Performance of distributed information systems

Widell, Niklas

2002

Performance of Distributed Information Systems

Niklas Widell

LUND UNIVERSITY

Department of Communication Systems
Lund Institute of Technology

ISSN 1101-3931
ISRN LUTEDX/TETS–1052–SE+78P

Printed in Sweden
KFS AB
Lund 2002

To Stina and Hugo

Contact information:
Niklas Widell
Department of Communication Systems
Lund University
P.O. Box 118
SE-221 00 LUND
Sweden

Tel: +46 46 222 71 95
Fax: +46 46 14 58 23
e-mail: niklasw@telecom.lth.se

Abstract

There is an increasing use of distributed computer systems to provide services in both traditional telephony as well as in the Internet. Two main technologies are Distributed Object Computing (DOC) and Web based services.

One common DOC architecture investigated in this thesis is the Common Object Request Broker Architecture (CORBA), specified by the Object Management Group. CORBA applications consist of interacting software components called objects. Two other DOC architectures investigated are the Telecommunications Information Networking Architecture (TINA) and a CORBA based Intelligent Network (IN/CORBA) system. In a DOC environment, the objects of an application are distributed on multiple nodes. A middleware layer makes the distribution transparent to the application. However, the distributed nature creates a number of potential performance problems.

Three problems in DOC systems are examined in this thesis: object distribution, load balancing and overload protection. An object distribution describes how objects are distributed in the network. The objective is to distribute the objects on the physical nodes in such a way that intern-node communication overhead is as small as possible. One way to solve the object distribution problem is to use linear programming. The constraints for the problem are then given by both ease of management of the system and performance concerns. Load balancing is used when there are multiple objects that can be used at a particular time. The objective of load balancing is to distribute the load efficiently on the available nodes. This thesis investigates a number of decentralized load balancing mechanisms, including one based on the use of intelligent agents. Finally, overload protection mechanisms for DOC systems are investigated. While overload protection is well-researched for telecom networks, only little work has been performed previously concerning DOC and overload protection.

Also, this thesis examines the use of overload protection in e-commerce web servers. Two schemes are compared, one which handles admission to the e-commerce site on request basis, and another which handles admission on session basis. The session based mechanism is shown to be better in terms of user-experienced performance.

Acknowledgments

First of all, I would like to sincerely thank my supervisors Dr. Christian Nyberg and Dr. Maria Kihl for their support and encouragements during the work with this thesis. In particular, I want to thank Christian for many, many interesting discussions and Maria for always helping me to find the right track when I strayed to far from my goals.

I would like to thank Professor Ulf Körner for being supportive in my work at the Department of Communication systems.

I would like to thank Torgny Holmberg because he is `root`.

I would like to thank the rest of the colleagues at the department for making it such a nice place to work at.

I would like to thank Kristofer Kimbler, now of Appium AB, for teaching me so much about service architectures.

I would like to thank Conor McArdle, of Teltec DCU, Ireland, for the all interesting work we performed together within the MARINER project.

A special thank goes to my family for all their encouragements and support. Thanks to Anders and Gertrud and my sister Karolina for supporting me.

Finally, all my love to my wife Stina and my son Hugo. Thanks for making my life joyful and full of happiness and for supporting me both at home and at work.

Contents

1 Introduction

The explosive growth of information technology creates a need for new and more flexible ways of implementing network services. The fast technological development and a telecom market moving towards de-regulation with a larger number of participants, means that the architectures on which the telecommunications networks are based need to become more flexible. In many cases this flexibility requires more intelligence in the networks in order to supply new services.

One technique used to make a system more flexible is Distributed Object Computing (DOC). The distribution makes it possible to spread the computational tasks among multiple processors in order to share the load. Object orientation is an important technology for creating great flexibility in the creation, maintainance and support of services.

One of the major DOC architectures is CORBA (Common Object Request Broker Architecture), a standard defined by the OMG (Object Management Group). CORBA supports the execution of object oriented programs across multiple nodes or processors. CORBA works as *middleware*, where a common layer is introduced between the application and the processing nodes beneath it. The middleware layer is called the Object Request Broker (ORB), which acts as a common communication channel between objects, making object interaction fully location transparent. Even if CORBA was not defined with telecommunications in mind, it is a promising architecture that shows both great flexibility as well as opportunities for quick service development.

The Telecommunications Information Networking Architecture (TINA) is another DOC architecture defined specifically to support telecom services. TINA is developed by the TINA Consortia (TINA-C). TINA is aimed at providing a full set of specifications for the support and execution of telecom services, in a fully distributed object oriented environment.

A middle ground between TINA and CORBA is IN/CORBA. This concept uses services as defined within TINA to replace the service logic functionality of a traditional Intelligent Network (IN) Service Control Point (SCP). It uses CORBA, with certain extensions, to implement some of the service logic parts of TINA.

Common to all the mentioned systems is a need to investigate performance and Quality of Service (QoS) aspects and try to formulate how these factors can be fulfilled. Currently, only little such work has been performed. Due to the distributed nature of the systems, as well as the general complexity of the services executed on them, performance analysis is difficult. Since the systems aim to compete in the telecom market, where traffic has real-time characteristics, the performance aspects are very important and must thus be studied carefully. Indeed, one major problem of the proposed systems is that the systems themselves can be performance bottlenecks. Objects may be distributed on multiple physical nodes. This means that different object distributions will require different amounts of inter-node communication. Inter-node communication is expensive in overhead and it is an important objective to keep it as small as possible while maintaining the positive sides of distribution.

The increasing reliance on the Internet for many organizations, businesses, government agencies and individuals requires a deeper understanding of how the Internet works. One area of active development is e-commerce, where a company's business, either directly with a consumer or with another company is handled over the Internet. E-commerce has many promises, but also many problems. One major factor, which has not received very much attention, but which is nevertheless very important, is performance of e-commerce sites. If the performance of an e-commerce site is poor, potential

customers will go to other sites. This means that the site generates no revenue, which is obviously not good in business. The most common solution to the performance problem is just to throw more money in, buying more and better servers, adding more cache, etc. These solutions, however, sometimes fail for a number of reasons. First, servers cost money, and the days are over when unlimited budgets for IT-departments were the norm. Second, correct dimensioning is very difficult, since the performance of a particular machine running a particular piece of software is seldom known *a priori*. The performance of an e-commerce site depends on many factors, among them are: hardware, network, code quality, site content etc. In addition, the contents of the site often changes greatly from day to day, with resulting changes in internal traffic patterns. Third, traffic prediction is very difficult, often being nothing more than wild guesses wrong 10 to 100 times up or down. Fourth, web servers are built using the same best-effort tradition as the rest of the Internet. This means that there are no special facilities to handle overload, apart from very low level resource constraints.

The main topics of this thesis are object distribution, load balancing and overload protection for DOC systems and overload protection for e-commerce web servers. Due to the little work performed in these areas for the particular systems, a large part of the work has been the development of accurate models of the systems. The main tool for investigation has been simulation.

The work is presented in a collection of four papers. The papers are presented as published or submitted, with only few minor editorial changes. The remainder of this introduction contains the following: a description of Distributed Object Computing, in particular CORBA, IN/CORBA and TINA, an overview of e-commerce web servers, a list of performance issues in DOC, a summary of results as well some ideas for future research. A list of publications appears at the end of the introduction.

2 Distributed Object Computing and Middleware

This section provides a short introduction to the area of Distrbuted Object Computing. It also discusses three examples of middleware systems: CORBA, IN/CORBA and TINA.

A distributed system is a collection of independent processors, connected through a communication network. The aim is to hide the distributed nature of the system, so that it appears as a single processor system to the user. A distributed application may be developed in many different ways. One common way is to use object oriented methods.

An object oriented application consists of a number of objects. An object is defined in terms of data and operations on that data. The operations are usually called methods. An object can invoke methods on other objects, which means that an object can use the operations of another object. Each object has its own data, and each object can be referred to by using a unique identifier.

A distributed application consists of tasks, where each task can be modeled as a sequence of method invocations. The execution of a task may span over many different processors. A task is generated by a user of the system, or for instance by a timer signal or the completion of another task.

The common way to use DOC is to use a middleware layer between the application objects and the operating system. The middleware layer provides a common programming abstraction that handles the communication between the (possibly heterogeneous) processors.

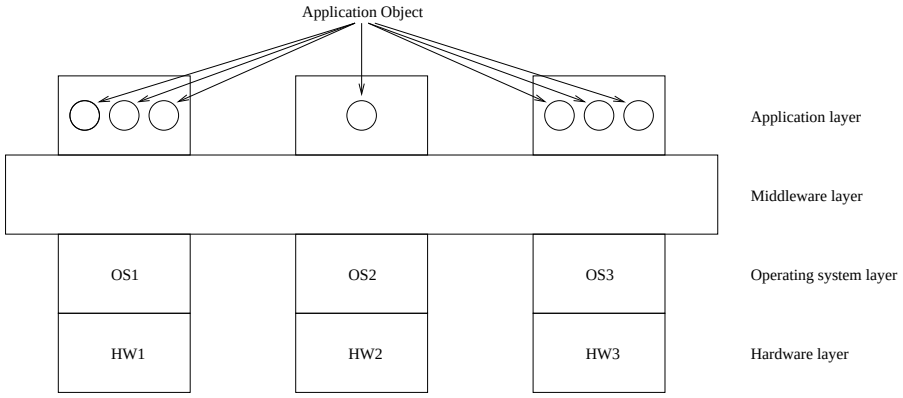

Figure 1: Application, Middleware, Operating system and Hardware layers

Figure 1 shows a sample system with a number of application objects executing on a middleware platform. The middleware layer shields the application developer from all low-level, tedious and error-prone implementation details of the underlying system, while providing a consistent set of abstractions for development of applications. Three main DOC models exist: OMG's CORBA [1], Microsoft's Distributed Component Object Model (DCOM) [2] and Sun's Java Remote Method Invocation [3].

2.1 CORBA

The Common Object Request Broker Architecture (CORBA) [1] by the Object Management Group (OMG) [4] is a widely accepted commercial standard for DOC middleware. CORBA provides the capability to do method invocations between CORBA objects transparently in a distributed system. An object in OMG's object model is an encapsulated entity with a distinct identity that can be accessed only through well-defined interfaces. In addition to the location transparency, CORBA also provides other powerful transparencies. CORBA is independent from programming languages, computing platforms and networking protocols, and therefore very suitable for distributed applications with heterogeneous distributed systems.

The three main components of CORBA that provides the transparencies are the system-neutral Interface Description Language (IDL), a set of language mappings and the Object Request Broker (ORB). CORBA separates the specification of an object's services and the actual implementation of those services by using IDL. An object's IDL describes the functional interface of the object and the type signatures of the operations of that object, completely independent of any programming language, operating system or communication medium.

The required code for the implementation of an object can be generated using a given IDL for a particular object and a language mapping to a particular language, such as C++, Java or C. The generation takes place in two parts, both for the client (calling) object and server (responding) object. A client uses a stub generated from the IDL to make a method invocation to a server object, while the server object uses a skeleton generated from the same IDL to implement the method. Both implementation details of the method and where the server is located is completely hidden from the client. Thus, a client object written in C++ may access a server object written in

Smalltalk, being completely unaware of where the server is located and how it is implemented, as long as it knows the IDL for the server's interface it wants to use.

The Object Request Broker (ORB) acts as a message bus to provide seamless interaction between objects. The ORB has the same functionality as the middleware layer in figure 1. The ORB also handles a set of Object Services, for the support of standard client/server interaction, such as a Naming Service that allows objects to be identified and accessed by names, a Life-cycle Service that can create and destroy objects and an Event Service that can transfer events using event channels.

The inter-operability among heterogeneous ORBs is achieved by using the General Inter-ORB Protocol (GIOP). The most common specialization of GIOP is the TCP/IP-specific Internet Inter-ORB Protocol (IIOP). By using GIOP/IIOP, objects running on different ORBs, by different vendors and on different machines, can interwork.

The interaction between two objects works like this: A client `Request` invocation is marshalled and dispatched to the ORB by the client stub. The ORB translates the message and uses GIOP/IIOP to transfer the request to the ORB on which the server object is located. If necessary, the client-side ORB uses the Naming Service to find where the server object is located by using it's reference to the server object. The request is then unmarshalled at the server side by the server skeleton and delivered to the server object. The server generates a `Reply` which is then delivered back to the client object.

2.2 Intelligent Networks using CORBA

Distributed computing middleware, such as CORBA, is being proposed for future Intelligent Network platforms as an appropriate solution to improve scalability, reliability and flexibility of the current IN systems, see for instance Capellman *et al* [5]. Today's Intelligent Networks have generally been viewed and implemented as tightly coupled, vertically integrated applications, with communication capabilities in the form of given "capability sets". Most of the installed systems acting as Service Control Points are monolithic products based on traditional, and reliable, switching system platforms. The disadvantages of this approach, although still technically adequate and economically viable, are little software re-use, reduced scalability and performance, cumbersome service management and limited interconnecting capabilities with other networks such as Internet, Intranet and private databases.

