buchholz, R. Lehnert, M. Pioro: 12th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB) together with 3rd Polish-German Teletraffic Symposium (PGTS). 12-15 September 2004, Technische Universitaet Dresden, Germany, pp. 125-134. VDE Verlag, 2004. ISBN 3-8007-2851-6.

[18] C. Flüs, B. Müller-Clostermann, M. Vilents: *VITO: A Tool for Capacity Planning of Computer Systems*. MMB'99, 10. GI/ITG-Symposium 22.-24.09.1999 in Trier. Research report No. 99-17, short papers and tool descriptions.

[19] P. Glasserman, P. Heidelberger, P. Shahabuddin: *Gaussian Importance Sampling & Stratification: Computational Issues*. Proceedings of the 1998 Winter Simulation Conference, IEEE Press, pp. 685-693, New York 1998.

[20] *Grasshopper* home page: http://www.grasshopper.de/

[21] R. S. Gray, G. Cybenko, D. Rus: *Mobile Agents: Motivation and State of the Art*. Technical Report TR2000-365, Department of Computer Science, Dartmouth College.

[22] *GrepAg* homepage: http://www.cs.uni-essen.de/SysMod/GrepAg/

[23] L. Himmelblau: *Process Analysis by Statistical Methods*. Wiley, New York, 1970.

[24] F. Hohl, P. Klar, J. Baumann: *Efficient code migration for modular mobile agents*. In: J. Bosch (Ed.): 3rd ECOOP Workshop on Mobile Object Systems: Operating System Support for Mobile Object Systems. Jyvälskylä (Finnland), June 1997. Vol. 1357 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1998.

[25] http://www.agentlink.org/

[26] *JavaDEMOS* home page: http://www.informatik.uni-essen.de/SysMod/JavaDemos/

[27] E.-N. Huh: *Certification of Real-Time Performance for Dynamic, Distributed Real-Time Systems*. Dissertation, Faculty of the Fritz J. and Dolores H. Russ College of Engineering and Technology, Ohio University, 2002.

[28] A. Iqbal, J. Baumann, M. Straßer: *Efficient Algorithms to Find Optimal Agent Migration Strategies*. Universität Stuttgart, Fakultät Informatik, Bericht Nr. 1998/05.

[29] R. Jain: *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc.; 1991. ISBN 0-471-50336-3.

[30] D. Johansen, N. P. Sudmann, R. van Renesse: *Performance Issues in TACOMA*. Proceedings of the 3rd ECOOP Workshop on Mobile Object Systems: Operating System support for Mobile Object Systems, Jyvälskylä, Finland, June 1997.

[31] F. Kluegl: *Multiagentensimulation. Konzepte, Werkzeuge, Anwendung*. Addison-Wesley, 2001, ISBN: 3827317908.

[32] F. Knabe: *Performance-oriented implementation strategies for a mobile agent language*. In: J. Vitek, Ch. Tschudin (Ed.): Mobile Object Systems: Towards the Programmable Internet (MOS'96), Linz, 1996. Vol. 1222 of Lecture Notes in Computer Science, pp. 229-224. Springer-Verlag, Berlin, 1997.

[33] D. Kotz, G. Jiang, R. Gray, G. Cybenko, R. A. Peterson: *Performance Analysis of Mobile*

Agents for Filtering Data Streams on Wireless Networks. Dartmouth College, Department of Computer Science, Technical Report TR2000-366, New Hampshire 2000.

[34] A. Küpper, A. S. Park: *Stationary vs. Mobile User Agents in Future Mobile Telecommunication Networks*. In: Proceedings of 2nd International Workshop on Mobile Agents 98, Springer, LNCS 1477, Stuttgart 1998, pp. 112-123.

[35] D. B. Lange, M. Oshima: *Programming and Deploying Java Mobile Agents with Aglets*. Addison Wesley, 1998.

[36] A. M. Law, W. D. Kelton: *Simulation Modeling and Analysis*. McGraw-Hill Companies, Inc., Third Edition, 2000. ISBN 0-07-059292-6.

[37] E. D. Lazowska, J. Zahorjan, G. S. Graham, K. C. Sevcik: *Quantitive System Performance. Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984. ISBN 0-13-746975-6.

[38] P. L'Ecuyer: *Good Parameters and Implementations for Combined Multiple Recursive Random Number Generators*, Operations Research, 47, 1 (1999), 159 - 164.

[39] A. Lingnau, O. Drobnik: *Simulating Mobile Agent Systems with Swarm*. Proceedings 1st International Symposium on Agent Systems and Applications and 3rd International Symposium on Mobile Agents (ASA/MA'99), Palm Springs, CA. September 1999, pp. 272 - 273.

[40] O. Matthes: *Discrete Simulation mit Java - Eine Umgebung zur Durchführung und Visualisierung von Simulationsexperimenten*. In German language. Diploma thesis, University of Essen, 1999.

[41] D. Milojicic: *Mobile Agent Applications*. „Trend War"-Kolumne in IEEE Concurrency Magazine, Juli-September 1999.

[42] H. Mohamed.: *Discrete Event Simulation Using JavaDEMOS*. Technical Report, http://www.informatik.uni-essen.de/SysMod/publikationen/index.html, University of Essen 2001.

[43] B. Müller-Clostermann: *Strukturierte Modellierung, Konzeptheft*. Informatik IV, University of Dortmund, 1985.

[44] T. R. Norton: *Simalytic Hybrid Modeling: Planning the Capacity of Client/Server Applications*. 15th IMACS World Congress 1997 on Scientific Computation, Modelling and Applied Mathematics conference, Berlin, August 24-29, 1997.

[45] T. R. Norton: *The Simalytic Modeling Technique.* In R. Dumke (Ed.): Performance Engineering, State of the Art and Current Trends. 349 pp. Springer-Verlag Berlin/Heidelberg 2001, ISBN 3-540-42145-9.

[46] R. Pan, B. Prabhakar, K. Psounis, D. Wischik: *SHRINK: A Method for Scalable Performance Prediction and Efficient Network Simulation*. To appear in IEEE INFOCOM 2003.

[47] K. Pawlikowski, H.-D.J. Jeong and J.-S. Ruth Lee. *On Credibility of Simulation Studies Of Telecommunication Networks*. (Extended version). IEEE Communications Magazine, January 2002, pp. 132-139.

[48] R., J. Pooley: *Formalising the Description of Process Based Simulation Models*. PhD Thesis, University of Edinburgh, 1995.

[49] K. Psounis: *Probabilistic Algorithms for Web Caching and Performance Prediction of IP*

Networks and Web farms. Ph.D. thesis, Stanford, 2002.

[50] A. Puliafito, O. Tomarchio, L. Vita: *MAP: Design and Implementation of a Mobile Agent Platform*. Technical Report TR-CT-9712, University of Catania, 1997.

[51] H. D. Schwetman: *Hybrid Simulation Models of Computer Systems*. In: Communications of the ACM, Vol. 21, No. 9, 1978.

[52] *SeSAm* Homepage: http://ki.informatik.uni-wuerzburg.de/sesam/.

[53] K. L. Sevcik, A. I. Levy, S. K. Tripathi, J. L. Zahorjan: *Improving Approximations of Aggregated Queueing Network Systems*. In: K. M. Chandy, M. Reiser: Computer Performance. North-Holland, 1977, pp. 11-23. ISBN 0444 85038 4.

[54] L. M. Silva, G. Soares, P. Martins, V. Batista, L. Santos: *Comparing the Performance of Mobile Agent Systems*. Journal of Computer Communications, Special Issue on Mobile Agents for Telecommunications, March 2000.

[55] G. Soares, L. M. Silva: *Optimizing the Migration of Mobile Agents*. In: A. Karmouch, R. Impley (Ed.): First International Workshop on Mobile Agents for Telecommunication Applications (MATA'99), Ottawa (Canada), October 1999.

[56] M. Straßer, M. Schwehm: *A Performance Model for Mobile Agent Systems*. In H. R. Arabnia (Hrsg.): Proceedings der internationalen Konferenz über „Parallel and Distributed Processing Techniques and Applications“ PDPTA '97. Volume II, pp. 1132 - 1140.

[57] M. Strüwer: *Simulative Aggregierung von Rechensysetmmodellen: Konzepte, Techniken und Einsatz im Modellierungstool HIT*. In German language. Diploma thesis, University of Dortmund, 1989.

[58] M. Tilev: *A Decisive Agent Based Exchange Platform*. Technical Report No. 7, University of Duisburg-Essen, January 2005.

[59] G. Vigna: *Mobile Code Technologies, Paradigms, and Applications*. Dissertation, Politecnico die Milano, 1998.

[60] L. R. Welch, A. D. Stoyenko, T. J. Marlowe: *Response Time Prediction for Distributed Processes Specified in CaRT-Spec*. Control Eng. Practice, Vol. 3, No. 5, pp. 651-664, 1995.

[61] G. Wessels: *Digitale Sprachübertragung in paketvermittelnden Netzen*. Dissertation Universität Bremen, 1982.

[62] D. Wong, N. Paciorek, T. Walsh, J. DCelie, M. Young, B. Peet: *Concordia: An Infrastructure for Collaborating Mobile Agents*. In: Proceedings of the First International Workshop on Mobile Agents (MA '97), Volume 1219 of Lecture Notes in Computer Science, SS. 86-97, Springer-Verlag, 1997.

[63] B. Zeigler, H. Praehofer, T. G. Kim: Theory of Modeling and Simulation. Second edition. Academic Press, San Diego, 2000, ISBN 0-12-778455-1.

Appendix A Model Results

A.1 Finite Horizon Analysis with SHRiNK

As described in section 3.5 the SHRiNK approach is investigated to check model validity and efficiency gains provided by SHRiNK. The dynamic behaviour of the M/G/1 model is analysed within a finite horizon, i.e. the variation of residence times *R* along the simulation timeline of 13,000 time units. 24 sources are modelled in the detail model, which start with an arrival rate $\lambda_i = 0.5$ jobs per time unit, each. After 5000 time units the arrival rate is doubled to $\lambda_i = 1.0$ per source *i* until 8,000 time units. This results in a cumulative λ_{cum} of 24.0. Finally, the arrival rate is set back to $\lambda_i = 0.5$ until 13,000 simulated time units. The service rate is set to $\mu = 30.0$ jobs per time unit. Simulation starts with an empty system. The SHRiNK models get scale factors $\alpha = 0.167$ and $\alpha = 0.5$. Two scenarios are analysed: First, *c.o.v.[S]* per source is set to 4.0, next, *c.o.v.[S]* is set to 6.0.
Results are averaged over a moving time window of fixed size. For the following experiments a window size of 500 time units has proved suitable. Furthermore, multiple replications of the models are run using sequential simulation. The number of replications is determined by reaching the threshold for the relative statistical error.

To investigate the correctness of the simulation results the residence time and utilisation are analysed. Figure A-1 shows jobs' mean residence times and figure A-2 shows the utilisation during 13,000 simulated time units under varying workload. To compare the models, results are averaged over the replications. Summarising, results are averaged per simulation run over windows of 500 seconds and these mean values are again averaged over all replications.

Concerning the dynamic behaviour of the system, the SHRiNK models show the same significant reaction on the workload enhancement between 5000 and 8000 time units where the server utilisation is up to 80 %. Finally, the models react in the same way on the workload reduction. The dent in figure A-1 between 6000 and 8000 time units in all models is very obvious. The different model types do not use common random numbers, but, random numbers are quite similar: The SHRiNK models use a reduced number of arrival streams. The remaining ones are the same as in the detailed model.