Much of the investigation into the future application of CORBA to IN systems has been initiated by the Eurescom P508 project [6]. This objective of this project was to determine the options for evolving from legacy systems towards TINA. The architecture of the middleware layer is based on an enhanced CORBA ORB [1], targeted at telecom applications. Therefore it must include real-time features and a flexible timeout mechanism. On top of the ORB, some generic CORBA services, i.e. not specifically targeted at telecom applications, are needed. At least, the Naming Service and the generic CORBA Event Service are needed, but other services may be also required. Generic CORBA services, however, are not enough to fulfil telecommunications application requirements. Telecom-specific services are needed, for instance to handle different kind of events and to establish a context (session) for the provision of telecom services. For inter-operability with legacy systems, a gateway between CORBA and the Signalling System No. 7 (SS7) protocol stack must be introduced. This gateway would enable inter-operation between CORBA servers and legacy IN.

2.3 Telecommunications Information Networking Architecture

The TINA architecture provides a set of concepts and principles that are to be used for specification, design, implementation and operation of software systems for telecommunication networks. TINA is based on two fundamental observations made about telecom networks. First, software in telecom networks can be seen as (large) distributed software systems, where techniques for distributed computing can be used. Second, the system itself can be described using Object Oriented methods.

TINA itself is an independent extension of CORBA , a standard for object middleware in a computer network. The TINA standard, for instance, adds stream bindings, a concept that does not appear in the original CORBA specifications.

The approach taken by TINA is to implement telecom services using Object Oriented methodologies. Each service is described as a set of interacting objects called Service Components (SCs). Each SC consists of one or more Computational Objects (COs) that are executed in a Distributed Processing Environment (DPE). The DPE corresponds to the middleware layer in the general DOC model. The DPE shields the services from the distributed nature of the system, taking care of communication between objects and maintains location and communication transparency in the system. The DPE in layered above the Native Computing and Communications Environment (NCCE), the native software that runs on the hardware.

In TINA, the DPE takes care of the distributed nature of the system. The computational objects of an application may reside in different places in the network, but the distribution should be fully transparent, both to the application itself and to the application developer. This is called access and location transparency.

The DPE also also has a Lifecycle mechanism, that can create, suspend, resume and destroy objects. Lifecycle mechanisms are particulary important in distributed systems, as an object may create another object on another node. Some computational objects in a TINA system are needed during just one single session, while others might "live" during long time periods (e.g. Databases).

To enable communication between remote COs and inter-DPE signalling, the DPE kernels on different nodes communicate with each other. This communication is achieved through the Kernel Transport Network (kTN) as described in [7]. The kTN is a technologically independent virtual network that is logically separated from the transport network. The kTN is similar to the Signalling System No. 7 (SS7) used in IN.

In TINA, a *session* is defined as being a series of activities carried out with the purpose of achieving a goal. There are four kinds of sessions in TINA:

- A Service session represents a single activation of a service. It connects the users of a service, so that they can interact, and keeps note of what resources are active.
- A User session represents a single user's interaction with a Service Session. It keeps track of information about the user, such as how long time it has been active, current status etc.
- A Communication session is used in a Service Session to keep track on any open stream connections (see below), checking for instance communication paths and Quality of Service parameters.
- An Access session keeps track on a user's interaction with the system, for instance holding information about what services a user presently has active.

All sessions can be suspended and later be resumed with no need for complicated setup. For instance a Service Session such as a multi-media conference might be suspended over the night. All expensive communication resources may then be released. The session may then be resumed in the morning again at the same state (i.e. same users and resources) as it had when it was suspended.

The reason for dividing into several different kinds of sessions is to separate out concerns. For instance the separation of Access and Service Sessions allows for the users using a service to change their location in the system. The separation of Service and User Session allows for the fact that not all users participating in a service may want or be allowed to use the same resources.

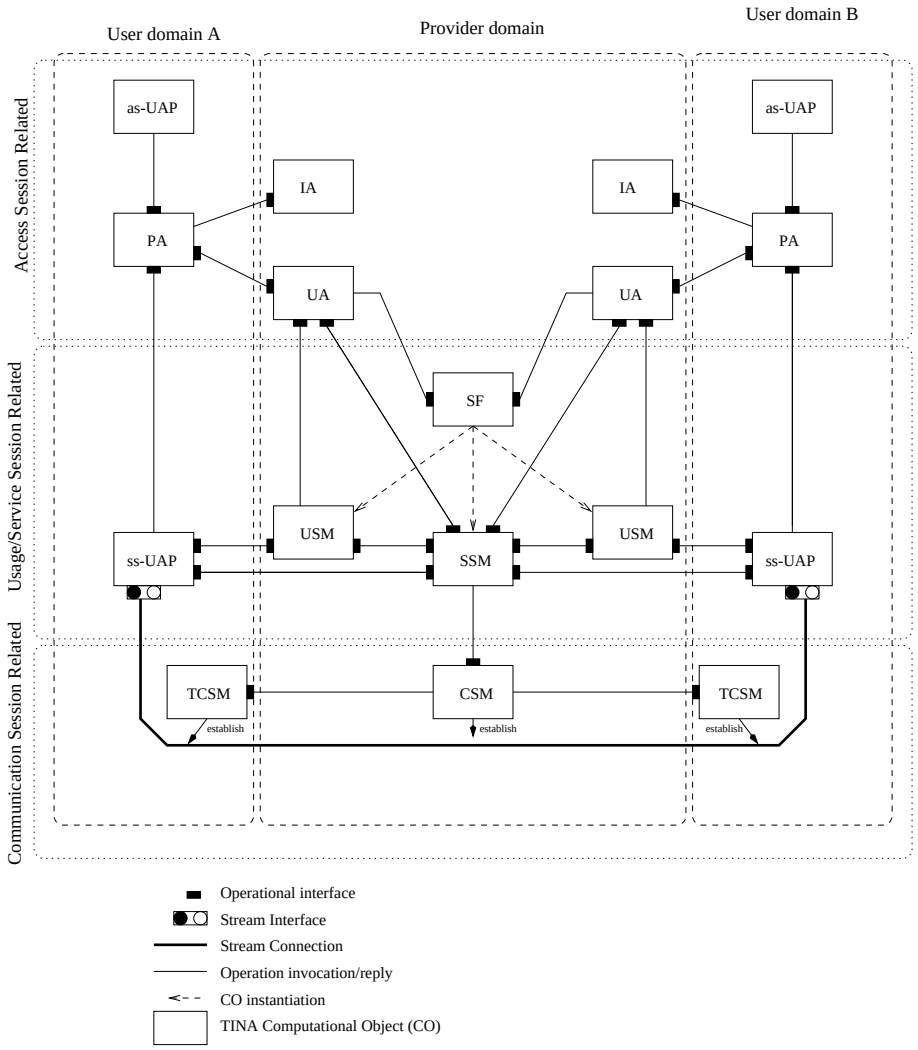

Figure 2: Computational Objects, Sessions and Domains in TINA

A Computational Object (CO) in TINA encapsulates data and functionality. Each service consists of a number of COs interacting through interfaces. Some COs are

service specific, while others are not. The only way in which COs can interact are through interfaces.

COs are divided into domains, according to which role they take in the service. Three domains are specified (see [9]). First is the *user domain*, that contains objects representing either service users or otherwise closely tied to the user. Second is the *provider domain*, that contains objects representing the different services available. Third is the *peer domain* that contains objects used when two provider domains cooperate together.

Figure 2 shows the relationship between COs, sessions and domains for the set of of COs specified in [9].

3 E-commerce Web Servers

Electronic commerce systems have become increasingly important in the business world. E-commerce services include web stores, e-traders, web auctions and Internet banks.

The basic protocol for web transactions is the HyperText Transfer Protocol (HTTP). HTTP is a lightweight stateless file transfer protocol. A web page ususally consists of one or more Hypertext Markup Language (HTML) files with zero or more embedded files such as pictures. A page is said to be static if the HTML code is not generated during run-time. A page is said to be dynamic if it is generated as a response to a HTTP-request.

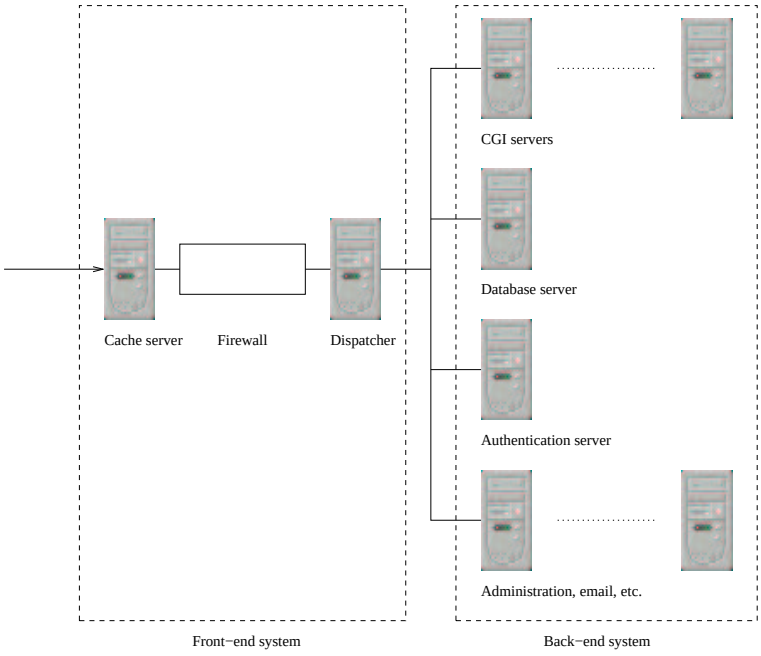

Figure 3: A typical mid-size e-commerce site

E-commerce web sites may obviously vary much in size: from small single computer systems to huge installations integrating hundreds of servers and mainframes. Figure 3

presents a typical e-commerce site for a mid-size business. The system is typically divided into two major parts: a front-end and a back-end.

The components belonging to the front-end are typically: Cache server, Firewall and Dispatcher. The cache can serve static HTML very quickly. Given a large cache, only a fraction of static HTML pages will be required to be served by the web servers. The firewall protects the system against intrusion. The dispatcher, or load balancer, distributes arriving requests to the web servers.

The components belonging to the back-end are typically: Web server, Database server, Authentication server, E-mail server and various administrative servers. The web server processes both static and dynamic HTML. During the generation of a dynamic page, interaction with the database and authentication servers are often required. The database server typically hold inventory, prices etc. in one or more databases. The authentication server is used to check the identity of users.

4 Performance Issues

The performance related work in this thesis concerns three issues: object distribution, load balancing and overload protection. This section gives a brief introduction to these problems.

4.1 Object distribution

The objects of a DOC application can be distributed more or less freely in a system. By increasing the distribution, i.e. by letting objects be more spread out, the processing power of more nodes can be used. More nodes means larger capacity, which means that more work can be accomodated. However, the downside with distribution is that the objects need to communicate. Communication costs processing power, which means that less capacity is left for real work. This leads to the object distribution problem, which may be formulated as: How can the objects of the application be distributed so that as much of the capacity is used as possible, while trying to limit the communication overhead between nodes?

4.2 Load balancing

Given an object distribution, there may be times when a choice must be made between two (or more) objects during the execution of a task. The objective for load balancing is to spread the load on the nodes so that no node is overloaded while another node has spare capacity.

4.3 Overload protection

Using good object distributions and load balancing algorithms allow the capacity of a system to be increased. However, the capacity is still finite. This means that at times of high traffic, the system may become overloaded. The common way to handle overload is to reject tasks coming to the system. Tasks can be rejected either at the entry to the system (system level or external overload protection) or at a node inside the system (node or internal overload protection). The overload protection problem may be formulated as: Which tasks should be rejected during overload so that the objectives of the application are fullfilled the best?

5 Summary of Results

This section outlines some of the results presented in the following papers.

Papers 1 to 3 introduce queuing models of TINA, IN/CORBA and CORBA systems. Paper 4 introduce a queuing model of an E-commerce web site.

A number of load balancing algorithms are compared in papers 1 and 2. All algorithms are decentralized, in that they do not need any centralized processing to operate. Paper 1 discuss four algorithms: one based on node queue lengths and three variants of random selection. The first random algorithm distributes load with fixed probabilities. The last two are adaptive and base the load distribution probabilites on the amount of internally rejected messages. The shortest queue performs best, but it is not feasible due to the amount of communication necessary. The adaptive random algorithms perform better than the fixed random. However, it is shown that the adaptive algorithms allows tasks to continue executing longer than the fixed random algorithm. This causes tasks to be rejected later, resulting in more wasted resources. This shows the need for external load control. Paper 2 uses fixed random and shortest queue algorithms as benchmark algorithms. These are compared to algorithms based on three ant based algorithms. An ant is light-weight mobile agent moving randomly in the network distributing load information. The ant based algorithms performs slightly worse than the benchmark algorithms, at least for static workload, as was the case in paper 2.

A method for optimal distribution of objects is presented in paper 2. The method formulates the optimal distribution problem as a Linear Programming problem. The objective is to minimise total processing cost on each node given a particular traffic to the system. The solution to the problem determines the relative flows between objects, and these flows can be used to derive routing probabilities for a random load balancing algorithm.

In paper 3 overload protection for CORBA systems with time constraints is discussed. As overload is rarely discussed for DOC systems, the paper has a thorough discussion of what overload is in terms of CORBA. A number of traditional overload protection mechanisms are then compared. A window based mechanism performs better than either call gapping or percent blocking. A response time based control loop is used to set mechanism parameters.

In paper 4 two types of overload protection for e-commerce web servers are compared. Users send requests to the site. The set of requests sent by one user is called a session. Sessions end with the user leaving the site either happy or angry. In the former case all requests sent by the user are processed completely. In the latter case the user is rejected first after a set of successful requests has been sent. Two schemes for overload protection are examined: request based and session based. The request based scheme limits the number of requests in the system. The session based scheme limits the number of sessions in the system. It is shown that the throughput is the same for both schemes, if the length of a user session is geometrically distributed. However, it is also shown that the number of angry users in the request based scheme is much larger than in the session based case. Therefore, session based admission control is preferable.

6 Future Research

Much work remains to be done in the area of performance aspects of distributed systems.

Better object distribution mechanisms should be adaptive to changing traffic patterns. One way that could be investigated further is to use the method using Linear Programming in paper 2 and calculate object distributions for a number of traffic scenarios. If a there is a large change in the input traffic, then objects can migrate to a new pre-planned distribution.

One difficulty for object distribution is that the computations become extremely heavy as the set of nodes, objects and tasks increase. However, for the solution to be optimal, significant knowledge about execution times of object methods, network latencies etc. is required. This means that even if an optimal distribution is found after long and heavy computations, it may be based on inaccurate system measurements leading to a (possibly) incorrect solution. Therefore, a dynamic and adaptive approach, based on heuristics and information feed-back during execution, will be a very interesting subject for research.

One area which current load balancing algorithms fail to acknowledge is that during the execution of a method, the server object may act as a client and make an method invocation to a third object. A load balancing algorithm may take this into account when making decisions on which object to chose.

Also, further investigation on the behaviour of load balancing algorithms during traffic transients is an area for more future work. Investigation of interaction between object migration and load balancing is also yet to be done.

Overload protection for DOC systems can be improved and made an integral part of a DOC framework. More investigations concerning adaptive overload protection mechanisms suitable to highly dynamic systems is also necessary.

Much remains to be done in the area of understanding web performance. Due to the many components involved, the dynamic nature of the web, the unpredictability of the Internet and the general immaturity of the field, building web server performance models is difficult. An understanding of how the different components interact is needed. In addition, models of how users interact with a site is important for traffic modeling.

7 List of Papers

The following papers appear in this thesis:

1. Kihl, M., Widell, N., Nyberg, C., *Load Balancing Strategies for TINA Networks*, Proceedings, 16th International Teletraffic Congress, Edinburgh, Scotland, June 1999.

2. McArdle, C., Widell, N., Nyberg, C., Lilja, E., Nystrom, J. and Curran, T., *Simulation of a Distributed CORBA-based SCP*, Proceedings, 7th International Conference on Intelligence and Services in Networks, IS&N 2000, Athens, Greece, February 2000.

3. Widell, N., Nyberg, C., Kihl, M., *Overload Protection for CORBA Systems with Time Constraints*, to be submitted. Paper is extended version of presentation given at OMG Workshop on Real-time and Embedded Distributed Object-Oriented Systems in San Francisco January 2002.

4. Kihl, M., Widell, N. and Nyberg, C., *Performance Modeling of Distributed E-commerce Sites*, submitted to Networking 2002.

The following papers to not appear in the thesis:

5. Widell, N., Kihl, M. and Nyberg, C., *Measuring Real-time Performance in Distributed Object Oriented Systems*, Proccedings, Performance and Control of Network Systems III, Boston, Massachusetts, September 1999.

6. Widell, N. (Editor) and Parsons, S. (Editor), *Recommendations on the Application of the MARINER Agent Architecture to IN/CORBA and MoIP*, ACTS Project MARINER Deliverable 7, February 2000.

7. Widell, N. and Nyberg, C., *Gateway-based Call Admission in Distributed Object Oriented Systems*, Proceedings, Fifteenth Nordic Teletraffic Seminar, Lund, Sweden, August 2000.

8. Kihl, M., Widell, N., Nyberg, C., *Perfomance Modelling and Control of Distributed E-commerce Sites*, Proceedings (on CD), SSGRR-2001, Infrastructure for e-Business, e-Education and e-Science on the Internet, L'Aquila, Italy, August 2001.

References

[1] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, Latest version 2.4, October 2000

[2] D. Box, *Essential COM*, Addison-Wesley, January 1998

[3] A. Wollrath, R. Riggs and J. Waldo, *A distributed Object Model for the Java system*, Computing Systems 9(4):265-290, Fall 1996

[4] www.omg.org

[5] C. Capellman, C.A. Licciardi, R. Minerva, G. Spinelli , P. Loosemore, T. Mota and T. Rodseth, *Migration Scenarios for evolving to TINA*, TINA'96 Conference, Heidelberg (Germany), September 1996.

[6] EURESCOM, *Introduction to Distributed Computing Middleware in Intelligent Networks: A Eurescom P508 Perspective*, OMG DTC Document orbos/97-09-11, September 1997

[7] M. Kudela and K. MacKinnon, *TINA Engineering Modeling Concepts (DPE Kernel Specification)*, TINA-C, www.tinac.com, 1994.

[8] P. Graubmann and N. Mercouroff (editors), *TINA Engineering Modelling Concepts (DPE Architecture)*, TINA-C, www.tinac.com, 1994.

[9] L. Kristiansen (editor), *TINA Service Architecture*, TINA-C, www.tinac.com, 1997.

Load balancing algorithms for TINA networks

Maria Kihl, Niklas Widell and Christian Nyberg

Department of Communication Systems, Lund University,
P.O. Box 118, SE-221 00 Lund, Sweden
{maria,niklasw,cn}@telecom.lth.se

Abstract

TINA is an open, object oriented, distributed telecom architecture, with many concepts taken directly from the latest computer research. In TINA, instances of the same object type can be placed on different physical nodes. Therefore, the network performance can be improved by introducing load balancing algorithms. These algorithms should distribute the traffic between the object instances in such way that the overall throughput and setup time are improved. We discuss and examine a number of simple distributed load balancing algorithms, that do not require any extra load information exchange between the nodes. The results show that it is difficult to find an algorithm that behave well for all traffic situations. The main problem is that the algorithms have not enough information about the load situation on the different nodes, since no load information is exchanged between the nodes. This problem can be solved by adding the feasibility of load status information to the TINA protocols.

1 Introduction

The trend in modern telecommunication systems is to provide a whole array of services, such as Freephone and Video conferences. To support these services, computers are integrated in the networks, adding for example central databases and allowing more complex service logic to be implemented. The Intelligent Network (IN) is an attempt to create a set of reusable components that can be connected to perform a new service. IN uses special nodes, called Service Control Points (SCPs), that are capable of executing complex services. If a service cannot be performed locally in the switch, it is passed on to the SCP. However, INs are not flawless. For instance, SCPs are potential bottlenecks and management of the network is difficult. Further, there is little vendor independence, which is an important factor in today's deregulated telecom market.

To solve these problems, the Telecommunication Information Networking Architecture (TINA) is being developed. TINA is an open, object oriented, distributed telecom architecture, with many concepts taken directly from the latest computer research. TINA is intended to be a good platform for service development in a rapidly expanding telecom environment. TINA is developed by a consortium which includes network operators and telecommunication and computer equipment suppliers. It is based on the Common Object Request Broker Architecture (CORBA), developed by the Object Management Group (OMG), that is a standard for object distribution and communication in computer networks. Services that may be provided by a TINA system include voice-based services, interactive multi-media services, information services and management services.

In TINA, instances of the same object type can be placed on several nodes. Since the traffic thereby must be distributed among the object instances, good load balancing algorithms could increase the network performance. The main objectives of a load balancing algorithm is to improve the throughput and lower the setup times by distributing more traffic to lightly loaded nodes than to nodes with a high load.

Several papers have discussed load balancing and load sharing in computer networks. However, none of these papers examine load balancing in TINA networks. Kremien and Kramer [6] discussed load sharing algorithms for distributed systems. Kunz [5] examined how the network nodes should exchange load status information to be used in load balancing schemes. Jordan and Varaiya [11] investigated call admission control policies for communication networks that support several services. Here, it is assumed that the control scheme knows the status of all network resources. Ramakrishnan et. al. [9] defined a optimization problem for how to assign a number of tasks to a number of processors in order to minimize, for example, the maximum completion time.

This paper extends the work by Kihl *et. al.* [14] [15] and it is partly based on the Master Thesis by Widell [18]. We discuss and examine feasible load balancing algorithms for TINA. The algorithms do not require any load status communication between the nodes, since they only consider the number of rejected signals. The investigations show that the algorithms behave differently depending on the traffic situation. Therefore, we also have a discussion about alternative algorithms that require some load status communication between nodes. Such an algorithm should have a good behaviour during all traffic situations.

2 TINA

The TINA architecture provides a set of concepts and principles to be applied in the development of software for telecommunication systems (see Chapman and Montessi [1]). TINA is very comprehensive. In order to be a fully distributed architecture, TINA contains a number of domains. The objects in two different domains are logically separated from each other. One domain is the *retailer domain*, which is intended to contain components necessary to create a "market place" for services provided in the *service provider* domain. The *user domain* contains the users of services and items closely related to the user.

Further, there are in TINA three types of sessions, that perform a set of activities during a specific period in time. The *Access session* deals with authentication and authorization of users, support of users' preferences, support for mobility and control of service interactions.The *Service session* provides users with an environment to support the execution of a service. Finally, the *Communication session* supports the activities needed to establish the communication between users.

The *service component (SC)* is an abstraction that encapsulates data and processing. It consists of a number of computational objects and provides a set of capabilities that can be used by other objects through two types of interfaces; the *stream interface* and the *operational interface*. The stream interface connects communication endpoints producing or consuming information flows, for example video or voice bitstreams. The operational interface deals with operations and requests from one object to another.

The TINA-consortium has defined a set of generic service components. Here follows a list of the more important ones:

Communication Session Manager (CSM) The computational counterpart of the Communication session.

Service Session Manager (SSM) Supports the capabilities that are shared among the users in a Service session.

User Session Manager (USM) The software representation of a user in the retailer domain.

Initial Agent (IA) and User Agent (UA) Act on behalf of a user within the network.

Provider Agent (PA) Is the user's point of contact with the provider.

User Application (UAP) Represents in the user domain the application that the end-user needs to use a service.

Terminal CSM (TCSM) Supports the Communication Session in the user domain.

Service Factory (SF) Creates instances of SSMs and USMs.

Generic Session End Point (GSEP) Connects the UAP and the USM.

A TINA application consists of computational objects that interact with each other. One feature in TINA is that an object does not have to know on which node another object is implemented. Instead, logical addresses are used in the communication. To accomplish this, TINA has a *Distributed Processing Environment (DPE)* that is used for the communication between objects (see Graubmann et al. [2]). The DPE knows on what nodes the objects are placed on. This means that from the computational objects' point of view, there is no difference between local and remote communication.

The DPE supports a number of services that are available for all applications. Here follows some examples of these services:

Trading service Helps computational objects to find appropriate objects to interact with.

Notification service Sends broadcasted messages, that may be picked up by any listening computational object.

Performance monitoring service Allows operators to monitor network performance.

3 A TINA service

Minetti and Utsunomiya [3] describe a simple TINA service in which two end-users exchange data via stream interfaces on the users' application objects (for example, an ordinary telephone call). In this paper the same service is used in the investigations. This section contains a description of how the service is modelled in the investigations. In the model only the setup of the service is considered.

The service uses ten different service components belonging to the user domain and the provider domain. The user domain components are Provider Agent (PA), User Application (UAP), Generic Session End Point (GSEP) and Terminal Communication Session Manager (TCSM). The components in the provider domain are User

Agent (UA), Initial Agent (IA), User Session Manager (USM), Service Session Manager (SSM), Communication Session Manager (CSM) and Service Factory (SF).

In the investigations a simplified SC model is used. First, the SCs in the user domain are modelled as one SC called USER. The reason for this is that the user domain components will probably be placed on the same physical node close to the users. Second, the IA and PA are modelled as one SC called *AGENT*. The other SCs are called the same as before, that is *USM, SSM, CSM* and *SF*.

The setup of a call (that is, the setup of the stream interface) consists of a number of signals being exchanged between the objects. Sometimes, the objects have to use a so called DPE service, for example the trader service. Table 1 contains the number of signals and DPE service requests that is needed to setup a call.

	USER	AGENT	USM	SSM	CSM	SF
Number of normal signals	10	9	11	7	2	5
Number of DPE service requests	0	4	0	1	0	0

Table 1: Signalling model

4 Network model

The network consists of K nodes that communicate via a signalling network. We assume that the network is very fast which means that the switching times are negligible. The network supports the service described in Section 3. Each node handles a number of SCs. The communication between the SCs is performed via a DPE that is placed on each node. A particular SC type can be placed on several nodes. In TINA, the computational objects in a service component can be placed on different nodes. However, we assume that all objects in an SC are placed on the same node. Further, SCs do not migrate in the network, since we assume that a steady state concerning the placement of SCs has been reached. Figure 1 shows a network with three nodes and three types of SCs.