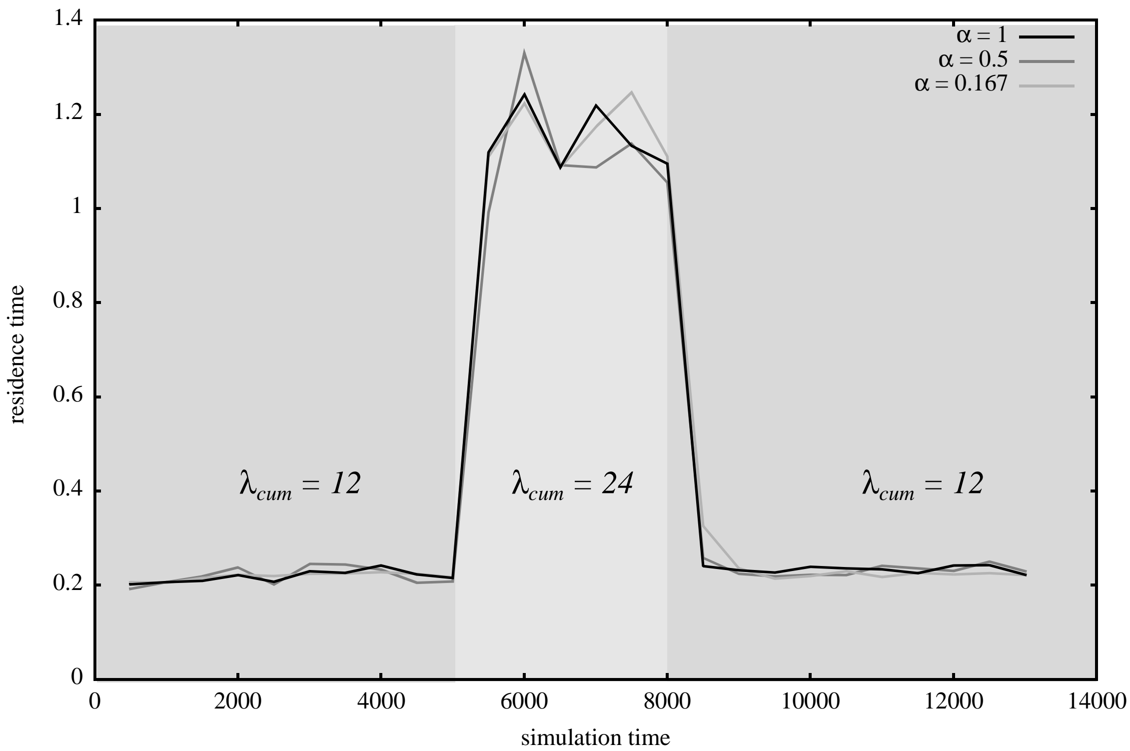

Figure A-1: **Mean residence times in M/G/1 model with c.o.v.[S] = 4.0**

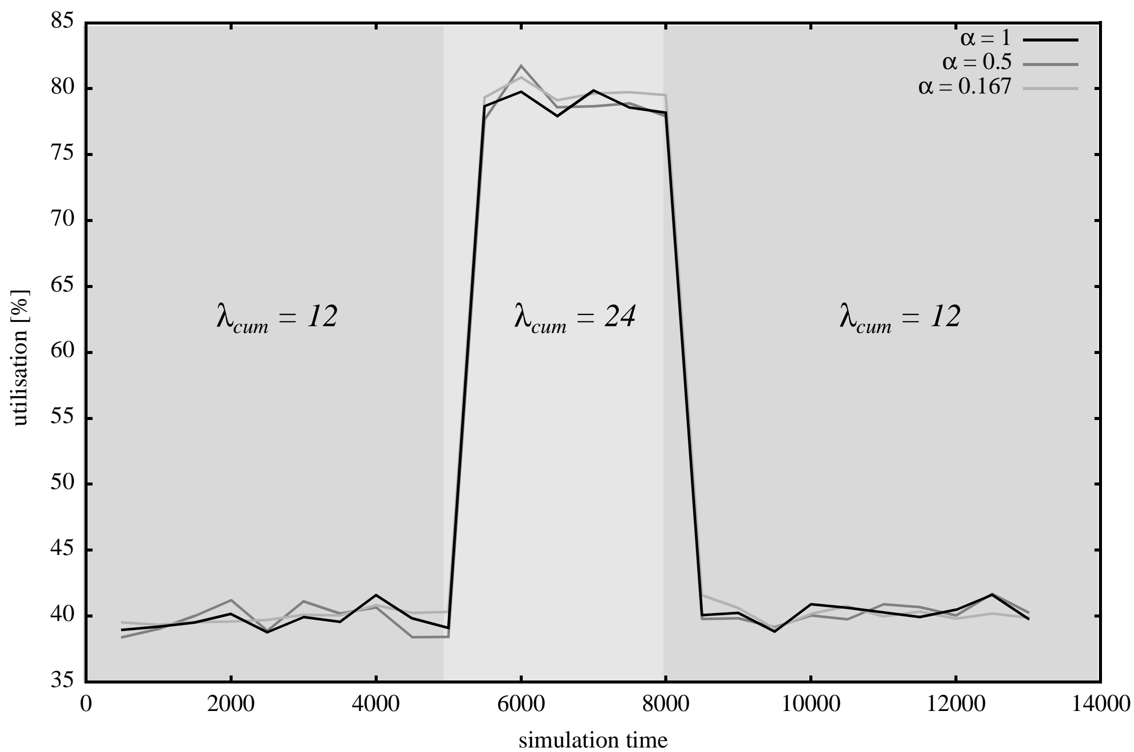

Figure A-2: **Utilisation in M/G/1 model with c.o.v.[S] = 4.0 (averaged over 500 time units)**

Furthermore, the random number streams which generate the service times provide the same (0,1) uniformly distributed random numbers in all model types. Cox-2 distributed values are achieved by inverse transformation. Thus, using the former described Cox-2 distribution, service times in all models differ by a deterministic addend ($\ln(\alpha)$). The c.o.v. is the same. These similar random numbers may explain the similar model behaviour.

To decide formally, whether results match satisfactorily again the *paired-t confidence interval* is calculated ([36] page 557 et seqq.). Differences of the mean values over the time windows of size 500 between the detailed model and the SHRiNK models are computed. A 95% confidence interval is calculated for these differences. The number of compared replications is determined by the minimum needed to achieve the threshold for the relative statistical error for the residence time, e.g. in the experiment with *c.o.v.[S]* = 4.0 70 replication runs are compared as only so many replications are available from the detailed model (see table A-3). The assumption is that SHRiNK models describe the same system as the detailed model if zero is included in the confidence interval, i.e. the "no-difference" value lies inside the interval.

	α = 0.5			α = 0.167		
model time [sec]	95 % CI of differences		% differences of mean values	95 % CI of differences		% differences of mean values
500	0.004	0.022	-4.809	-0.034	0.010	2.400
1000	-0.010	0.022	-0.119	0.004	0.034	-0.559
1500	-0.020	-0.002	4.462	-0.034	0.014	2.909
2000	-0.041	-0.008	7.448	-0.041	0.001	0.351
2500	-0.011	0.015	-2.822	-0.018	0.022	5.806
3000	-0.033	-0.002	6.819	-0.001	0.033	-2.129
3500	-0.044	-0.008	7.820	0.004	0.041	-0.588
4000	0.001	0.029	-3.672	-0.002	0.041	-6.108
4500	0.008	0.033	-7.966	0.012	0.052	0.606
5000	-0.002	0.026	-3.408	-0.014	0.026	0.839
5500	0.019	0.213	-11.502	-0.008	0.306	-0.865
6000	-0.179	0.073	7.037	-0.063	0.233	-1.518
6500	-0.133	0.078	0.460	-0.152	0.098	-0.070
7000	0.093	0.285	-10.819	0.145	0.381	-3.738
7500	-0.122	0.101	0.428	-0.098	0.239	9.978
8000	-0.009	0.135	-3.638	0.002	0.278	1.310
8500	-0.034	0.010	7.260	-0.151	-0.052	35.511
9000	-0.006	0.022	-3.091	-0.021	0.036	1.811
9500	-0.006	0.026	-3.685	-0.014	0.043	-5.600
10000	0.008	0.035	-7.054	0.001	0.037	-8.197
10500	0.007	0.033	-6.346	-0.004	0.039	-2.593
11000	-0.023	0.008	3.166	-0.007	0.032	-7.013
11500	-0.022	0.007	4.456	-0.033	0.011	0.303
12000	0.001	0.032	-4.645	-0.027	0.022	-7.849
12500	-0.028	0.001	2.980	-0.025	0.023	-7.100
13000	-0.018	0.012	3.526	-0.034	0.012	-0.437

Table A-1: **Differences of mean residence times of SHRiNK models compared to detailed M/G/1 model (c.o.v.[S] = 4.0) with 70 replication runs**

Table A-1 shows the confidence intervals of the differences of residence time between the SHRiNK models and the detailed models per time window. The shadowed fields indicate the confidence intervals which do not include zero. Because of the great many of these fields it can-

not be concluded that the SHRiNK models provide the same behaviour for the residence time as the detailed model, but, if one considers SHRiNK as an approximation for the detailed model results are satisfying. According to a "rule of thumb" the difference of mean residence times of an approximation and a detailed system should not exceed 30%. Table A-1 shows that this threshold is exceeded only once with $\alpha = 0.167$. Hence, for $\alpha = 0.5$ the SHRiNK model provides a good approximation of the residence time. Figure A-2 shows the similarities in utilisation.

The reasons for the differences between the mean residence times over the time windows of 500 time units result partly from the fact that SHRiNK generates less events than the detailed model (reduced by factor α) within the time window. With decreasing α, less jobs have finished within 500 time units. This means, the moving average is built from less jobs than in the detailed model, which can compound differences between the model results.

Figure A-3 and figure A-4 show the development of residence times and utilisation in a M/G/1 model scenario with c.o.v.[S] = 6.0. All models show the same dynamical behaviour, i.e. the same reaction on the workload variation. Again, there is the dent in residence time and utilisation between 6000 and 8000 time units. The similarity between the detailed M/G/1 model with c.o.v.[S] = 4.0 is due to the fact that both models use the same arrival streams. The similarity between the different models types of M/G/1 model with c.o.v.[S] = 6.0 can be explained, as before, by the similar random number streams which the SHRiNK models use.

Table A-2 shows the confidence intervals of the differences of residence time between the SHRiNK models and the detailed models per time window. The shadowed fields indicate the confidence intervals which do not include 0. With $\alpha = 0.5$ there are only three of those confidence intervals. Thus, one could conclude that for this scenario SHRiNK could model the same system. But, it should be noticed that confidence intervals are still quite large, i.e. they are not very precise after 110 replication runs.