New service requests arrive at the network in a Poisson stream. The requests are evenly distributed among the USER objects. When a USER object receives a service request, the setup sequence starts. The setup sequence consists of a number of signals that are sent between the SCs (see Section 3). Each signal generates a job that must be processed on the node on which the receiving SC is placed. After the processing, the next signal in the setup sequence is transmitted.

The nodes are modelled as single server systems with infinite FIFO job queues. We assume that they have unlimited memory capacity. The jobs belonging to the SCs on a node are executed concurrently with a simple time-sharing algorithm. The execution time for a job depends on what kind of SC it belongs to. If the SC uses a DPE service, the execution time is multiplied with a certain factor.

The transmission times are modelled as an added execution time in the sending and receiving nodes. These times represent the packing and unpacking of protocols. We assume that the DPE is intelligent enough to be able to separate between local and remote communication. Therefore, the transmission time is zero if a signal is sent between two SCs that are placed on the same node. Note that the SCs themselves have no knowledge about the physical addresses. Only the DPE knows on which network node a particular SC is placed.

Each node has an internal overload control scheme for protecting themselves. The objective of the internal overload control scheme is to keep the load on the node at a target load, ρ_t . The scheme measures the average load on the node during intervals of length τ seconds. If the measured load is above the target load, only a certain fraction of the incoming signals is accepted. The acceptance probability during interval k, $p(k)$, is calculated as follows. Let the measured load during interval $k-1$ be $\rho(k-1)$. If $\rho(k-1) > \rho_t$ then $p(k) = p(k-1) - step$, else $p(k) = p(k-1) + step$. $p(k)$ can never be less than zero or greater than one.

One problem with an internal control is that it is not very efficient (see Kihl [14] or Houck et. al. [11]). In order to achieve a high network throughput, calls should be rejected as early as possible. If a call is rejected late, much processing capacity is wasted. Therefore, in order to improve the internal control scheme a call can be rejected only the first time it uses a service component. If a call has had one accepted signal to a particular SC, all other signals to this SC are accepted. In this way, the overall network throughput is improved. However, there will still be much rejection in the network, since a call consists of signals to several different SCs. To achieve a maximum throughput, a global control scheme which tries to optimise the overall network performance should be implemented. We have decided not to implement such a control scheme, since this scheme would interfere with the load balancing algorithms that are the main concern of this article.

5 Load balancing algorithms

In TINA networks, multiple instances of a service component can be placed on different nodes. One result of this is the feasibility of load balancing. If one node suffers from heavy traffic, the other nodes can help this node by sending the signals elsewhere. The main objectives of a load balancing algorithm are to improve the *throughput* and the *setup time* in the network.

Since load balancing has not yet been considered during the TINA development, it is not certain how much a load balancing algorithm might know about the network. Therefore, we have investigated a number of simple load balancing algorithms. We have decided to place the load balancing mechanism in the DPE on each node, since it is only the DPE that has knowledge about the other nodes. The DPE uses the algorithm to find an appropriate SC instance. The algorithm is used only the first time a signal is sent to a particular SC type during the setup sequence. The next time a signal in the same setup sequence is sent to this SC, the same SC instance is used.

The algorithms use measurements to decide the current load status of the nodes. These measurements are performed during time intervals, called control intervals, of length τ seconds. At the end of each control interval, the load status is updated. The algorithms are described below.

5.1 Random

In this algorithm, an SC is chosen randomly, with the same probability for each node that contains instances of the particular SC type. This algorithm is of course very simple to implement in the network, and can be considered as a "default" algorithm.

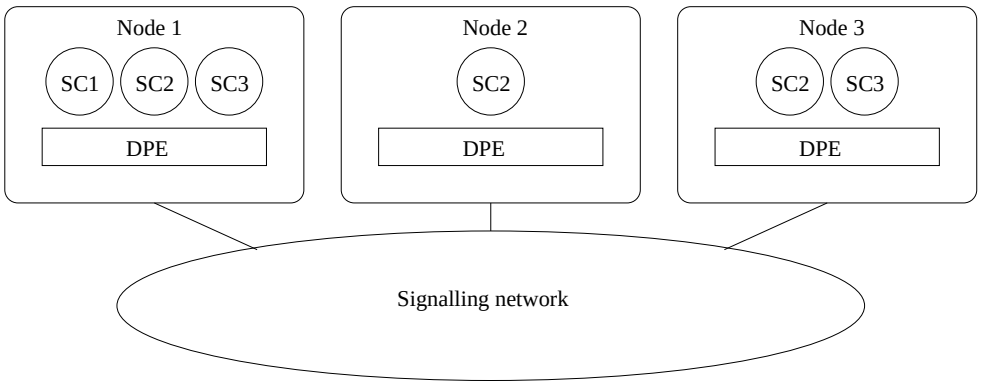

Figure 1: Network with three nodes and three SC types

5.2 Shortest queue

In this algorithm, the DPE selects the SC instance that is placed on the node with the shortest job queue. The shortest queue algorithm can be considered as an optimal algorithm concerning the setup time. However, it would be very difficult to implement this algorithm in a real network, since the DPE will not have this knowledge about other nodes. Therefore, this algorithm is only used for comparison with the other three strategies.

5.3 Acceptance probabilities

In this algorithm, each node uses two metrics: $N_{tot}(i)$ and $N_{rej}(i)$, where $N_{tot}(i)$ stands for the number of signals sent to node i and $N_{rej}(i)$ stands for the number of signals sent to node i that has been rejected. At the end of the control intervals, each node estimates the acceptance probabilities for the other nodes. Let $A(i)$ denote the estimated acceptance probability for node i. $A(i)$ is estimated as

$$A(i) = \frac{N_{tot}(i) - N_{rej}(i)}{N_{tot}}$$

The node chooses an SC instance on node i with the probability $P(i)$. $P(i)$ is given by

$$P(i) = \frac{A(i)}{\sum_{i \in V} A(i)}$$

where V is the set of nodes that contain the particular SC type.

In order to remove the effects due to statistical fluctuations, $P(i)$ is low pass filtered. If $P_k(i)$ is the estimated probability in the kth interval, the low pass filtered probability, $P^*(i)$, is calculated as

$$P^*(i) = \alpha \cdot P_k(i) + (1 - \alpha) \cdot P^*_{k-1}(i)$$

where $P_{k-1} * (i)$ is the low pass filtered probability from the previous interval.

5.4 Load status values

In this algorithm, each node uses a metric $L(i)$, which denotes the load status of node i. The load status is updated at the end of the control intervals. The update is performed as follows. If there have been any rejected messages from node i, $L(i)$ is decreased with one. Otherwise, $L(i)$ is increased with one. $L(i)$ can never be less than L_{min} and more than L_{max}.

The node chooses an SC instance on node i with the probability $P(i)$. $P(i)$ is given by

$$P(i) = \frac{L(i)}{\sum_{i \in V} L(i)}$$

where V is the set of nodes that contain the particular SC type.

6 Investigations

The load balancing algorithms were examined in a number of simulation cases. The main objective of the investigations was to find better load balancing algorithms than the Random algorithm. Therefore, all results should be compared with the results for the Random algorithm. The parameters used in the simulations are shown in Table 2.

6.1 Execution times

It is of course very hard to estimate appropriate execution times for the SCs, since there are no TINA systems in operation today. The execution times depend on the amount of work related to each signal. If the signal requires a DPE service, even more execution time is needed. We have decided to use the following execution times. Signals belonging to USER, AGENT, USM and SSM objects have an execution time of 1 ms. Signals belonging to CSM objects require 4 ms, since we assume that the CSM perform more complex tasks than the previous objects. Further, signals belonging to SF objects require an execution time of 2 ms. If the SC uses a DPE service, the execution time of this signal is multiplied with five. Further, the transmission times are modelled as extra execution times in the sending and receiving nodes. This extra execution time is 0.25 ms both for the sending and receiving node.

	Value
Number of nodes, K	10
Control interval length, t	1 second
Parameter in control scheme, step	0.05
Parameter in low pass filter, a	0.2
Minimum load status value, L_{min}	1
Maximum load status value, L_{max}	30

Table 2 Parameter settings

6.2 Service component distributions

Since there can be several instances of each SC in a TINA network, the instances must be distributed among the nodes in some way. We used two SC distributions, one

balanced distribution in which the load was evenly shared among the nodes and one focused distribution in which node 6 received more load than the other nodes.

If nodes containing the USER component become overloaded, calls will be rejected before they enter the network. This means that they are rejected before the load balancing can have any effect. Therefore, nodes 1-5 only contain the USER component. Table 3 shows the SC types on each node.

	USER	AGENT	USM	SSM	CSM	SF
Balanced	1-5	6-8	6-8	9 and 10	9 and 10	9 and 10
Focused	1-5	6-8	6-8	6 and 9	6 and 10	6 and 10

Table 3 Service component distributions

6.3 Traffic profiles

Two basic traffic profiles were used in the simulations. The first profile is *Low* traffic, in which the arrival rate is so low that no nodes are overloaded in the network.This means that all calls that arrive at the network are accepted. The second profile is *High* traffic, in which the arrival rate is high enough to cause overload in one or several nodes. This means that some calls will be rejected.

In the balanced distribution, the load is evenly shared among the nodes. This means that a network using this distribution can have a higher arrival rate before it becomes overloaded than a network using the focused distribution. Therefore, the actual arrival rates for Low and High traffic depend on the SC distribution. The arrival rates are shown in Table 4.

	Balanced distribution		Focused distribution	
Traffic profile	Low	High	Low	High
Arrival rate	29	67	25	60

Table 4 Arrival rates (calls/second)

6.4 Simulation cases

The load balancing algorithms described in Section 5 were examined for all combinations of algorithms and traffic profiles. Each case was simulated with both the balanced and focused SC distributions.

7 Results and discussion

This section contains the results from the simulations. Since the confidence intervals are small for all results, we have decided not to shown these. The resulting throughputs and setup times must be considered together, since a low throughput usually results in a low setup time for the calls that are finished without being rejected. Therefore, an algorithm with a low setup time is not always the best algorithm.

7.1 Results with the balanced SC distribution

The results for the balanced SC distribution are shown in Table 5. During Low traffic, all cases have 100% throughput. However, the Shortest queue algorithm has the lowest setup times. During High traffic, the algorithm with Acceptance probabilities and the

Random algorithm behave similarly. The Shortest queue algorithm has low setup times, though the throughput is lower as well.

	Low Traffic		High Traffic	
	Throughput	Setup time (s)	Throughput	Setup time (s)
Random	100%	0.13	75%	0.31
Shortest queue	100%	0.11	70%	0.20
Acc. prob.	100%	0.13	75%	0.31
Load status	100%	0.12	70%	0.48

Table 5: Results for the balanced SC distribution

During High traffic, the algorithm with Load status values behaves strangely. The throughput is rather low, which usually result in shorter setup times as well. However, the setup times are much higher than for the other algorithms. The reason for this is that, compared to the other algorithms, the calls are rejected later in the setup phase. This results in more wasted capacity which means lower throughput and longer setup times for the calls that are finished. This means that the algorithm with load status values cannot be considered good when the SC distribution is balanced, since it behave worse than the Random algorithm.

7.2 Results with the focused SC distribution

The results for the focused SC distribution are shown in Table 6. Here, the difference between the algorithm with Acceptance probabilities and the algorithm with Load status values is seen more clearly. In the case with High traffic, the node with a high load will get a small load status value in the other nodes. The nodes with a low load will get high load status values, since they never reject any messages. Thereby, these nodes will receive relatively more traffic and this helps the node with high load (node 6). In the algorithm with Acceptance probabilities, the acceptance probabilities are compared for each node. Since, node 6 has about 90% acceptance probability, and the nodes have about 100% acceptance probability, there is not much difference, which means that the traffic will be distributed rather evenly among the nodes. Therefore, the algorithm with Acceptance probabilities and the Random algorithm have a similar behaviour during high traffic. During low traffic, all three algorithm behave similarly. This because the nodes reject very few messages, which means that the algorithms will distribute the traffic evenly among the nodes.

Further, the Shortest queue algorithm minimises the setup time, however it cannot maintain a high throughput in the case with high traffic. This is probably due to the fact that it is not certain that a node with a short queue has a low load. During low traffic the Shortest queue algorithm is definitely the best algorithm since it has both the highest throughput and the lowest setup time.

	Low Traffic		High Traffic	
	Throughput	Setup time (s)	Throughput	Setup time (s)
Random	97%	0.14	45%	0.26
Shortest queue	100%	0.11	67%	0.18
Acc. prob.	98%	0.14	49%	0.25
Load status	98%	0.14	76%	0.30

Table 6: Results for the focused SC distribution q

8 Alternative load balancing algorithms

As can be seen in the results it is feasible to improve the network performance in TINA by introducing simple load balancing algorithm. The algorithms we have suggested need no extra communication between the nodes, since they only consider the rejected messages from other nodes. However, one problem with these types of algorithms is that the nodes have no complete knowledge about the traffic situation (see, for example Kihl and Nyberg [13]). If for example, a node has not sent any messages to a specific node for a long time, it has no knowledge about the load situation on that specific node. Therefore, it is very difficult to develop a load balancing algorithm that behave well for all load situations if we assume that the nodes cannot exchange any load status information.

However, the main problem with these simple algorithms is that they only work when there is overload in the network. If there is a normal load situation in the network, that is no overload, there are no rejected messages to use in the load balancing algorithms. This means that the traffic is distributed evenly among the nodes. However, also during normal load situations it would be feasible to improve the setup times by sending relatively more traffic to lightly loaded nodes. This requires that the nodes exchange load information, which means that it is necessary to add the feasibility of load status communication to the protocols used in TINA. One simple way to implement this is to use a load status field in all messages. This is already implemented in the Automatic Congestion Control (ACC) used in telephone networks (see ITU-T [16]). Here, the node adds its load status to all messages it sends to the other nodes. In ACC this load status is defined to be zero, one or two, where zero means that there is no overload and two means that it is severe overload. Of course, it is better to use more load status values, for example 0-10 (Northcote and Rumsewicz [12]). The other nodes could use this load status value to update the parameters in a load balancing algorithm.

9 Conclusions

In TINA, instances of the same service component can be placed on different physical nodes. Therefore, the network performance can be improved by introducing load balancing algorithms. These algorithms should distribute the traffic among the SC instances in such way that the overall throughput and setup time are improved.

We have examined four simple load balancing algorithms. The first algorithm randomly distributes the traffic, with the same probability for all instances. The second one sends a call to the node with the shortest job buffer. The third and fourth algorithms use the number of rejected messages to decide an appropriate traffic distribution. All algorithms except the second one can easily be implemented in a TINA network, since they do not require any extra communication between the nodes.

The results show that it is difficult to find a simple algorithm that behave well for all traffic situations. The main problem is that the algorithms have not enough information about the load situation on the different nodes, since no load information is exchanged between the nodes. This problem can be solved by adding the feasibility of load status information to the TINA protocols.

References

[1] M. Chapman and S. Montesi, *Overall Concepts and Principles of TINA*, TINA Consortium, 1995.

[2] P. Graubmann, W. Hwang, M. Kudela, K. MacKinnon, N. Mercouroff and N. Watanabe, *Engineering Modelling Concepts (DPE Architecture)*, TINA Consortium, December 1994

[3] R. Minetti and E. Utsunomiya, *The TINA Service Architecture*, Proceedings of the TINA Workshop at TINA'96 Conference, Heidelberg, Germany, 1996

[4] R. Minerva, *TINA Service Architecture: some Issues in Service Control*, Proceedings of the TINA'95 Conference, Melbourne, Australia, 1995.

[5] L. Kristiansen, *TINA Service Architecture*, version 5.0, TINA Consortium, 1997.

[6] O. Kremien and J. Kramer, *Methodical Analysis of Adaptive Load Sharing Algorithms*, IEEE Trans. on Parallel and Distributed Systems, Vol. 3, No. 6, Nov. 1992.

[7] T. Kunz, *The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme*, IEEE Trans. on Software Engineering, Vol.17, No.7, July 1991.

[8] S. Jordan and P. Varaiya, *Control of Multiple Service, Multiple Resource Communication Networks*, Proceedings of Infocom'91, Bal Harbour, Florida, USA, 1991.

[9] S. Ramakrishnan, I. Cho and L.A. Dunning, *A Close Look at Task Assignment in Distributed Systems*, Proceedings of Infocom'91, Bal Harbour, Florida, 1991.

[10] A. Parhar and M.P Rumsewicz, *A Preliminary Investigation of Performance Issues Associated with Freephone Service in a TINA consistent network*, Proceedings of the TINA'95 Conference, Melbourne, Australia, 1995.

[11] D.J. Houck, K.S Meier-Hellstern, F. Saheban and R.A. Skoog, *Failure and Congestion Propagation Through Signalling Controls*, Proceedings of the 14th International Teletraffic Congress, Juan-les- Pines, France, 1994.

[12] B. Northcote and M. Rumsewicz, *An Investigation of Tandem Overload Control Issues*, Proceedings of the International Conference on Communications, Seattle, USA, 1995.

[13] M. Kihl and C. Nyberg, *Investigation of overload control algorithms for SCPs in the Intelligent Network*, IEE Proceedings, Vol. 144, No. 6, Dec 1997, pages 419-424.

[14] M. Kihl, C. Nyberg, H. Warne and P. Wollinger, *Performance Simulation of a TINA Network*, Proceedings of Globecom'97, Phoenix, Arizona, USA, 1997.

[15] M. Kihl, *On Overload Control in TINA Networks*, Proceedings of the 6th IEE Conference on Telecommunications, Edinburgh, United Kingdom, 1998.

[16] ITU-T, Recommendation Q.723 *Specifications of Signalling System Number 7.*

[17] ITU-T, Recommendation X.900, *Basic Reference Model for Open Distributed Processing (RM-ODP)*, 1993.

[18] N. Widell, *STINA - Performance Simulation of TINA Networks*, Master Thesis, Dep. of Communication Systems, Lund Institute of Technology, Sweden, 1998.

Simulation of a Distributed CORBA-based SCP

Conor McArdle[1], Niklas Widell[2], Christian Nyberg[2], Erik Lilja[2], Jenny Nyström[2], Thomas Curran[1]

[1] Dublin City University, Glasnevin, Dublin 9, Ireland mcardlec@teltec.dcu.ie
[2] Department of Communication Systems, Lund Institute of Technology, P.O. Box 118, SE-221 00, Lund, Sweden
niklasw@tts.lth.se cn@tts.lth.se

Abstract

This paper examines load balancing issues relating to a distributed CORBA-based Service Control Point. Two types of load balancing strategies are explored through simulation studies: (i) a novel ant-based load balancing algorithm, which has been devised specifically for this type of system. This algorithm is compared to more traditional algorithms, (ii) a method for optimal distribution of the computational objects composing the service programs. This is based on mathematically minimising the expected communication flows between network nodes and message-level processing costs. The simulation model has been based on the recently adopted OMG IN/CORBA Interworking specification and the TINA Service Session computational object model.

1 Introduction

There is increasing interest in the use of object-oriented Distributed Processing Environments (DPE) as the infrastructure for new telecommunications service platforms as they promise the benefits of more flexible service design and service deployment, increased software reuse and increased interconnection capabilities with external resources such as the Internet and private databases. One such technology, the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA), has already gained acceptance in the industry for use in network management applications and there have been recent standardisation initiatives for its introduction into the Intelligent Network (IN) [3]. One of the first evolutionary steps towards the introduction of CORBA to the IN has been seen as the replacement of the Service Control Point (SCP) with a CORBA-based distributed system [6]. This approach allows investment in most of the legacy IN infrastructure to be preserved while bringing to bear the advantages of CORBA.

Although CORBA promises many technological and business advantages for this type of application, there are important performance concerns that need to be addressed so that distributed CORBA-based systems can provide the real-time performance and reliability characteristics that are required of telecommunications systems. Sharing of load between the nodes of the distributed system is one important issue that can greatly impact on overall performance and reliability. This paper examines

this issue, taking into account both processing load due to service related tasks and processing load due to inter-node communications, which can be quite significant in CORBA-based systems. Two solutions to load sharing in this type of environment are examined. (i) The choice of location of service objects on nodes in the network can greatly effect performance. An optimal method is used, which minimises total processing costs. Although location of service objects provides a basis for combating loading problems, it is static i.e. determined at design time based on expected service demands and it does not account for queuing and stochastic effects in the network. (ii) More dynamic methods are required to cope with variations in the arrival rates of service requests. This paper presents an ant-based load-sharing algorithm along with several simpler algorithms that are used for benchmarking purposes. Both solutions have been incorporated into a CORBA-based SCP simulation model and results are presented in this paper.

Section 2 of this paper introduces the recent developments in the area of CORBA in the Intelligent Network. Section 3 describes the simulation model, which is based on the IN/CORBA Gateway and the TINA Service Session components and has been implemented using MIL3's Opnet Modeler. Three test services, Virtual Private Network, Ringback and Call Forwarding have been simulated. In Section 4, the performance issues relating to these types of networks are discussed. Section 5 presents an optimal service object placement method. Section 6 describes the load sharing algorithms that have been simulated and Section 7 presents simulation results.

2 CORBA-Based IN

Much of the investigation into the application of CORBA to IN systems has been initiated by the Eurescom P508 project [4], the goal of which was to determine the options for evolving from legacy systems towards TINA. A major result of the project was that the gradual introduction of a TINA DPE (i.e. CORBA technology enhanced with real-time capabilities) into the existing IN environment represents a fundamental prerequisite for such an evolution. During the course of the P508 project, White Papers [1] and [2] were produced and submitted to the OMG in order to support the then emerging activities on IN/CORBA interworking. These White Papers were targeted at providers of information technology solutions and had the purpose of stimulating their interest towards telecommunication operator specific needs. They analyse a specific element of the problem area: the introduction of CORBA into the Intelligent Network. The central idea put forward is to adopt the OMG CORBA standard, enhancing it to make it suitable for telecommunications systems, particularly IN. Subsequently, the work was continued within the Telecommunications Domain Task Force of the OMG, which has recently produced a standard [3]. This standard focuses on the interworking of CORBA-based systems with TC-User applications, such as traditional Intelligent Network and mobile systems.

The primary goal of the IN/CORBA Interworking standard is to provide interworking mappings and supporting CORBA services that enable traditional IN systems (such as a Service Switching Point (SSP)) to interwork with CORBA-based implementations of IN systems (such as a CORBA-based Service Control Point (SCP). In order to do this, the standardised interworking mappings produce IDL for a CORBA object model that provides interfaces to legacy IN systems from the CORBA domain and also provides interfaces to CORBA-based IN applications from legacy IN systems. In effect, this object model may be used to build a gateway that provides protocol conversion

and alignment of execution semantics between the IN and CORBA domains, allowing CORBA-based services to be introduced to the IN.

Figure 1 below shows the main IDL interfaces defined by the standard and how they interact to provide an interworking function (gateway) between the IN and CORBA domains. A more complete description of the standard may be found in [5].

A legacy SSP interacts with a CORBA-based SCP using the IN/CORBA object model defined in the Interworking specification. The objects shown in grey are CORBA objects whose interfaces are defined in OMG IDL in accordance with standard. The Gateway Administration object (GWAdmin) performs the functions of name translation and object location between the two domains. Messages arriving from a legacy SSP are addressed to a particular SS.7 Application Entity (AE), identified by a particular AE title. The GWAdmin provides an interface for translating the AE title to the CORBA object reference of a Service Interface Factory object, which may create instances of the appropriate Service Interface Object. This Service Interface Object acts as a proxy for the CORBA-based SCF. In order to represent the SSP in the CORBA domain, a SSF Proxy object is required. This object provides an IDL interface for invocation of INAP operations on the SSF from the CORBA domain and performs the protocol translation and communication with the SS.7 stack. The SSF Proxy Factory provides a standardised means of instantiating a SSF Proxy. The Service Interface Object provides a complementary IDL interface for invocation of INAP operations from the SSF to the CORBA-based SCP. Protocol translation for these invocations is provided in the gateway.

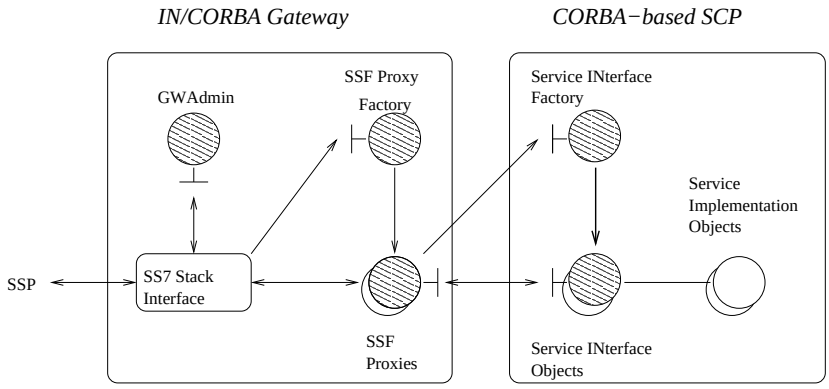

Figure 1: Main elements of the IN/CORBA Interworking Gateway

The Service Implementation Objects are not defined by the specification and may be implemented by some arbitrary set of fine-grained CORBA objects, which provide the functionality required for service execution. One such model based on the TINA Service Session, which forms the basis for the simulation studies, is detailed in the next section.

2.1 TINA-based Computational Objects

With this approach, the IN service logic and data, residing on the CORBA-based SCP, are modeled as a subset of the computational objects composing the TINA Service Architecture. An approach given in [6] is adopted, which defines methods

for modeling IN services executing in a TINA environment. Here it is assumed that all calls originate and terminate on the IN side so that neither the calling nor called party uses a TINA end-system and thus, is not modeled as a TINA user. This is appropriate for the CORBA-based SCP scenario as all SSPs resides in the IN domain and these are the only originators of calls. As a result, the IN service capabilities may be encapsulated entirely within the TINA Service Session COs. That is, the TINA Access Session is ignored and the COs that provide this functionality are not required. All calls are established through the IN Service Switching Function (SSF) under the supervision of the TINA Service Session Manager (SSM).