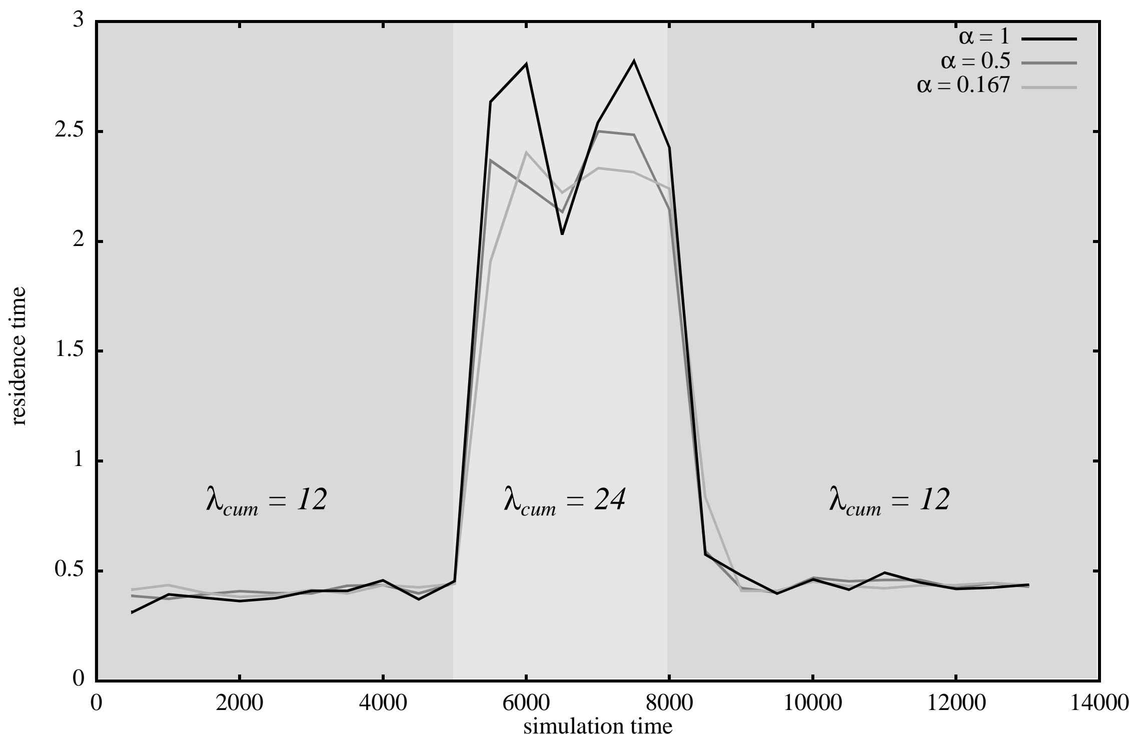

Figure A-3: **Mean residence times in M/G/1 model with c.o.v.[S] = 6.0**

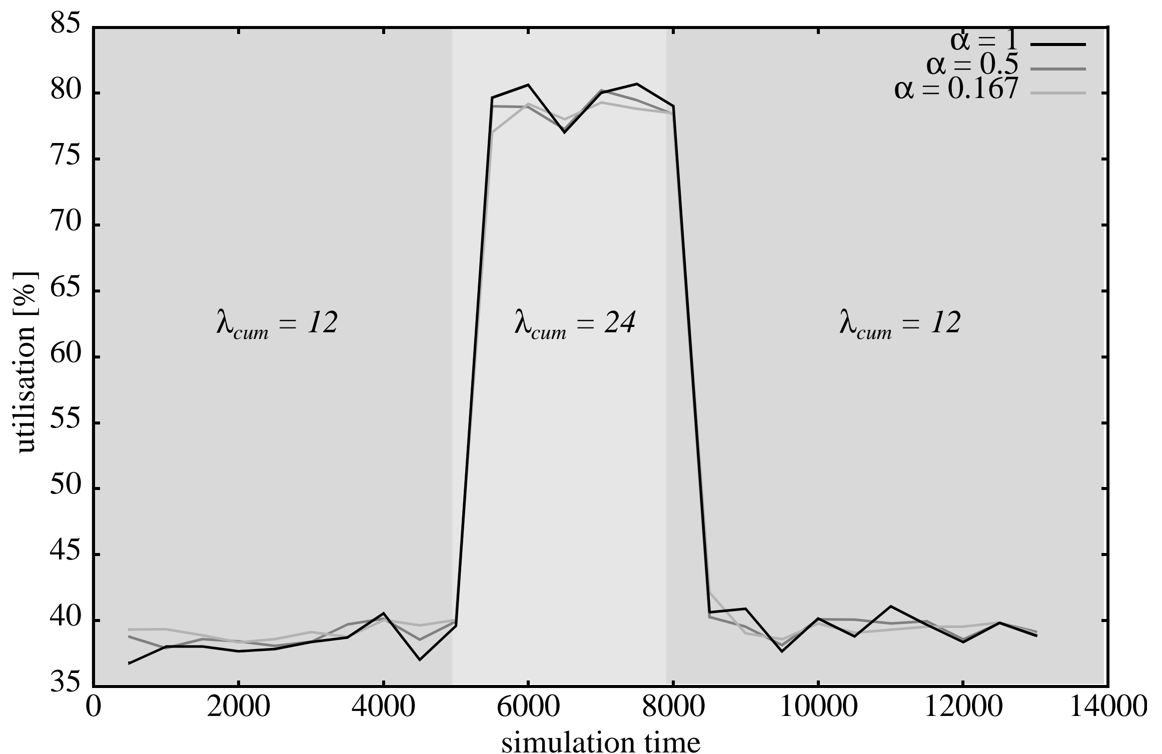

Figure A-4: **Utilisation in M/G/1 model with c.o.v.[S] = 6.0 (averaged over 500 time units)**

Summarising, it can be concluded that SHRiNK models can be used to approximate M/G/1 systems. But, the scale value α should not be too large. For the observed scenarios $\alpha = 0.5$ is recommended.

Table A-3 shows the efficiency gain[1] of the SHRiNK models. $\alpha = 1$ corresponds to the detailed model. Due to the fixed simulation period the efficiency gains of SHRiNK per replication run is significant. But, to gain results the required statistical quality it is necessary to run more replications for the SHRiNK models as for the detailed models[2]. Table A-3 shows the cumulative CPU time which was necessary to push the relative statistical error below a threshold of 0.1 for all time intervals[3], i.e. the cumulative CPU time is the time elapsed for all replication runs. While with *c.o.v.[S]* = 4.0 the efficiency advantage of SHRiNK is significant, with *c.o.v.[S]* = 6.0 there is hardly any efficiency gain with SHRiNK. On the contrary, with $\alpha = 0.5$ the performance of SHRiNK is even poorer.

Hence, SHRiNK is not generally applicable to increase efficiency. It depends on the simulated scenario. Moreover, the scale value α should not be too large. Otherwise approximation errors arise.

1. Experiments were executed using JavaDEMOS (based on Java jdk 1.4.1) on a PC with Intel 4 processor with 2.0 GHz, 512 MByte memory and with Windows XP operating system.
2. For practical reasons the number of replications necessary for the statistical significance was incremented by steps of width 10.
3. Because of the high performance effort with the SHRiNK M/G/1 models c.o.v.[S] = 6.0, the threshold of 0.1 was not completely reached for every time interval.

model time [sec]	α = 0.5 95 % CI of differences		α = 0.5 % differences of mean	α = 0.167 95 % CI of differences		α = 0.167 % differences of mean
500	-0.073	-0.008	22.742	-0.102	0.027	32.337
1000	-0.024	0.048	-4.929	-0.137	0.010	10.925
1500	-0.017	0.051	4.271	-0.034	0.076	6.034
2000	-0.045	0.033	12.619	-0.017	0.067	5.443
2500	-0.032	0.041	6.222	-0.032	0.073	3.932
3000	-0.016	0.059	-2.984	-0.043	0.075	0.768
3500	-0.044	0.027	5.873	-0.059	0.046	-2.799
4000	-0.012	0.078	-5.086	-0.049	0.100	-4.879
4500	-0.022	0.042	6.726	-0.097	0.026	14.126
5000	-0.066	0.026	-2.119	0.001	0.106	-2.892
5500	-0.176	0.606	-10.122	0.435	1.335	-27.528
6000	0.006	0.664	-19.750	-0.611	0.853	-14.320
6500	-0.494	0.041	5.065	-1.353	-0.019	9.443
7000	-0.080	0.467	-1.695	0.107	0.833	-8.241
7500	0.096	0.813	-11.945	0.471	1.192	-17.980
8000	-0.127	0.524	-11.635	-0.005	0.776	-7.631
8500	-0.158	0.061	2.617	-0.578	-0.071	45.058
9000	-0.042	0.057	-12.174	-0.052	0.110	-14.896
9500	-0.021	0.062	0.895	-0.034	0.085	3.004
10000	-0.028	0.051	1.575	-0.044	0.104	-2.418
10500	-0.040	0.041	9.006	-0.093	0.068	4.050
11000	-0.041	0.077	-6.987	0.008	0.134	-14.591
11500	-0.085	0.023	2.471	-0.114	0.054	-2.872
12000	-0.077	0.035	1.025	-0.099	0.042	4.273
12500	-0.074	0.019	4.788	-0.174	-0.011	5.055
13000	-0.044	0.042	-2.090	-0.182	0.035	-0.837

Table A-2: **Differences of mean residence times of SHRiNK models compared to a detailed M/G/1 model (c.o.v.[S] = 6.0) with 110 replication runs**

c.o.v. [S]	scale α	CPU time per run [sec] (mean value)	no of runs	cumulative CPU time [sec]	Efficiency gains of SHRiNK (decrease of cumulative CPU time)
4.000	1.000	102.716	70	7,190.143	
4.000	0.500	50.904	80	4,072.290	43.36%
4.000	0.167	16.880	280	4,726.460	34.26%
6.000	1.000	101.176	110	11,129.345	
6.000	0.500	50.638	240	12,153.205	-9.20%
6.000	0.167	17.099	640	10,943.378	1.67%

Table A-3: **Efficiency gains of SHRiNK vs. detailed M/G/1 model**

model time [sec]	95 % CI $\alpha = 1$		% rel. stat. error	95 % CI $\alpha = 0.5$		% rel. stat. error	95 % CI $\alpha = 0.167$		% rel. stat. error
500	0.193	0.210	4.063	0.183	0.201	4.759	0.195	0.217	5.204
1000	0.196	0.216	4.930	0.191	0.221	7.201	0.194	0.216	5.254
1500	0.199	0.218	4.558	0.209	0.228	4.485	0.203	0.227	5.392
2000	0.209	0.234	5.518	0.222	0.254	6.766	0.209	0.236	6.135
2500	0.196	0.218	5.151	0.189	0.213	6.047	0.206	0.232	5.901
3000	0.218	0.240	4.757	0.229	0.261	6.436	0.213	0.236	5.212
3500	0.214	0.238	5.337	0.227	0.260	6.913	0.211	0.238	6.139
4000	0.231	0.253	4.500	0.220	0.246	5.729	0.214	0.240	5.629
4500	0.210	0.235	5.531	0.193	0.217	6.027	0.211	0.237	5.647
5000	0.203	0.227	5.558	0.194	0.221	6.646	0.204	0.229	5.613
5500	1.035	1.204	7.554	0.921	1.061	7.016	1.017	1.203	8.389
6000	1.158	1.326	6.784	1.207	1.452	9.203	1.138	1.308	6.947
6500	0.991	1.183	8.818	1.005	1.179	7.936	1.008	1.164	7.192
7000	1.135	1.303	6.869	0.997	1.178	8.341	1.074	1.273	8.449
7500	1.037	1.229	8.463	1.033	1.242	9.182	1.123	1.369	9.832
8000	1.028	1.162	6.141	0.969	1.141	8.132	1.028	1.190	7.298
8500	0.226	0.255	5.929	0.234	0.282	9.272	0.295	0.357	9.378
9000	0.219	0.244	5.425	0.211	0.237	5.806	0.222	0.249	5.794
9500	0.214	0.240	5.762	0.203	0.234	7.184	0.201	0.227	6.010
10000	0.227	0.251	4.995	0.209	0.236	5.958	0.208	0.231	5.134
10500	0.222	0.249	5.718	0.208	0.234	5.741	0.217	0.242	5.434
11000	0.221	0.246	5.361	0.225	0.257	6.527	0.206	0.229	5.374
11500	0.214	0.236	4.769	0.221	0.250	6.187	0.214	0.238	5.291
12000	0.228	0.255	5.709	0.216	0.245	6.382	0.209	0.236	5.931
12500	0.231	0.254	4.842	0.234	0.265	6.373	0.212	0.238	5.740
13000	0.211	0.233	4.927	0.212	0.247	7.643	0.209	0.233	5.308

Table A-4: **Confidence intervals (CI) of mean residence time of M/G/1 models (c.o.v.[S] = 4.0). 70 replication runs with $\alpha = 1$, 80 replication runs with $\alpha = 0.5$, 280 replication runs with $\alpha = 0.167$**

model time [sec]	95 % CI α = 1		% rel. stat. error	95 % CI α = 0.5		% rel. stat. error	95 % CI α = 0.167		% rel. stat. error
500	0.293	0.324	4.979	0.351	0.388	4.999	0.321	0.371	7.293
1000	0.379	0.418	4.941	0.372	0.411	4.992	0.427	0.483	6.176
1500	0.357	0.396	5.208	0.348	0.384	4.924	0.356	0.396	5.296
2000	0.337	0.377	5.563	0.357	0.398	5.436	0.329	0.357	4.107
2500	0.350	0.394	5.864	0.343	0.383	5.557	0.336	0.371	4.916
3000	0.385	0.428	5.270	0.368	0.406	4.845	0.372	0.415	5.410
3500	0.386	0.429	5.200	0.397	0.438	4.984	0.410	0.451	4.758
4000	0.422	0.479	6.321	0.416	0.462	5.287	0.424	0.479	5.982
4500	0.340	0.384	6.032	0.354	0.388	4.669	0.386	0.433	5.696
5000	0.437	0.491	5.816	0.473	0.525	5.250	0.383	0.417	4.312
5500	2.478	2.921	8.204	2.248	2.664	8.458	1.679	1.965	7.843
6000	2.667	3.175	8.692	2.382	2.756	7.295	2.509	3.056	9.830
6500	1.916	2.250	8.015	2.081	2.392	6.967	2.555	3.090	9.484
7000	2.369	2.753	7.494	2.149	2.404	5.606	1.932	2.190	6.248
7500	2.596	3.038	7.846	2.202	2.583	7.961	1.830	2.078	6.347
8000	2.189	2.652	9.570	2.111	2.415	6.712	1.939	2.201	6.319
8500	0.530	0.634	8.912	0.567	0.699	10.452	0.801	1.008	11.436
9000	0.448	0.506	6.090	0.455	0.504	5.164	0.425	0.481	6.134
9500	0.373	0.426	6.623	0.369	0.410	5.226	0.366	0.409	5.549
10000	0.439	0.494	5.898	0.429	0.470	4.528	0.409	0.459	5.802
10500	0.393	0.441	5.756	0.393	0.438	5.344	0.404	0.470	7.611
11000	0.469	0.531	6.224	0.470	0.529	5.908	0.399	0.439	4.782
11500	0.417	0.482	7.282	0.473	0.524	5.160	0.444	0.503	6.243
12000	0.391	0.440	5.817	0.422	0.487	7.158	0.425	0.480	6.079
12500	0.397	0.446	5.764	0.434	0.488	5.834	0.507	0.574	6.121
13000	0.409	0.469	6.916	0.426	0.475	5.412	0.486	0.567	7.627

Table A-5: **Confidence intervals (CI) of mean residence time of M/G/1 model (c.o.v.[S] = 6.0). 110 replication runs with α = 1, 240 runs with α = 0.5, 640 runs with α = 0.167**

A.2 CFS - Steady State Analysis

Station types	λ [ma/sec]	Detailed model util	CFS util	Detailed model 90% CI RTT [sec]	CFS 90% CI RTT [sec]	90% CI of differences
$G/D_4/1$, c.o.v.[A] = 3.0	0.05	14.9%	15.0%	[70.479, 70.760]	[70.476, 70.755]	[-0.009, 0.001]
	0.20	59.8%	59.8%	[145.516, 155.066]	[145.515, 155.407]	[-0.223, 0.564]
	0.27	80.7%	80.7%	[264.235, 306.123]	[266.441, 308.919]	[1.387, 3.615]
$M/H_4/1$	0.05	15.0%	15.0%	[72.140, 72.680]	[72.140, 72.680]	[0.000, 0.000]
	0.20	60.2%	60.2%	[170.454, 175.201]	[170.454, 175.201]	[0.000, 0.000]
	0.27	80.3%	80.3%	[369.215, 390.498]	[369.215, 390.498]	[0.000, 0.000]
G/G/1,	0.05	14.8%	15.0%	[303.896, 331.969]	[302.665, 330.961]	[-4.913, 2.673]
c.o.v.[A]= 3.0,	0.20	60.8%	60.7%	[1832.054, 2075.205]	[1810.408, 2177.041]	[-49.887, 130.079]
c.o.v.[S_i] = 5.0	0.27	78.8%	79.9%	[4483.073, 5973.983]	[4611.951, 5977.817]	[-202.128, 334.840]

Table A-6: **Comparison of steady state results of CFS and detailed model**

Station types on PC	λ [ma/sec]	CPU time detailed model [min.]	CPU time CFS [min.]	Gains by CFS
$G/D_4/1$, c.o.v.[A] = 3.0	0.05	38.044	29.003	23.8%
on *blofeld*	0.20	43.280	33.230	23.2%
	0.27	52.237	40.929	21.6%
$M/H_4/1$	0.05	19.343	14.324	25.9%
on *goldfinger*	0.20	23.221	17.458	24.8%
	0.27	29.847	23.236	22.1%
G/G/1, c.o.v.[A] = 3.0,	0.05	14.434	11.479	20.5%
c.o.v.[S_i] = 5.0	0.20	48.737	43.981	9.8%
on *trinity*	0.27	169.617	162.999	3.9%

Table A-7: **Comparison of efficiency of CFS and detailed model**

A.3 CFS - Finite Horizon Analysis of System with M/H_4/1 Agent Servers

System description see section 4.4.3.

model time [sec]	90 % CI Detail model		% rel. stat. error	90 % CI CFS		% rel. stat. error
8000	70.393	72.653	1.580	70.393	72.653	1.580
12000	71.178	73.287	1.460	71.178	73.287	1.460
16000	70.303	72.333	1.423	70.303	72.333	1.423
20000	72.448	74.295	1.259	72.448	74.295	1.259
24000	443.881	467.398	2.581	443.881	467.398	2.581
28000	614.890	673.824	4.573	614.890	673.824	4.573
32000	680.229	763.850	5.791	680.229	763.850	5.791
36000	685.658	759.084	5.082	685.658	759.084	5.082
40000	631.734	708.812	5.750	631.734	708.812	5.750
44000	633.999	744.512	8.017	633.999	744.512	8.017
48000	618.968	703.962	6.425	618.968	703.962	6.425
52000	568.485	647.371	6.488	568.485	647.371	6.488
56000	610.399	697.407	6.653	610.399	697.407	6.653
60000	569.018	704.506	10.639	569.018	704.506	10.639
64000	300.373	405.879	14.939	300.373	405.879	14.939
68000	71.840	73.864	1.389	71.840	73.864	1.389
72000	70.571	72.326	1.228	70.571	72.326	1.228
76000	69.972	71.187	0.860	69.972	71.187	0.860

Table A-8: **Confidence intervals (CI) of mean residence time (10 replication runs)**

model time [sec]	90 % CI of differences	
8000	0.000	0.000
12000	0.000	0.000
16000	0.000	0.000
20000	0.000	0.000
24000	0.000	0.000
28000	0.000	0.000
32000	0.000	0.000
36000	0.000	0.000
40000	0.000	0.000
44000	0.000	0.000
48000	0.000	0.000
52000	0.000	0.000
56000	0.000	0.000
60000	0.000	0.000
64000	0.000	0.000
68000	0.000	0.000
72000	0.000	0.000
76000	0.000	0.000

Table A-9: **Differences of mean RTT of CFS model compared to detailed model (10 replication runs)**