With this approach, the service capabilities are modeled within a User Application (UAP), interacting with an Service Session Manager (SSM), which makes use of a service specific IN Service Support Object (SSO), e.g. a database containing number translation tables. As there is no call-party specific access session, the User Agent (UA) is anonymous and acts on behalf of all IN users. The Provider Agent (PA) is also generic in this case. Figure 2 below shows the COs required for an implementation of a Virtual Private Network (VPN) service.

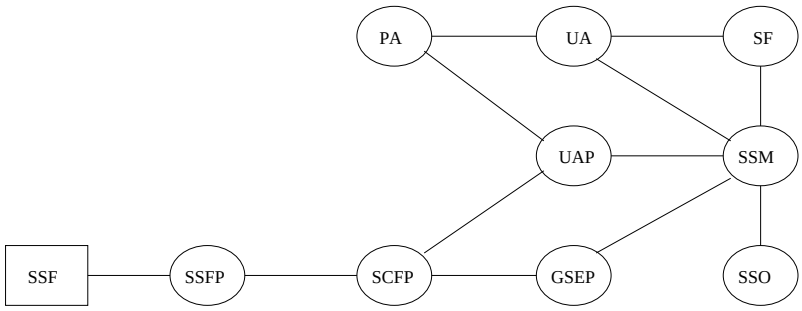

Figure 2: Computational Objects and the interactions required for a VPN Service

Generally, for any service session, on receipt of the initial service request from the SSF, the SSF Proxy (SSFP) passes the initial call to the UAP via the SCF Proxy (SCFP), which in turn initiates a corresponding TINA service session via the PA. The PA interacts with a UA in order to perform a generic access session for service session establishment. Once the SSM has been created and initialised by the Service Factory (SF), a control relationship is established between the IN SSF and the TINA SSM. The interactions between components are thence dependent on the specific service in execution.

Note that the IN/CORBA Interworking is modeled by considering only the core objects necessary for communication between the IN and CORBA domains during a service session i.e. the Proxy objects. The SSF Proxy object accepts INAP operations from the SSF over SS7 and translates them to CORBA invocations on the SCF Proxy. The SCF Proxy accepts INAP IDL invocations from the UAP and GSEP, transfers them to the SSF Proxy object which translates them to the corresponding INAP operations on the SSF. Proxies for the Intelligent Peripheral (IP) will also exist if required by the service. IP Proxies act in an identical manner to SSF Proxies.

3 CORBA-Based SCP Simulation Model

This section provides an overview of the distributed CORBA-based SCP model which has been developed to provide the basis for simulation studies of likely performance bottlenecks and for study of suitable strategies for load balancing in this type of environment.

In this scenario, the Intelligent Network Service Control Function (SCF) and Service Data Function (SDF) are no longer encapsulated within single functional entities but are decomposed into fine-grained Computational Objects (COs) which use the CORBA Object Request Broker (ORB) for communication. These objects communicate with entities in the legacy Intelligent Network via the IN/CORBA Gateway. Thus, the service logic programs and data that normally reside at the SCP and SDP are distributed across a multi-node network. Figure 3 shows the general network configuration of the CORBA-based SCP scenario and how it interconnects to a legacy Intelligent Network.

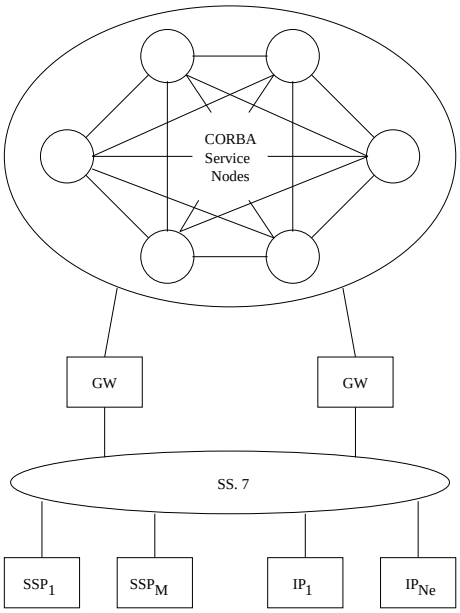

Figure 3: Network Scenario for CORBA-based SCP

The scenario under study consists of a network of ten CORBA Service Nodes and two IN/CORBA Gateway Nodes. Gateway Nodes may communicate with all CORBA Service Nodes and all Service Nodes may communicate with each other i.e. the Gateway Nodes and Service Nodes form a fully connected network which is connected to the SS.7 Network at the Gateway Nodes. The network connection scenario in the CORBA domain is intended to represent machines interconnected over a local area network. The number of Service Nodes has been chosen so that adequate processing power is available to replace the processing power provided by a legacy SCP. It is assumed that individual Service Nodes have considerably less processing capacity than a legacy SCP and that service execution requires considerably more processing due to distribution. It is assumed that two Gateway nodes are required so that there

is an element of fault tolerance within the system. The Gateway Nodes execute the functionality required for interworking between SSPs, IPs and the CORBA-based SCF, which is a distributed application executing on the CORBA Service Nodes. It is assumed that the Gateway function consists of the standard IN/CORBA interworking components described in Section 2. Thus, each Gateway Node executes CORBA Proxy objects, which provide an interface for invocation of IN operations on the SSP and IP from objects in the CORBA domain. The Gateway Nodes also execute the functionality that translates incoming messages from the legacy IN entities to CORBA invocations on SCF Proxies, which reside in the CORBA Service Node network. All other COs equired to complete service execution reside on the CORBA Service Nodes. The legacy IN entities (SCP and IP) and the SS7 network are not modeled explicitly but are viewed as the source and sink of messages arriving to and departing from the Gateway Nodes.

3.1 Execution Model

The processing time for a service is decomposed into processing times for the set of messages passed between COs that are required to complete service execution. Each message passed between two COs has associated with it a CORBA marshaling (protocol encoding) time on the client-side node, a CORBA demarshaling (protocol decoding) time on the server-side node plus a processing time for completion of some service specific task on both the client and server side nodes.

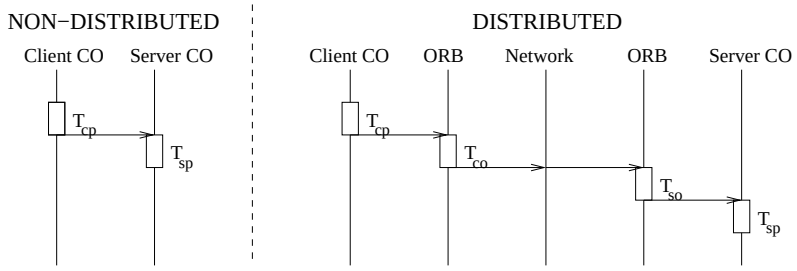

Figure 4: Execution times for messages passed between COs. In the right hand figure, two COs are executing on different processors (i.e. COs are distributed). The processing times T_{CP} (service processing time) and T_{CO} (marshalling time) give the total processing time at the client node associated with this message. Similarly, T_{SO} (demarshalling time) and T_{SP} give the total processing time at the server node. In the left hand figure both COs are executing on the same processor so the total processing time is given by T_{CP} and T_{SP}.

Marshalling, demarshalling and processing times remain constant for a particular message over all sessions of a service. If the communicating COs are located on the same node, the marshalling and demarshalling times are not included in the overall processing time for the message as CORBA is not required for communication.

The marshalling and demarshalling times used for simulation are based on times measured on a commercial ORB (Visibroker 3.3 running on a Sparc Ultra 5). The IDL used for determining timing measurements is based on the IN/CORBA specification and the TINA Ret Reference Point specification so that each message has associated with it the appropriate marshaling and demarshalling times. Processing times for

actual service related tasks are based on the processing times for the service executing on a legacy SCP.

An asynchronous invocation mechanism is assumed, such as the CORBA Messaging Service. Thus, a CO making a CORBA method call does not block the process while waiting for a response from the server side. As a result, it is also assumed that all CORBA objects on a node execute in a single thread of execution and that the servers are modeled as a single FIFO job queue.

It is assumed that delays in network transmitter queues and transmission times on the network are negligible compared to delays due to marshalling and demarshalling of CORBA method calls between nodes. Experiments have shown that marshalling and demarshalling times for the IDL used for this model are typically an order of magnitude greater than transmission times over IP on a fast network, such as 100Mbps Ethernet. It is also assumed that the order of messages is preserved in the network and that there is no message loss.

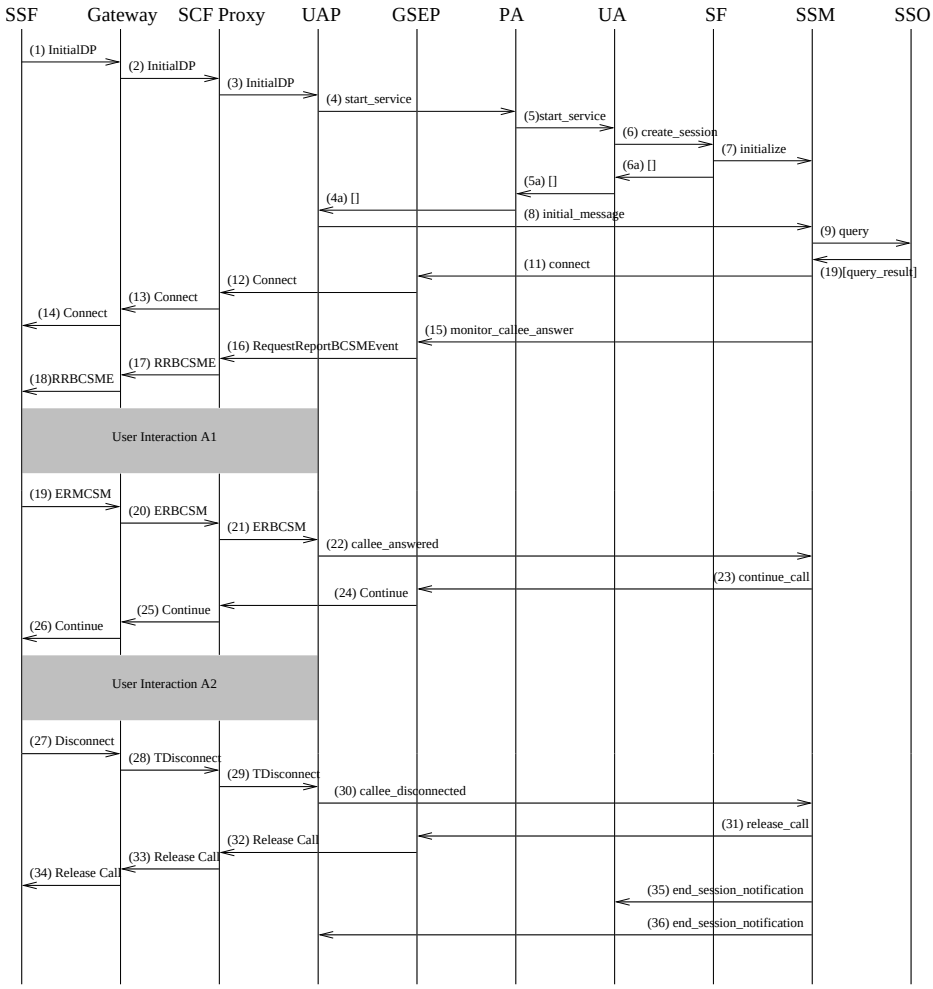

Figure 5: Message Sequence Chart for Test Service A

3.2 Test Services

Three different test services have been chosen to execute on the CORBA-based SCP in order to study the performance issues:

- Service A Virtual Private Network
- Service B Ringback
- Service C Restricted Access Call Forwarding

The COs required for Service A and their intercommunications are shown in Figure 5 (above) Service B and Service C have been similarly defined. The duration of User Interaction A1 (phone ringing) is drawn separately for each service session from a negative exponential distribution with a mean of 5 seconds. The duration of User Interaction A2 (Conversation period) is drawn separately for each service session from a negative exponential distribution with a mean of 100 seconds. It is assumed that service users never abandon ongoing service sessions and thus the messaging for handling these cases does not need to be defined.

4 Performance Issues

One of the most important benefits of distributed computing is the ability to split computational tasks among multiple processors. However, the distribution of software objects across different physical nodes can cause severe performance problems such as synchronisation of memory or databases, as well as much inter-object communication between nodes, which creates a large computational overhead. Thus, it is important to be aware of how the distribution of software objects affects performance. This section first describes some areas where performance issues are likely to appear, then discusses three related areas that can be taken into account to reduce these problems.

4.1 Problems

The performance of a telecommunications system is important for several reasons. Firstly, the users of the system have several expectations, such as fast system response times and reliability. Secondly, the network operators require their systems to operate as efficiently and cost effectively as possible.

One major factor in the real time performance of a CORBA-based system is the processing overhead caused by inter-node communication. Typically the transmission times of a message are very low when compared to the time required by protocol wrapping and unwrapping (marshalling and unmarshalling in CORBA) at the sending and receiving node. Thus, too much, or simply inefficient distribution of objects on physical nodes can easily cause overload due to protocol overhead alone.