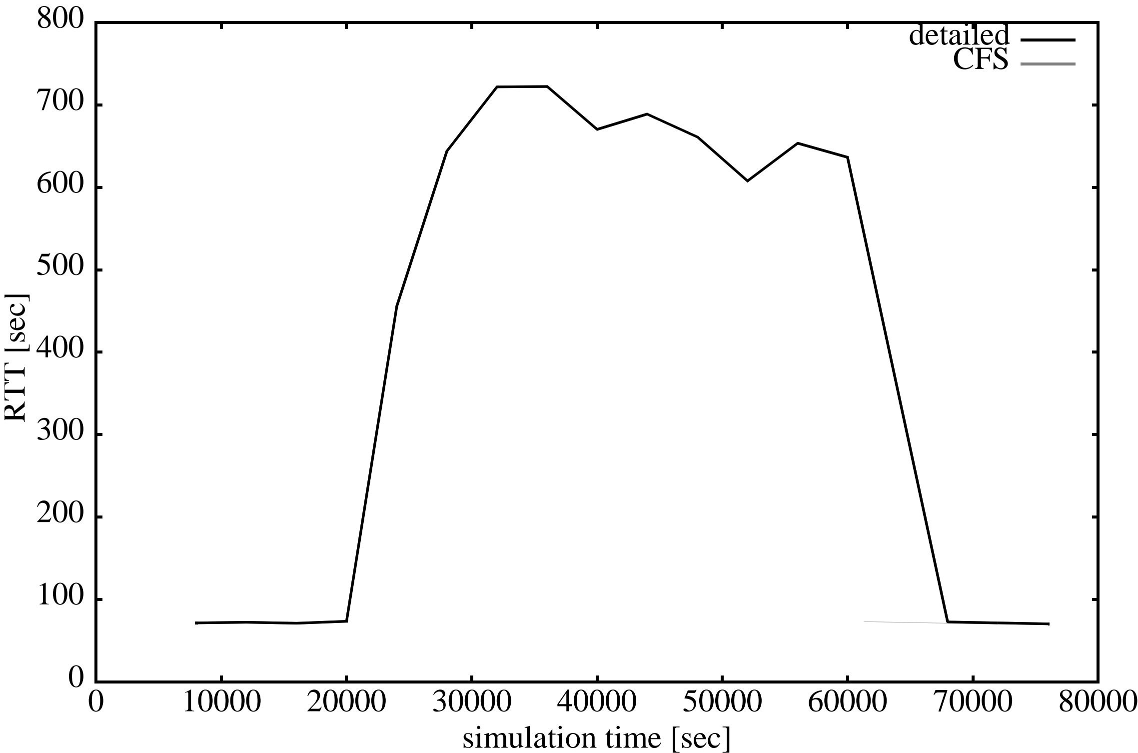

Figure A-5: **RTTs in an M/H_4/1 network (averages over 10 replications)**

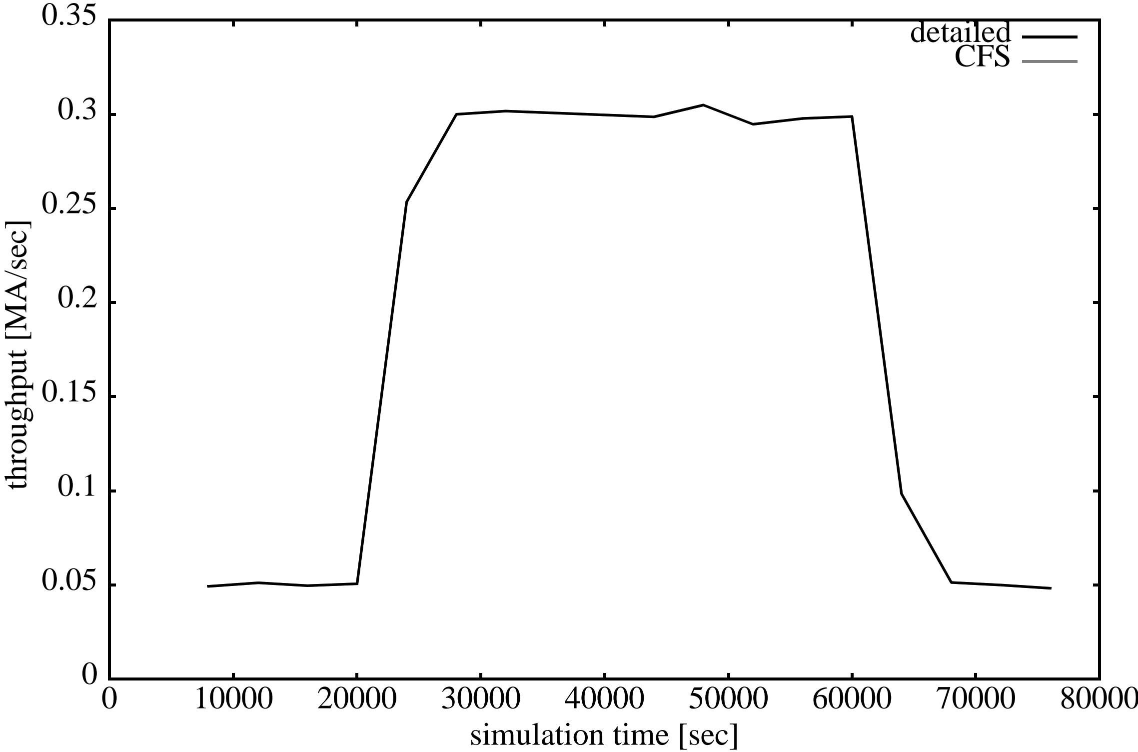

Figure A-6: **Throughput in an M/H_4/1 network (averages over 10 replications)**

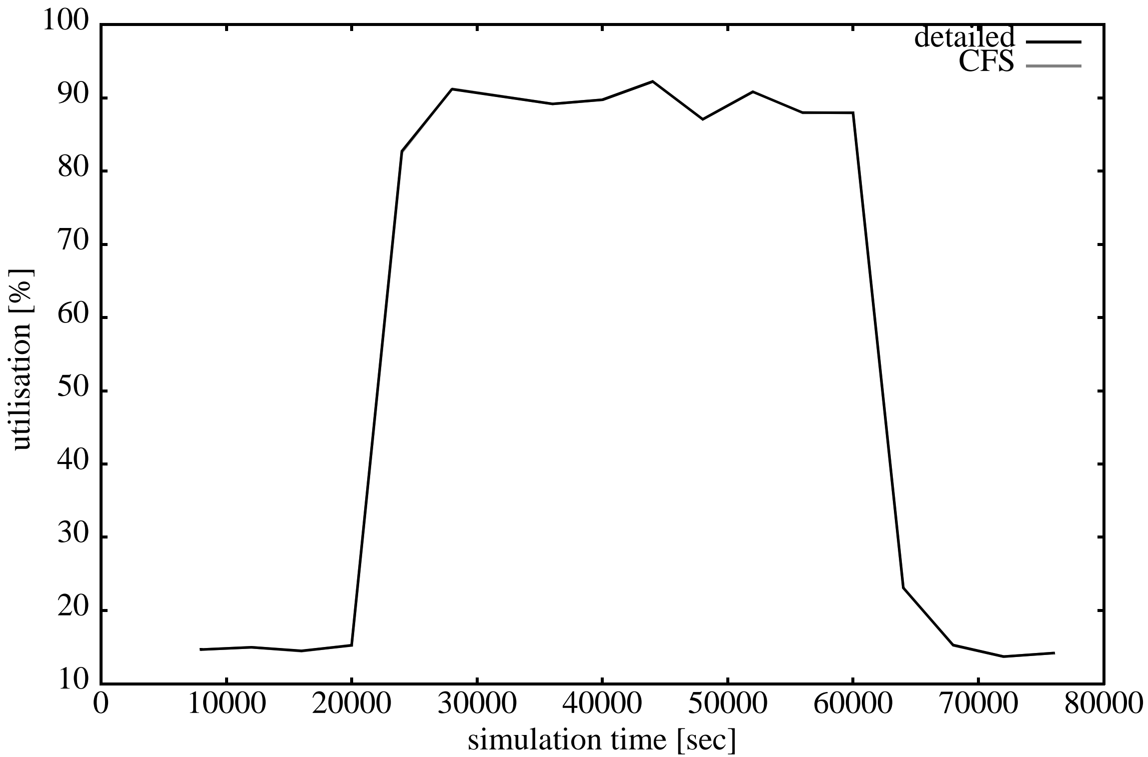

Figure A-7: **Utilisation of server "Bond" in a 10 Mbit network with mobile agents of 150 KB (averages over 10 replications)**

A.4 CFS - Finite Horizon Analysis of System with $G/D_4/1$ Agent Servers

System description see section 4.4.3.

model time [sec]	90 % CI Detail model		% rel. stat. error	90 % CI CFS		% rel. stat. error
8000	70.264	70.686	0.300	70.274	70.700	0.302
12000	69.757	70.260	0.360	69.764	70.272	0.363
16000	70.336	70.864	0.374	70.339	70.866	0.373
20000	70.255	70.768	0.363	70.242	70.750	0.360
24000	363.793	402.570	5.060	363.892	402.314	5.015
28000	430.777	499.502	7.388	430.514	498.861	7.354
32000	405.252	456.342	5.930	404.692	455.725	5.931
36000	421.544	484.073	6.905	420.621	483.131	6.917
40000	401.088	474.725	8.408	401.459	475.553	8.448
44000	424.376	492.614	7.441	423.447	491.705	7.459
48000	398.280	468.257	8.075	398.494	467.577	7.977
52000	425.458	492.951	7.349	425.431	492.231	7.279
56000	455.728	537.085	8.195	455.352	535.998	8.135
60000	364.547	431.945	8.462	366.154	432.977	8.362
64000	171.718	222.818	12.952	171.957	223.621	13.060
68000	70.473	71.027	0.391	70.486	71.044	0.394
72000	70.387	70.941	0.392	70.390	70.949	0.396
76000	70.050	70.626	0.410	70.048	70.619	0.405

Table A-10: **Confidence intervals (CI) of mean residence time (100 replication runs)**

model time [sec]	90 % CI of differences		% differences of mean values
8000	0.002	0.022	0.017
12000	-0.004	0.023	0.014
16000	-0.010	0.016	0.004
20000	-0.034	0.002	-0.022
24000	-1.099	0.942	-0.021
28000	-1.912	1.008	-0.097
32000	-2.030	0.853	-0.137
36000	-2.288	0.422	-0.206
40000	-0.871	2.071	0.137
44000	-2.625	0.787	-0.200
48000	-1.771	1.305	-0.054
52000	-1.958	1.211	-0.081
56000	-2.236	0.773	-0.147
60000	-0.027	2.666	0.331
64000	-0.639	1.681	0.264
68000	-0.002	0.032	0.021
72000	-0.009	0.021	0.008
76000	-0.024	0.015	-0.007

Table A-11: **Differences of mean RTT of CFS model compared to detailed model (100 runs)**

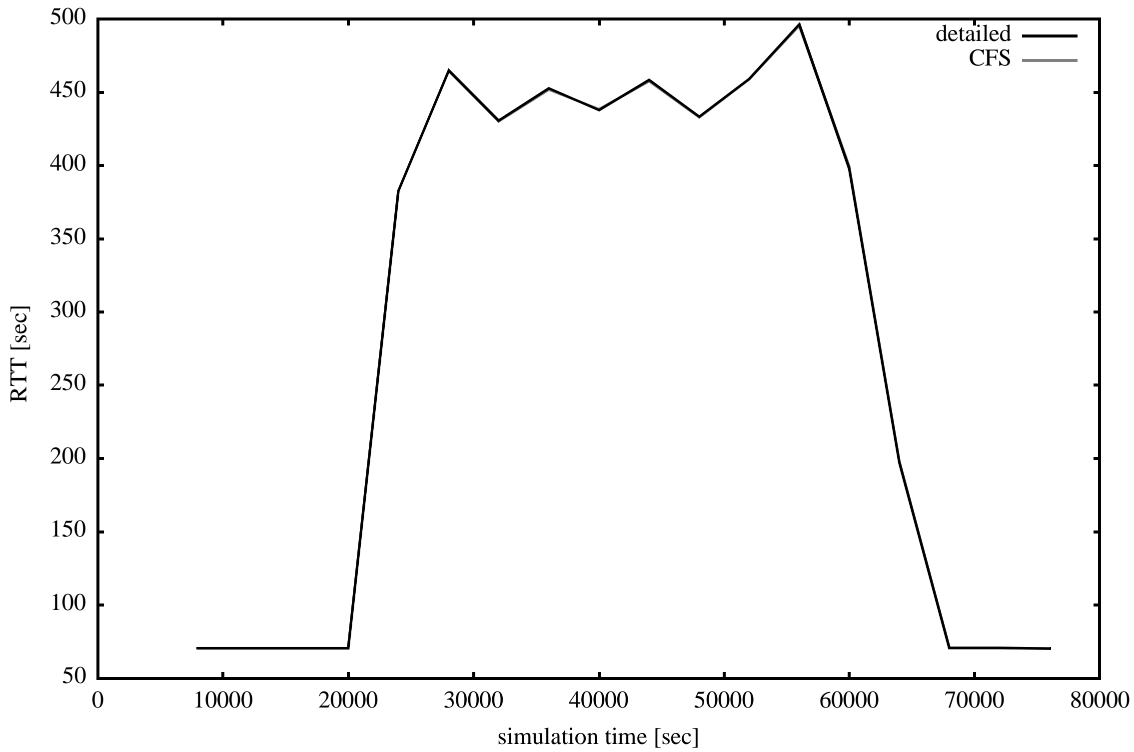

Figure A-8: **RTTs in an $G/D_4/1$ network (averages over 100 replications)**

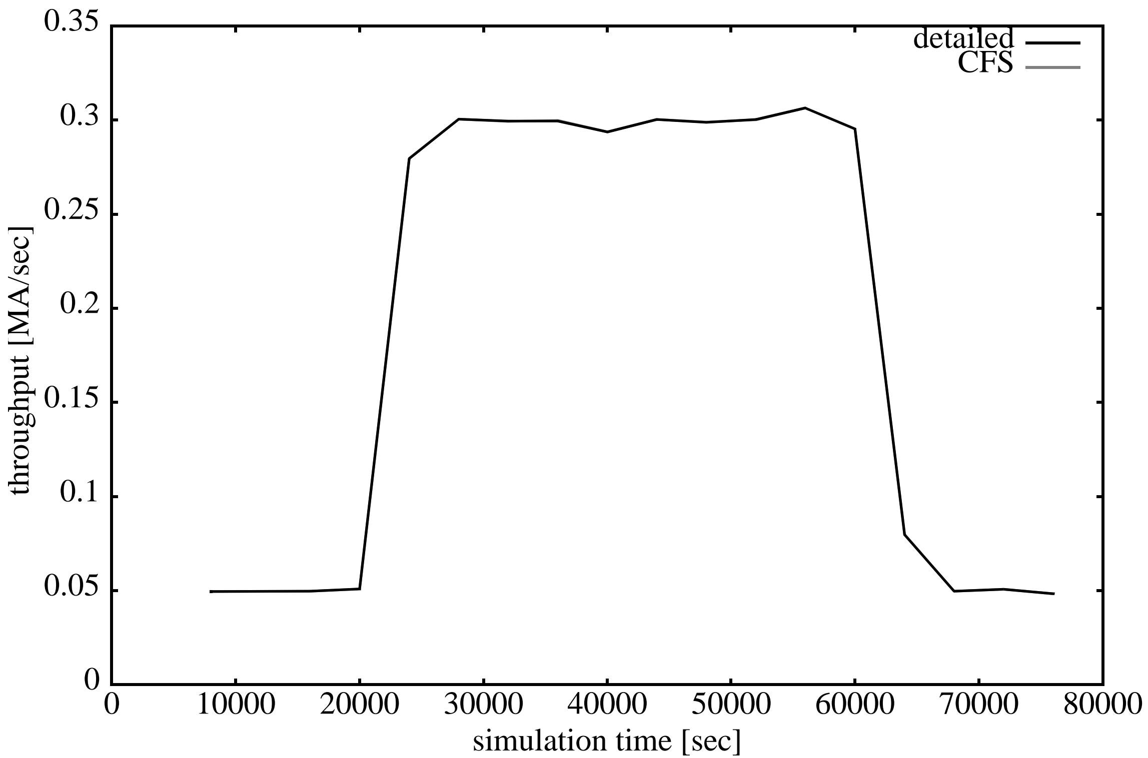

Figure A-9: **Throughput in an $G/D_4/1$ network (averages over 100 replications)**

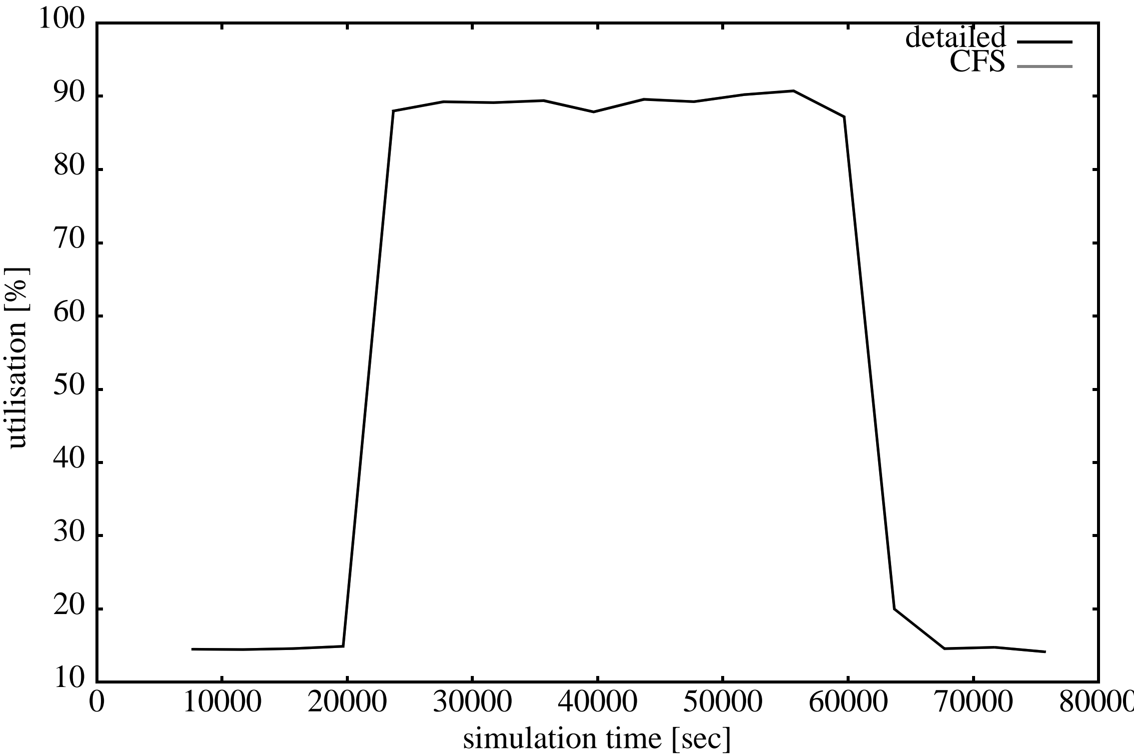

Figure A-10: **Utilisation of server "Bond" in a 10 Mbit network with mobile agents of 150 KB (averages over 100 replications)**

A.5 CFS - Finite Horizon Analysis of System with G/G/1 Agent Servers

model time [sec]	90 % CI Detail model		% rel. stat. error	90 % CI CFS		% rel. stat. error
8000	281.190	308.650	4.655	280.774	307.665	4.570
12000	275.186	303.492	4.891	282.736	313.663	5.186
16000	269.701	298.069	4.996	272.106	302.183	5.237
20000	244.811	275.334	5.868	243.620	273.075	5.701
24000	1408.953	1475.883	2.320	1400.938	1469.328	2.383
28000	2820.814	2954.856	2.321	2815.358	2949.836	2.333
32000	3770.924	3958.839	2.431	3769.986	3959.572	2.453
36000	4367.875	4584.459	2.419	4381.744	4592.644	2.350
40000	4869.408	5113.462	2.445	4896.467	5133.944	2.368
44000	5417.763	5685.813	2.414	5435.546	5701.183	2.385
48000	6118.571	6385.999	2.139	6111.520	6375.379	2.113
52000	6609.191	6862.362	1.879	6607.837	6855.509	1.840
56000	6927.189	7197.642	1.915	6922.682	7185.596	1.864
60000	7342.259	7631.971	1.935	7334.275	7615.140	1.879
64000	7337.386	7674.352	2.245	7341.972	7676.816	2.230
68000	5358.739	5800.047	3.955	5367.343	5820.070	4.047
72000	381.391	677.099	27.937	401.289	720.892	28.481
76000	296.684	328.898	5.150	288.743	320.587	5.226

Table A-12: **Confidence intervals (CI) of mean residence time (100 replication runs)**

A.6 CFS - Finite Horizon Analysis - Efficiency Gains

Station types / PC	CPU time detailed model [min.]	CPU time CFS [min.]	Efficiency gains of CFS (decrease of CPU time)
$G/D_4/1$, c.o.v.[A_i] = 3.0 on *blofeld*	15.092	11.811	21.7%
$M/H_4/1$ on *goldfinger*	9.283	7.282	21.6%
G/G/1, c.o.v.[A_i] = 3.0, c.o.v.[S_i] = 5.0 on *trinity*	43.389	41.126	5.2%

Table A-13: **Amount of CPU time per run**

A.7 CRRS - Steady State Analysis

Station types	λ [ma/sec]	Detailed model util	CRRS util	Detailed model 90% CI RTT [sec]	CRRS 90% CI RTT [sec]	90% CI of differences
$G/D_4/1$,	0.05	14.9%	15.0%	[73.450, 73.842]	[73.589, 73.964]	[0.093, 0.168]
c.o.v.[A] = 3.0	0.20	59.8%	59.8%	[140.593, 148.497]	[141.609, 150.04]	[0.798, 1.762]
	0.27	80.7%	80.7%	[228.465, 259.650]	[230.934, 263.378]	[1.848, 4.347]
$M/H_4/1$	0.05	15.0%	15.0%	[68.534, 68.918]	[68.681, 68.996]	[-0.004, 0.229]
	0.20	60.1%	60.2%	[139.499, 143.911]	[141.918, 145.642]	[1.189, 2.962]
	0.27	80.8%	80.9%	[270.685, 284.050]	[273.61, 289.662]	[1.349, 7.189]
G/G/1,	0.05	15.0%	15.3%	[137.079, 142.569]	[132.537, 142.033]	[-6.619, 1.542]
c.o.v.[A] = 3.0,	0.20	60.6%	60.1%	[857.766, 1019.822]	[848.530, 973.256]	[-86.528, 30.726]
c.o.v.[S_i] = 5.0	0.27	82.7%	79.8%	[2867.265, 3227.301]	[2483.47, 3250.030]	[-472.526, 111.475]

Table A-14: **Confidence intervals of round trip time**

Station types / PC	l [ma/sec]	CPU time detailed model [min.]	CPU time CRRS [min.]	Gains by CRRS
$G/D_4/1$, c.o.v.[A] = 3.0	0.05	83.203	8.094	90.3%
on *goldeneye*	0.20	96.897	10.151	89.5%
	0.27	108.108	12.775	88.2%
$M/H_4/1$	0.05	81.843	8.719	89.3%
on *goldeneye*	0.20	96.917	10.926	88.7%
	0.27	112.637	14.098	87.5%
G/G/1, c.o.v.[A] = 3.0,	0.05	84.981	9.354	89.0%
c.o.v.[S_i] = 5.0	0.20	142.528	22.342	84.3%
on *goldeneye*	0.27	236.543	65.266	72.4%

Table A-15: **Amount of CPU time**

A.8 CRRS - Finite Horizon Analysis of System with M/H_4/1 Agent Servers

System description see section 4.6.2.

model time [sec]	90 % CI Detail model		% rel. stat. error	90 % CI CRRS		% rel. stat. error
8000	67.953	69.277	0.965	68.198	69.377	0.857
12000	67.709	68.779	0.784	67.615	68.706	0.800
16000	67.172	68.298	0.832	67.321	68.587	0.931
20000	68.896	70.255	0.976	68.853	70.193	0.963
24000	298.082	310.178	1.989	299.039	314.621	2.539
28000	423.829	461.051	4.206	427.182	466.288	4.377
32000	448.079	509.529	6.417	454.328	516.617	6.415
36000	502.485	538.074	3.420	511.274	543.658	3.070
40000	470.210	527.305	5.724	484.727	539.546	5.352
44000	479.134	541.111	6.075	495.493	555.145	5.678
48000	483.654	535.117	5.051	495.051	547.612	5.041
52000	437.142	487.439	5.440	448.500	502.521	5.680
56000	450.266	504.961	5.726	461.571	519.726	5.926
60000	446.888	511.956	6.786	461.029	525.077	6.495
64000	298.532	383.365	12.441	305.525	397.544	13.088
68000	68.519	69.709	0.860	68.446	69.628	0.856
72000	67.838	69.316	1.077	67.893	69.427	1.117
76000	66.672	67.697	0.763	66.750	67.750	0.743

Table A-16: **Confidence intervals (CI) of mean residence time (15 replication runs)**

model time [sec]	90 % CI of differences		% differences of mean values
8000	-0.099	0.445	0.252
12000	-0.246	0.078	-0.123
16000	0.023	0.416	0.324
20000	-0.244	0.139	-0.076
24000	-0.296	5.696	0.888
28000	0.304	8.285	0.971
32000	0.367	12.970	1.393
36000	0.201	14.172	1.381
40000	8.445	18.312	2.682
44000	11.752	18.641	2.979
48000	7.842	16.051	2.345
52000	5.898	20.542	2.860
56000	6.919	19.151	2.729
60000	9.430	17.832	2.843
64000	4.538	16.634	3.105
68000	-0.204	0.050	-0.111
72000	-0.022	0.189	0.122
76000	-0.050	0.181	0.097

Table A-17: **Differences of mean RTT of CRRS model compared to detailed model (15 replication runs)**

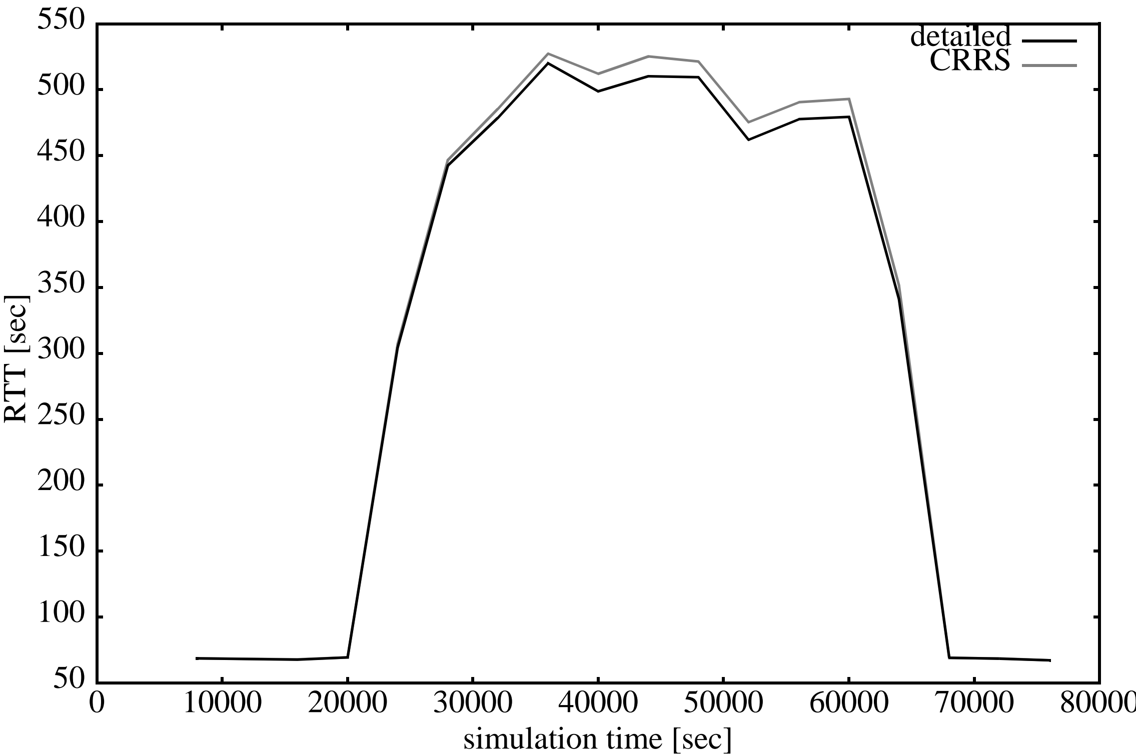

Figure A-11: **RTTs in an M/H_4/1 network (averages over 15 replications)**

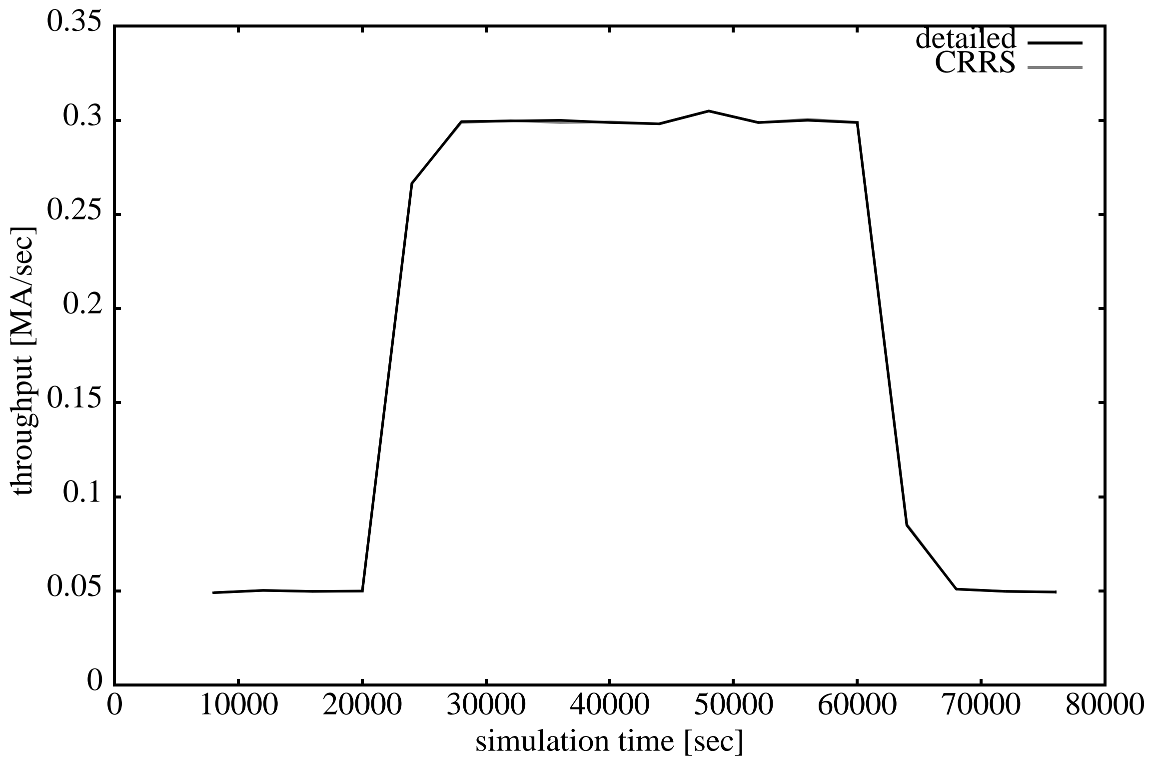

Figure A-12: **Throughput in an $M/H_4/1$ network (averages over 15 replications)**