The characteristics of traffic within the CORBA-based SCP are different from the traffic in normal non-telecom CORBA-applications. The real-time characteristic of teletraffic is more "now or never" than normal traffic which has more stricter rules for finishing a task than finishing it within a given time frame. The "now or never" property of teletraffic allows us to block incoming calls if they would result in degraded performance for the already accepted ones.

4.2 Problem Solutions

Solving the performance problem requires several different techniques and strategies. Below is a list of three different areas where large performance gains are likely to be found.

1. Object distribution: This concerns the placement of objects on different nodes. Different configurations can cause very different communication patterns and, as communication is very computationally expensive, we want to make sure that we have no more communication than is necessary. While object distribution can be static, meaning that objects stay where they were placed at design time, it is possible to move objects around during run time and thus dynamic distribution schemes are possible. One must note here, however, that since moving objects around is a computationally intensive operation it is not likely to be feasible solution to solve short-term traffic transients.

2. Load Balancing: When a object distribution has been decided upon, there must be some kind of mechanism that directs the object remote procedure calls if there are multiple nodes that offer the same object type. It is the purpose of the load sharing mechanism to direct these procedure calls so that they, if possible, keep some nodes from being overloaded while others are almost idle. Load sharing is difficult since the object and node which we chose might have favorable conditions at the decision time, but these conditions can change quickly and we must therefore take some account of future load as well.

3. Load control: When load sharing is not enough to handle the offered amount of traffic, then some kind of load control mechanism is necessary. Load control is not usually used in distributed systems, since the traffic must get through if the application is to work. As it was said earlier, this is not necessarily true in a telecommunications environment, where it is more important to finish some work in time rather than to finish all work long overdue.

This paper is concerned with solutions 1 and 2. Future work will explore solution 3.

4.3 Measuring Performance

Measuring performance in this type of environment can be difficult due to the very flexible nature of the application. Below are a number of possible considerations which have been used in the past for evaluating the effectiveness of performance controls:

1. *Throughput*: A traditional comparison, throughput measures the number of finished service requests per time unit.

2. *Scalability*: Scalability is very important for an algorithm, since we want our algorithms to work for any network size.

3. *Transient survival*: Due to the high reliability requirements on telecommunication networks, the algorithms must be able to quickly adapt to changing conditions such as traffic peaks or node failures, which might cause rapidly changing traffic patterns.

4. *Algorithm Complexity*: Since the system is very complex to start with, we want the algorithms to be simple and easily implemented. Also, simple algorithms tend to be fast and have little overhead.

In this paper we have chosen to use two simple measurements for evaluation of our performance controls:

1. *Mean Service Completion Time*: This is an important factor for a realtime system. It also gives an indication of the level of queueing delays in the network.

2. *Maximum Load*: This is the mean load on the highest loaded node in the network and gives an indication of the effectiveness of the load balance amongst network nodes.

5 Optimised Computational Object Distribution

The choice of location for service objects on nodes in the network can greatly effect performance. If objects requiring large amounts of processing are clustered on a small number of nodes then queues lengths increase on these nodes or worst, an overload can occur. Conversely, if objects are distributed too much then large amounts of unnecessary protocol processing is incurred which lengthens the service time. This section describes an optimal method that allows processing capacity in the network to used efficiently under normal loading conditions. The problem is stated below.

With the exception of the Proxies, there is no restriction on assignment of COs to CORBA Service Nodes. The SCF and IP Proxies reside only on the two Gateway Nodes, as these objects need to communicate directly with the SS.7 stack. It is assumed that an instance of a service, initiated through an SCF Proxy on a particular node, may use only SCF and IP proxies on that same node for the duration of the service execution. All other COs may be duplicated across many nodes. However, COs are assumed to be atomic i.e. may not be decomposed and distributed between nodes. The assignment of these COs to network nodes is determined by minimising the total processing cost on all nodes given the expected user demands from the two Gateway nodes for each service and given the maximum load allowed on each of the Gateway and CORBA Service nodes. This approach is similar to that found in [7] where the communications costs between COs are to be minimised.

The problem may be formulated as a Linear Programming problem in the form given in Equation (1). The variable for minimisation (vector x) denotes the processing costs associated with the total traffic flow during a service session between each communicating component pair, relative to the processing cost due to the traffic offered to the SSF Proxy CO by the SSF for that service. Note that it is assumed in the formulation of problem that copies of all COs reside on all nodes. If in the solution, the costs associated with a particular CO copy on a particular node are all zero, then that CO need not exist on the node.

In the objective function, C is simply the unit matrix as it is required that all processing costs in the network are minimised and that all costs are of equal importance. This form of the objective function determines that the problem is linear.

The A matrix and b vector in the constraint inequality are determined by: (i) the number of units of input traffic load offered by each SSF, for each service. These are equality constraints; (ii) the relative processing costs between associated component

pairs. These are also equality constraints with each constraint expressing the processing cost associated with a component pair relative to the processing cost associated with one other component pair. An adequate number of constraints are required to associate all components in the service graphs. The relative processing costs are derived by summing the processing times for all messages passed between each pair of COs during a service session. When both COs reside on the same node the costs express the sum of all client and server service processing times for interactions between these COs. When the COs reside on different nodes then the costs also include ORB processing costs for both client and server; (iii) the limit on processing capacity for each node. These are inequality constraints that limit the sum of all costs associated with a node. These constraints may be set to give a component distribution that is optimal at a particular operating point, for example, to give a maximum of 40% loading on all the nodes.

$$\begin{aligned} minimize \quad & C^T x \\ s.t. \quad & Ax \leq b \\ & l \leq x \leq u \end{aligned} \tag{1}$$

The bounding inequality is defined to constrain x to be positive. There is no upper bound on x as the limiting factor is the total processing cost associated with each node, which is expressed as part of the constraints ((iii) above).

The solution to this minimisation determines the optimal placement of COs in the network. That is, if the costs associated with a particular copy of a CO on a node are all zero, then the CO is removed from that node. Having removed all such null COs, the remaining COs are optimally placed in the network. The solution also determines the relative flows between each CO and copies of the COs with which it communicates. That is, the routing probabilities for requests between COs are determined by the solution. These routing probabilities may be used as the basis for a load balancing algorithm which aims to minimise overall network load. Such an algorithm is presented in Section 6.

6 Load Balancing Algorithms

In more tightly coupled systems, generally, two approaches to load balancing have been taken: an idle processor may request more work from other processors or a busy processor may send excess work to idle processors. These approaches do not work very well in CORBA-based distributed systems since it costs too much in terms of protocol processing to move jobs. In this type of systems, load balancing can be done when an instance of a service component exists at more than one node. The load balancing algorithm is required to choose the most suitable node on which the component shall be executed, given certain data relating to the current state of the network nodes.

The main purpose of our work is to investigate so called *ant* algorithms, where mobile agents are used to find the most suitable node, in terms of low load, on which to execute the required component. To allow evaluation of this strategy, two simple benchmark algorithms, described below, have also been implemented. In order to describe the operation of these algorithms, the following notation is introduced. Let R be the set of all nodes which contains service component r. If the next service component needed is r we have to choose one of the nodes in R where r will be executed.

6.1 Benchmark Algorithms

The benchmark algorithms have been chosen to allow the lower bounds (or close to the lower bounds) of the performance measures to be established. These algorithms are not intended to be viable as a practical solution to the load balancing problem but allow the ant based algorithms to be evaluated against theoretically near-optimal solutions. The benchmark algorithms are as follows:

1. *Shortest queue*: the node in R with the shortest processor queue is chosen. If nodes with the same queue lengths are found, the lowest numbered node is chosen. We assume that all nodes have instantaneous knowledge of the queue lengths in the nodes in R. This assumption obviously renders the algorithm impractical. However, the results are expected to give close to the lower bound of the Service Completion Time as queuing delays at nodes are maintained at a low level.

2. *Random*: one node is chosen randomly in R. The probability for choosing a particular node is derived from the static routing scheme determined by optimisation (see Section 5). This assumption renders the algorithm impractical as it does not respond to transients and drifts in service mix. However, assuming constant service mix, the algorithm it expected to maintain loading at a near-optimal level.

6.2 Ant Algorithms

Ants are simple agents that are sent out by the nodes to probe the load status of all nodes in the system. An ant is sent away from a node (called the sending node) to another node (called the receiving node) and then returns to the sending node. The sending and receiving node may be the same node. In our investigations we have compared three load status parameters: queue length, load of the receiving node and the roundtrip time. The load of a node is measured as the proportion of an interval the server is busy. The length of these intervals is 0.1 seconds. In our model the ants are handled like this:

1. First the ant is created in the sender node. It is put back in the high priority processor queue of the sender and after queuing it is wrapped into a CORBA protocol and sent to the receiving node.

2. At the receiving node the ant is queued and its CORBA protocols are unwrapped. After this it is queued once more and the queue length or load is put into the ant. Then the ant is queued once more after which it is wrapped into the CORBA protocols again and sent back to the sending node.

3. When the ant has returned to the sender it is queued in a high priority queue, its CORBA protocols are unwrapped, it is queued once more after which its load status information is stored in a load status table as described below.

We assume that each node in the network keeps a load status table with information about all nodes in the network. When an ant returns to the sender, the load status for the receiving node is updated in the table.

The values used can be either the receiving node's queue length, its load or the roundtrip time of the ant. For all these measures we have that the higher the load

status value, the fever calls should be sent to the node. The load status table is used in this way to chose a node in R: Assume that the load status of node i is $l(i)$. Observe that $l(i)$ may be queue length, load or roundtrip delays. Then we calculate probabilities for all nodes in R as follows:

$$p(i) = \frac{l(i)^{-k}}{\sum_{j \in R} l(i)^{-k}}$$

With probability $p(i)$ node i is chosen. In this way there is a larger probability of sending a call to a node with a low load status value. The k in the calculation is the factor that describes the randomness of the weights. $k = 0$ is the case where every node is chosen with equal probabilities, larger k's give larger weights to values on the limits.

Ants are generated in a node according to a Poisson process with rate λ. The generation rate is an important parameter. If it is too low, the values in the load table will not be updated fast enough which could lead to oscillations. If it is high, the ants themselves will increase the load of the system. When an ant has been generated, it must be decided where to send it. We have chosen the following algorithm: with probability α the receiving node is chosen randomly with equal probability for all nodes, with probability $1 - \alpha$ the node with the lowest value in the load status table is chosen, i.e. the node with lowest load. Thus we have two parameters λ and α that must be tuned.

7 Simulation Results and Analysis

Several simulations were run using the model defined in Section 3. Both the benchmark algorithms and the three ant-based algorithms were simulated. The simulation parameters are listed in Table 1 and the simulation results are presented in Table 2.

As indicated in Table 1, all algorithms were simulated under a low loading and high loading scenario. The service mix is balanced in each case, i.e. the mean arrival rates for all three services are equal in each case. These arrival rates were chosen to give an average load of about 30% in the low load case and about 80% in the high load case. The services executing in the simulation are Virtual Private Network (Service A), Ringback (Service B) and Restricted Access Call Forwarding (Service C), described in Section 3.2. The distribution of service objects was obtained using the method described in Section 5. This distribution is optimised for the given arrival rates. The ant spawning intervals for the ant-based algorithm were tuned for each variation and are given in Table 1.

	Value
Arrival Rate Low Load	$225s^{-1}$
Arrival Rate High Load	$675s^{-1}$
Object distribution	Optimised for balanced service mix
Service Types	Services A, B and C
Service Mix	Balanced
Ant Spawning Interval (Load Query)	$0.05s^{-1}$
Ant Spawning Interval (Round Trip)	$0.01s^{-1}$
Ant Spawning Interval (Shortest Queue)	$0.015s^{-1}$

Table 1. Simulation Parameters

		Benchmark		**Ant-Based**		
		Random	*Shortest Queue*	*Load Query*	*Round Trip*	*Shortest Queue*
Low Load	*Service Time*	12.4ms	11.4ms	12.4ms	13.0ms	12.9ms
	Max Load	25.8%	55.3%	25.6%	30.4%	28.9%
High Load	*Service Time*	35.2ms	25.1ms	34.6ms	40.9ms	40.5ms
	Max Load	78.7%	87.0%	74.5%	81.7%	81.5%

Table. 2. Simulation Results

As indicated in Table 2, The *Mean Service Completion Time* was measured for each algorithm and averaged over all three services. Note that the User Interaction times, indicated in Section 3.2, are excluded from this measurement as they are independent of loading in the network. The *Max Load* value indicated is the load on the most heavily loaded node in the network.

Considering the performance of the benchmark algorithms, as expected the *Shortest Queue* algorithm performs best overall in terms of minimising the service time. This is due to the fact that queue sizes are kept as short as possible. This method will not perform as well when there is a large difference in the processing required from message to message, as queue size will not accurately indicate how long an arriving message will need to wait for service. However, for the service simulated, there is not a large difference in message processing and thus the Service Time is close to the minimum and gives a good benchmark. The uneven loading, indicated by the high *Max Load* value for the *Shortest Queue* benchmark algorithm is due to the fact that the node with the lowest number is chosen when several queues have the same length. This condition will occur frequently at low loading levels when there is a significant probability that the queue length is small.