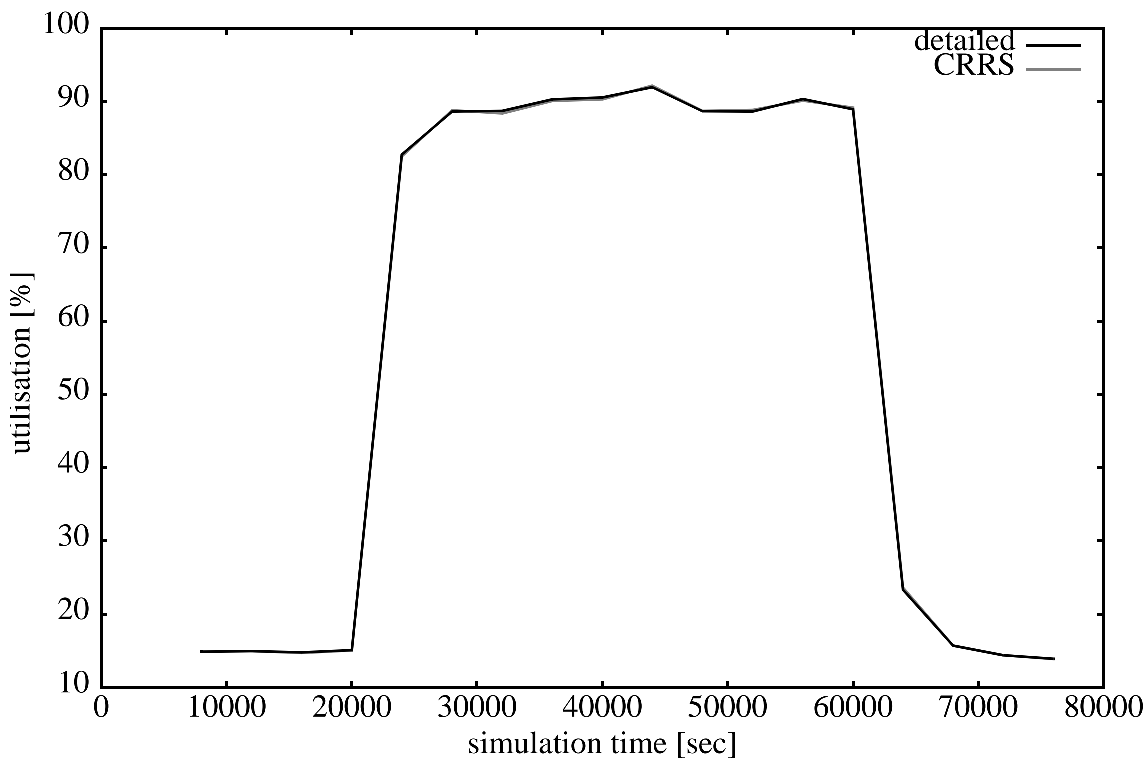

Figure A-13: **Utilisation of server "Bond" in a 10 Mbit network with mobile agents of 150 KB (averages over 15 replications)**

A.9 CRRS - Finite Horizon Analysis of System with $G/D_4/1$ Agent Servers

System description see section 4.6.2.

model time [sec]	90 % CI Detail model		% rel. stat. error	90 % CI CRRS		% rel. stat. error
8000	73.265	73.819	0.377	73.345	73.891	0.371
12000	72.710	73.356	0.442	72.808	73.432	0.427
16000	73.342	73.998	0.446	73.339	73.997	0.446
20000	73.200	73.850	0.442	73.191	73.849	0.447
24000	282.460	305.568	3.930	285.532	309.052	3.956
28000	351.262	398.222	6.266	357.627	405.514	6.275
32000	333.402	372.229	5.502	340.284	379.973	5.510
36000	343.251	388.659	6.204	350.095	396.718	6.243
40000	329.412	380.348	7.177	336.471	388.853	7.222
44000	346.411	398.230	6.959	353.231	406.136	6.967
48000	329.386	378.417	6.927	335.856	386.400	6.998
52000	342.960	390.064	6.426	349.625	398.175	6.492
56000	375.469	432.127	7.016	383.685	442.559	7.126
60000	312.544	360.185	7.082	319.159	368.762	7.211
64000	180.008	239.111	14.102	184.211	246.083	14.379
68000	73.450	74.106	0.445	73.557	74.219	0.448
72000	73.528	74.217	0.466	73.633	74.331	0.472
76000	73.058	73.744	0.467	73.096	73.801	0.480

Table A-18: **Confidence intervals (CI) of mean residence time (100 replication runs)**

model time [sec]	90 % CI of differences		% differences of mean values
8000	-0.006	0.157	0.103
12000	0.009	0.165	0.119
16000	-0.091	0.087	-0.003
20000	-0.087	0.077	-0.007
24000	2.568	3.988	1.115
28000	5.687	7.970	1.822
32000	6.206	8.419	2.073
36000	6.270	8.631	2.036
40000	6.283	9.281	2.193
44000	6.047	8.679	1.978
48000	5.837	8.616	2.042
52000	6.143	8.632	2.016
56000	7.441	11.206	2.309
60000	6.112	9.080	2.258
64000	3.590	7.585	2.666
68000	0.027	0.193	0.149
72000	0.015	0.204	0.148
76000	-0.035	0.130	0.065

Table A-19: **Differences of mean RTT of CRRS compared to detailed model (100 runs)**

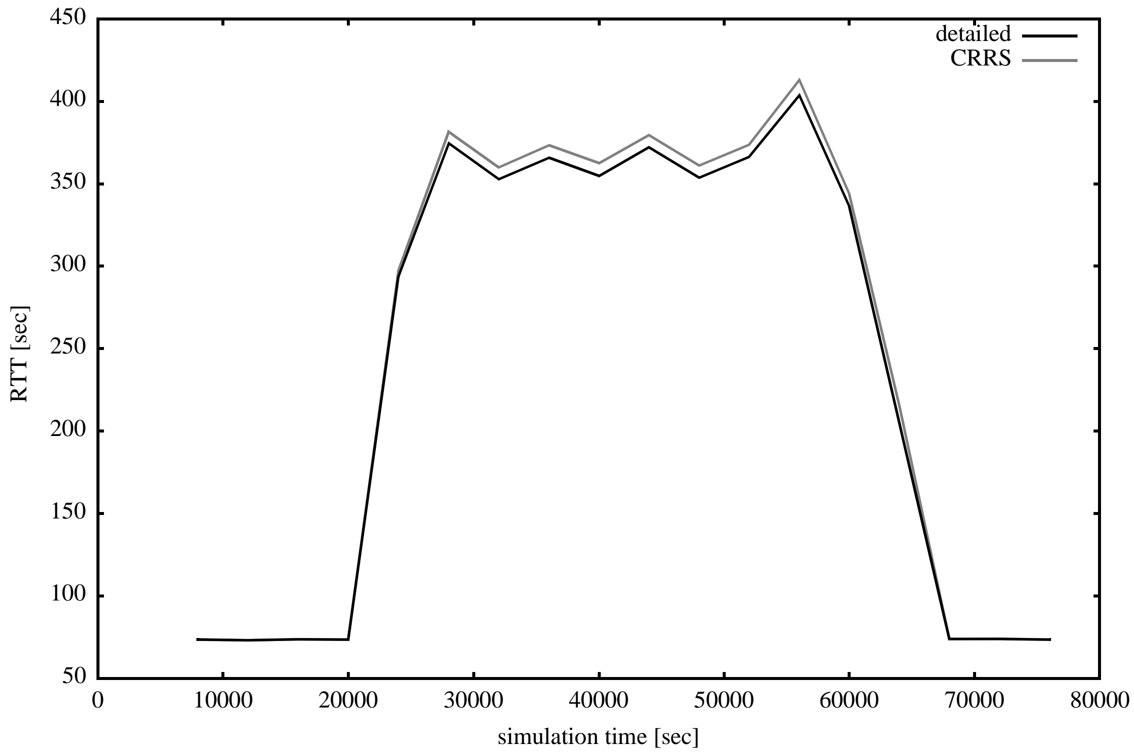

Figure A-14: **RTTs in an G/D_4/1 network (averages over 100 replications)**

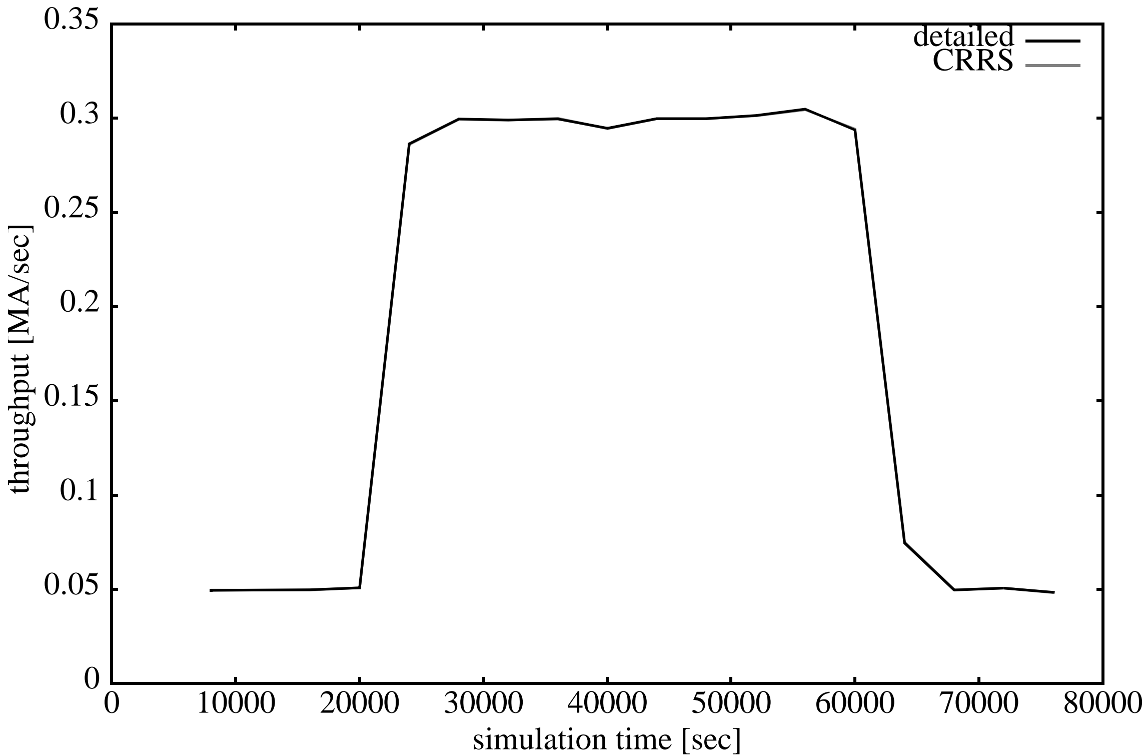

Figure A-15: **Throughput in an G/D_4/1 network (averages over 100 replications)**

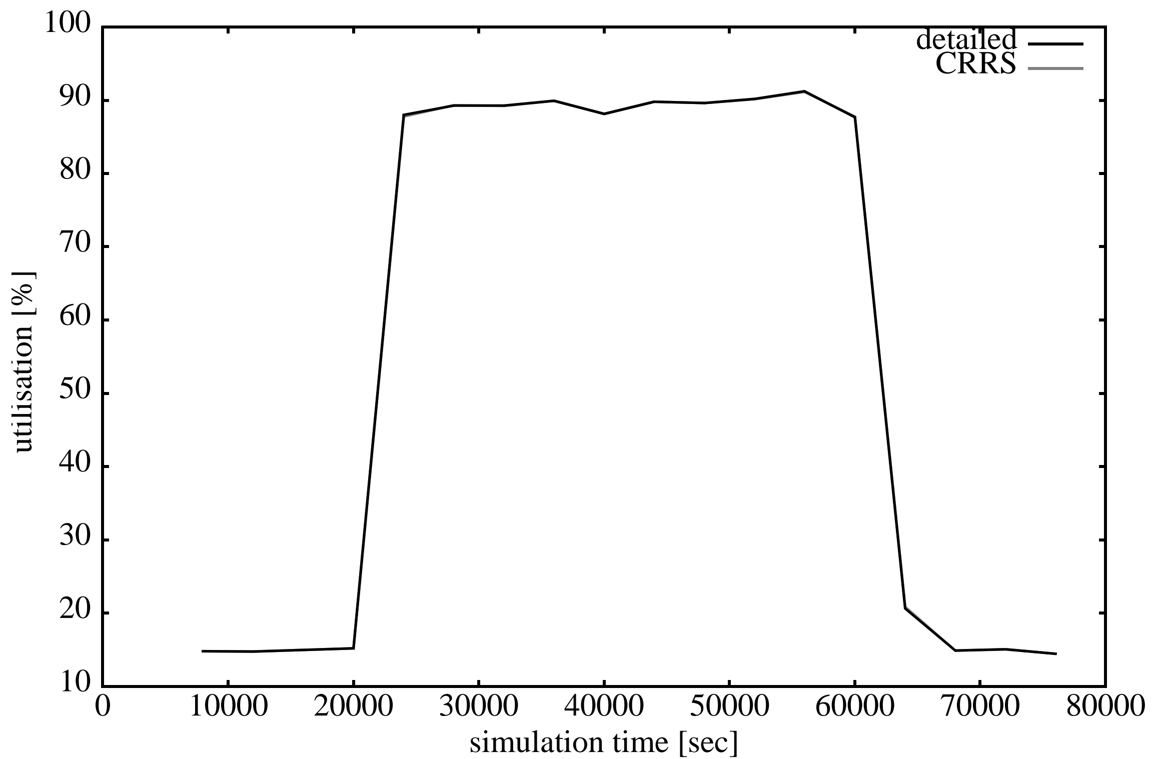

Figure A-16: **Utilisation of server "Bond" in a 10 Mbit network with mobile agents of 150 KB (averages over 100 replications)**

A.10 CRRS - Finite Horizon Analysis of System with G/G/1 Agent Servers

model time [sec]	90 % CI Detail model		% rel. stat. error	90 % CI CRRS		% rel. stat. error
8000	122.785	134.786	4.659	123.385	135.484	4.674
12000	124.068	135.735	4.491	125.805	137.929	4.597
16000	120.999	130.801	3.893	123.009	132.786	3.822
20000	108.494	117.992	4.194	108.577	118.551	4.391
24000	653.167	686.090	2.458	660.370	693.756	2.465
28000	1480.100	1553.227	2.411	1485.767	1558.403	2.386
32000	2022.951	2129.288	2.561	2033.831	2140.508	2.556
36000	2399.380	2530.388	2.657	2443.983	2565.529	2.426
40000	2658.833	2820.371	2.948	2708.512	2865.283	2.813
44000	2797.935	2952.722	2.692	2835.079	2997.852	2.791
48000	3175.430	3349.353	2.666	3216.972	3393.803	2.675
52000	3623.945	3787.638	2.209	3663.602	3837.020	2.312
56000	3838.714	4035.179	2.495	3917.987	4116.847	2.475
60000	4079.499	4268.394	2.263	4095.314	4303.641	2.480
64000	5985.847	6299.672	2.554	6049.319	6379.904	2.660
68000	4459.875	5014.918	5.858	4525.075	5100.405	5.977
72000	158.370	278.603	27.515	171.121	295.978	26.730
76000	126.044	139.458	5.052	130.065	143.499	4.911

Table A-20: **Confidence intervals (CI) of mean residence time (100 replication runs)**

A.11 CRRS - Finite Horizon Analysis - Efficiency Gains

Station types / PC	CPU time detailed model [min.]	CPU time CFS [min.]	Efficiency gains of CFS (decrease of CPU time)
$G/D_4/1$, c.o.v.$[A_i]$ = 3.0 on *blofeld*	96.031	12.259	87.2%
$M/H_4/1$ on *goldfinger*	87.825	6.899	92.1%
G/G/1, c.o.v.$[A_i]$ = 3.0, c.o.v.$[S_i]$ = 5.0 on *goldeneye*	58.223	17.970	69.1%

Table A-21: **Amount of CPU time**

Index

A

agent
- gateway 19
- mobile 1, 19
- service 8, 12, 21, 37, 49, 50, 51, 52, 53, 56, 70, 71, 72, 73, 74, 81, 88
- system 16, 18, 19, 25, 30

agent platform 1, 3, 4

agent server 1
- malicious 2

aggregated model 36, 37

aggregation 35, 36, 37, 38, 45, 47

Aglets 4, 10

Ara 4

arrival
- bulk arrival 37
- interarrival time 24, 37, 53, 89, 94
- rate 9, 10, 24, 40, 70, 73, 87, 89
- time 53

artificial intelligence. 1

autonomy 9, 37

B

batch means analysis 26
- batch size 26

BCMP networks 37, 41

benchmarking 4, 9, 40

Bin 13

binary agent code 10, 99

BlockQ 16

bottleneck 7, 22, 23, 86, 102, 107
- movement of 108

C

calibration 18, 24, 87, 89, 91, 102

capacity planning 2, 3, 4, 7, 85, 86, 97

client/server approach 1, 5

coefficient of variation 24, 28, 45

Collegiate Timetable Service 97, 98, 105

common random numbers 60, 81, 125

communication 8, 12, 18
- blackboard 12, 50
- transfer messages 1, 12, 18, 19, 50

completely decomposable 37

composite queue 36, 37

compute server 15, 16, 24, 70, 72, 75, 87

concatenated FIFO servers, CFS 52

concatenated round robin servers, CRRS 70, 71

Concordia 4

CondQ 13, 19

confidence interval 13, 25, 26
- paired-t 48, 90

confidence level 25, 91, 92

CPU 15, 16, 17, 24, 70, 75, 87, 90

D

decomposition 36, 37

DEMOS 13, 14

dispatcher 17, 72

distributed system 1, 2, 8, 25

DNS server 23, 24, 87

E

effective speed 70, 71

efficiency 3, 35, 40, 43, 47, 51, 62, 63, 69, 70, 71, 72, 75, 77, 78, 82, 83, 129, 130

entity 13

F

FIFO 15, 16, 18, 23, 37, 52, 53, 54, 60

file server 15, 16, 59

finite horizon analysis 25, 26, 27, 35, 91, 92

G

ghost delay 18, 102

Grashopper 4, 10

GrepAg 94, 102

H

home server 1, 5, 8, 10, 12, 19, 24, 25, 37, 59, 64, 87

hybrid modelling 3, 38, 39, 71

I

I/O device 15, 16, 52, 55, 90

importance sampling 38

infrastructure 4, 7, 10, 23, 24, 28, 87, 92

instrumentation 90

Internet 2, 8

Intranet 1, 2, 8, 98

J

Jade 4

JaDEMAS 10, 14

JAM 11

Java 1, 10, 11, 13, 43, 129

JavaDEMOS 10, 11, 13, 14

job list 71, 72

K

Kalong 4

L

Little's Law 40

load dependent server 55, 72

loose coupling 37, 45, 47

M

MAP 4, 5

MASIF 10

mathematical analysis 5, 9, 22, 23, 35, 91

measurement 4, 15, 85, 88, 89, 90, 93, 112

message queue 19, 21

migration 4, 11, 12, 23, 33, 89
  delay 24, 87
  overhead 18, 24, 87, 88
  pull-all-to-all 11
  pull-all-units 11
  pull-per-unit 11
  push-all-to-next 11
  strategy 11
  weak 11

modelling effort 3

MOLAB 93, 94, 98

Mole 4, 5

multi processors 17, 72

multiagent system 1

N

nearly completely decomposable, NCD 37, 39

network
  delay 22, 23, 26, 27, 50, 55, 56, 72, 73, 83, 89, 102
  extended network model 23, 87
  latency 1, 5
  load 4, 5
  model 22
  monitoring 94
  simple network model 22

Norton's theorem 36

O

OMG 10

overhead 18, 23, 24, 33, 87
  migration 18, 24, 87
  system 18, 24, 87

P

paired-t confidence interval 91

performance modelling 2, 3, 4, 5, 7, 9, 35, 85, 112

planned system 7, 85, 86, 91, 92

Pollaczek-Khintschin formula 40

pre-analysis 36, 37

product form 35

Q

quality of service 2, 7, 47, 92, 93, 106

quantile 31, 45, 92, 110

quantum 70, 74, 75

queuing network 35, 36, 39, 72, 91, 112

queuing system 35, 37, 40

R

rare event simulation 35, 38

real system 3, 85, 86

reduced rescheduling 71, 72, 74, 81

relative statistical error 25, 26, 91, 92, 93

Res 13, 16, 53, 54, 59

rescheduling 71

residence time 26, 27, 83, 90, 91
  CFS 50, 51, 52, 53, 55, 56
  CRRS 71, 72, 73, 74
  r_A 88, 89, 102
  r_N 88, 89, 102
  SHRiNK 40, 41, 42, 43, 44, 125, 126, 127, 128, 130

response time 91

round robin 16, 70, 71, 72

round trip 1
  agent round trip time 14, 23, 25, 26, 27, 47, 55, 73, 74, 75, 89, 90, 92, 94, 102
  TCP round trip time 24, 87, 99

routing table 22, 24, 27, 87, 99

S

SATP 11

security 2, 3, 8, 49

separable network 37

sequential simulation 26

service rate 18, 24, 37, 40, 55, 56, 72, 87, 91, 102

service time 9, 16, 18, 24, 26, 31, 37, 40, 41, 52, 53, 55, 56, 60, 70, 71, 88, 89, 90, 127
  residual 53, 72

SeSAm 5

SHRiNK 35, 40

Simalytic 38

simulation 3, 5, 7, 9, 10, 11, 12, 14
  discrete event simulation 11, 13

SMART 92

stationary agent 1, 8, 24, 26
  malicious 2

stationary agents 8, 10, 11

steady state analysis 9, 25, 26, 35, 37, 38, 40, 42, 59, 75, 91, 92, 105

submodel 36, 37, 38, 40, 45, 47

substitute representation 36

Swarm 5

T

TCP pipe 22, 23, 99

throughput 5, 14, 25, 26, 27, 89, 90, 91, 94, 102
  CFS 55
  CRRS 81

time slice 16, 24, 70, 87

Tracy 4, 10, 11, 33

transient analysis 9, 26, 83

U

user load 18, 24, 87, 88

utilisation 14, 18, 25, 26, 27, 89, 90, 91, 102, 125
- CFS 55, 56
- CRRS 73, 74
- SHRiNK 40, 126, 129
- utilisation law 55, 90

V

validation 89, 91, 93, 102

visit counts 9

Voyager 4

W

WaitQ 13, 16

what-if-analysis 2, 7

workload 9, 18, 88
- burstiness 89
- density 89
- description 24, 28, 29, 87, 92
- generator 10, 23, 89, 94
- source 59