The *Random* benchmark algorithm gives a low *Max Load* value as expected, however, this is not the lowest overall value. This is because of the static nature of the algorithm. The algorithm will give minimum loading on all nodes only at exactly the intended operation point i.e. perfectly balance service mix with deterministic arrivals. To avoid this problem, future work will consider combining this algorithm with a more dynamic algorithm. The service times observed for the *Random* benchmark algorithm are increased compared to the *Shortest Queue* benchmark due the constraint on node loading. Because the optimisation is constrained to maintaining load below a particular level on all nodes, service execution is more distributed and thus service times increase due to increased protocol processing costs. Future work will consider "softening" this constraint to allow load to be less balanced but so that shorter overall service execution times can be obtained.

The simulation results show that the *Ant-Based* algorithms compare quite favorably with the benchmark results. In the low load condition, the service times are comparable to the service time for the *Shortest Queue* benchmark. The *Max Load* for the *Ant-Based* algorithms is also close to the value for the *Random* benchmark for low loading. For high loading, the service times increase considerably compared to the *Shortest Queue* benchmark. Further work is required to refine the Ant-based algorithms. In the results presented here, the weighting factor (k) has been set to 1. Results are required to investigate the performance with higher weighting factors and to investigate tuning of the ant spawning interval.

Overall, further work is required in a number of areas. The results presented here were generated under stable network conditions. In order to fully assess the algorithms,

the behavior under transient traffic conditions needs to be studied. Overload protect has not been considered here and needs to be investigated and incorporated into the algorithms. The algorithm computational complex and robustness also needs some consideration.

8 Conclusions

This paper has presented a number of approaches for improving the performance of a distributed CORBA-based Service Control Point. Although distributed systems technologies can contribute greatly to this area by allowing processing requirements to be divided among a large number of less expensive processors, it is unwise to assume that increasing processing power or memory sizes of network processors ad infinitum will alone guarantee high performance. The solutions offered in this paper aim to increase the efficiency and cost effectiveness of resources with a view to making CORBA-based solutions more suitable for high performance, reliable systems required by telecommunications environments.

Acknowledgements

This work has been partially sponsored by the Commission of the European Union under the project MARINER, project number AC333 of the ACTS program. This work has also been partially sponsored by the Swedish Research Council for Engineering Sciences (TFR) under Contract 271-97-203.

References

[1] Object Management Group, *Intelligent Networking with CORBA*, OMG DTC Document:telecom/96-12-02, December, 1996

[2] Object Management Group, *White Paper on CORBA as an Enabling Factor for Migration from IN to TINA: A P508 Perspective*, OMG DTC Document: telecom/97-01-01, January, 1997

[3] Object Management Group, *Interworking between CORBA and TC Systems*, OMG document telecom/98-10-03, August, 1998

[4] EURESCOM Project P508, *Introduction of Distributed Computing Middleware in Intelligent Networks White Paper*, OMG DTC Document: 97-09-01, September, 1997

[5] Nilo Mitra, Rob Brennan, *Design of the CORBA/TC Inter-working Gateway*, Proceedings 6th International Conference on Intelligence in Services and Networks, Han Zuidweg et. al. (eds.), ISBN: 3-540-65895-5 Springer-Verlag, April, 1999

[6] U. Herzog, T. Magedanz: *From IN toward TINA - Potential Migration Steps*, Technology for Cooperative Competition, Fourth International Conference on Intelligence in Services and Networks, Springer Publishers, Cernobbio, Italy, May, 1997

[7] Anagnostou, M.E., *Optimal Distribution of Service Components*, Proceedings, IS&N'98, 5th International Conference on Intelligence in Services and Networks, Antwerp, Belgium, May 1998

[8] Kihl, M., Widell, N., Nyberg, C., *Load balancing strategies for TINA networks*, Proceedings, 16th International Teletraffic Congress, Edinburgh, Scotland, June 1999

Overload Protection for CORBA Systems with Time Constraints

Niklas Widell, Christian Nyberg and Maria Kihl

Department of Communication Systems, Lund University,
P.O. Box 118, SE-221 00 Lund, Sweden
{niklasw,cn,maria}@telecom.lth.se

Abstract

Scalable and reliable distributed object-oriented computing (DOC) middleware systems is an important technology in, for example, telecommunications service logic and distributed web servers. The Common Object Request Broker Architecture (CORBA), developed by the Object Management Group (OMG) is a specification of a common platform for DOC systems. CORBA acts as middleware, by inserting itself between the Operating System (OS) layer and the Application layer on a host. CORBA provides support for transparent interaction of objects situated on different nodes. The original CORBA specifications had no support for timing constraints in applications and very little support in the terms of performance optimizations. Present extension to CORBA include support for real-time applications and a number of performance enhancements such as load balancing. However, no work so far address the issue of overload in a CORBA system. This paper presents a discussion of overload issues in distributed CORBA systems with time-constrained tasks. First a performance model of a CORBA system is introduced. Second, overload in distributed CORBA systems is discussed. Third, a number of classic overload protection mechanisms are applied to the performance model and investigated using simulation. The simulations show that even by using very simple protection mechanism, a good throughput can be achieved.

1 Introduction

Scalable and reliable distributed object-oriented computing systems like CORBA are being used to implement, for example, telecommunications service logic and distributed web servers. As the systems become larger, it becomes more difficult to predict the needs of the applications for dimensioning purposes, in particular since the needs may change during the execution of the application.

Depending on the time constraints put on a system, it can be classified as either real-time or best-effort. A real-time system is a system where the correctness of a task depends not only on the logical result of the task, but also on the time at which the results are produced. A best-effort system is one where the time it takes to complete a task has no impact on the correctness of the execution.

In general, a task in a real-time system has a deadline associated with it. The deadline is the time when the task is supposed to be finished. A real-time system can be further classified into being either *hard* or *soft* real-time. In a hard real-time system all deadlines must be met and the failure to meet a deadline may have disastrous results. A hard real-time system need not be fast, but it must be very

predictable. In the soft real-time case occasional timing violations may occur, as long there is no system failure. A typical performance criteria for a soft real-time system is to minimize the number of missed deadlines. Jensen [1] provides a discussion of soft real-time systems.

Soft real-time systems include systems where full predictability is not possible, or not even necessary. It may be that the timing of the completed task is important, but the overhead introduced by real-time operations and/or complexity of the system may prohibit the use of a full real-time environment. Further, there might be a dynamic request pattern, as arrival times for new tasks is not known.

An e-commerce site may be regarded as a soft real-time system, even though it is not implemented using a real-time infrastructure. The traffic to the site is not likely to be known with great accuracy. This means that the system may become overloaded, causing long response times. If a response from a server takes to long to reach a client it may mean that the client leaves the site without making any purchases, resulting in a loss of revenue for the site. In a market-place where a customer can chose between many different e-commerce sites, poor performance can be disastrous for a site. Also, for an e-commerce site, it is the *user-experienced* performance that matters, and nothing else.

CORBA is a suitable platform for soft real-time applications such as e-commerce systems. In order to improve performance, support for load balancing, migration of objects and distributed scheduling exist in several different implementations of CORBA. An object distribution mechanism decides how objects are distributed on physical nodes, taking into account the possible migration of objects from one node to another. A load balancing mechanism attempts to distribute the workload so that the system performs as efficiently as possible. These facilities allow given systems to work with better efficiency and to approach soft real-time capabilities.

In many cases load balancing and object distribution mechanisms are not sufficient to guarantee the timing requirements of a system. These mechanisms are only sufficient when the workload caused by arriving tasks is well below system capacity. In, for example, web servers and telecommunications service logic, tasks may arrive at so high rate that system capacity is exceeded and the system becomes overloaded. If no overload protection is used in this case, task throughput suffers and overloaded servers cause response times to grow above acceptable levels. The fundamental observation for overload protection is that in some cases it is better to prevent some tasks from entering a system, than to let all tasks enter and thereby let all tasks experience poor QoS.

Several papers have discussed load balancing and load sharing in computer networks. Othman *et al* [2] presents the load balancing scheme used in the TAO real-time implementation of CORBA. The Realize Resource Management system described in Melliar-Smith *et al* [3] describes a full system that takes care of the run time needs of real time distributed applications. Realize supports replication for fault tolerance, migration and dynamic scheduling of objects using an off-the-shelf ORB. Kreimen *et al* [4] discussed load sharing algorithms for distributed systems. Kunz [5] examined how the network nodes should exchange load status information to be used in load balancing schemes.

Overload protection has been studied extensively for telecommunication systems, see for instance Körner *et al* [6] for a survey of early results. Berger [7] compares the efficiency of two classic protection schemes, call gapping and percent thinning. The ACTS project MARINER has published extensively on using market-based agent technology to protect distributed systems in Intelligent Networks, see [9]. Kihl [10]

describes a simple throttle scheme for the Telecommunications Information Networking Architecture (TINA). TINA is an architecture for next generation telecommunication networks, see the TINA Consortias home page [8]. Jordan *et al* [11] investigated call admission policies for communication networks that support several services. In the ACTS project MARINER CORBA-based IN systems were investigated and some results on the use of agents for load balancing in CORBA system can be found in Conor *et al* [12]. Also in [12] is an investigation in how to assign a number of objects to to a number of processors in order to minimize, for example, the mean session completion time.

Other than Kihl [10] and Rumsewicz [13], no research has been performed in the area of overload protection schemes for distributed object oriented systems. The reason for this is probably that overload control has not been deemed necessary for the systems under study, due to the assumption that the offered workload is not above a certain acceptable limit. However, for some systems, typically where the resources are used by "outsiders", such as CORBA based web servers, the overload problem can be very serious and must therefore be studied. While some results from classic telecommunication research can be used, new problems, such as how to identify overload conditions in a distributed system and how to react to overload when it has been identified in a way that still provide good service to the users etc. remains to be investigated.

This paper focuses on the introduction of overload protection in CORBA systems with timing constraints. The main objective is for the system to preserve the user-experienced QoS during short periods of overload. The paper is organized as follows: Section 2 provides background on CORBA. Section 3 introduces a performance model of a distributed CORBA system. Section 4 discuss the philosophy behind overload protection. Section 5 mentions how overload protection may be implemented in CORBA. Section 6 describes a number of simulation cases using the performance model together with some classic protection mechanisms. Section 7 summarizes the work.

2 CORBA

CORBA is an open standard for distributed object computing under development by the Object Management Group [16]. By the use of standards, CORBA aims to provide independence from programming language, computing platform and communication medium.

A CORBA platform provides many necessary functions for distributed processing, such as object registration, location and activation, parameter passing among others. CORBA defines a set of mechanisms that allow a client object to invoke a method on a server object on a remote node with full location transparency. Location transparency means that neither the client object nor the server object needs to know anything about on which node they are executing, since the CORBA infrastructure hides the distribution from the application.

An object in CORBA is an encapsulated entity with a distinct identity whose services can only be accessed through well defined interfaces. Thus an object's services and how the services are implemented is separated. The interfaces of an object are specified in the *Interface Description Language, IDL*. The IDL is then used to generate stubs and skeletons in a some programming language, such as C++ or Java. The stubs are used by a client object to make method invocations to a server object. The server uses the skeleton to implement the service.

The CORBA Object Request Broker (ORB) acts as a message bus to provide seamless interaction between client and server objects. Using the General Inter-ORB Protocol (GIOP) and the TCP/IP specific Internet Inter-ORB Protocol (IIOP) heterogeneous ORBS can interoperate.

In the original CORBA specifications there were no support for real-time applications. Real-time CORBA 1.0 [17] adds support for static scheduling of tasks. The proposed Real-time CORBA 2.0 specification will include support for "pluggable" dynamic schedulers.

2.1 CORBA performance improvements

A number of features exists in CORBA that improves the performance of applications. These include, for example, object distribution, object migration and load balancing.

The process of distributing objects on multiple nodes is fundamental to CORBA. However, due to the extra processing time necessary to make method invocations between nodes, the distribution of objects can have large impact on performance. A good distribution will improve performance, while a poor distribution will decrease performance.

A CORBA application with a given object distribution can be re-configured to another object distribution during run-time using object migration. However, object migration is a complex and computationally expensive operation and its use under conditions of short term overload remains to be investigated. Object migration makes it possible to change the system if the conditions for the system change. Such changes include new task types being added, new nodes being added or if the traffic pattern of arriving tasks changes.

To make distribution efficient, support for load balancing mechanism is required. Load balancing allows the system to spread the load from arriving tasks to objects on different nodes so that no node has excess spare capacity while another node in the system is overloaded. In CORBA, this means that given a set of distributed objects to chose from, the mechanism should chose an object residing on the node with the least load.

The current load balancing mechanism in CORBA is simple (see Othman *et al* [2]. The mechanism works by sending all requests to one node until that node is overloaded, then the mechanism decides which other applicable node has the least load and sends all further requests there, until that node is overloaded.

3 Performance Model

This section presents a performance model of a distributed CORBA system to aid in the discussion of overload protection.

Consider a system containing a network with m nodes. Nodes are labeled N_1 to N_m. As an example, figure 1 shows a simple five node (N_1 to N_5) network. To allow for an inhomogeneous network, node i has a relative processing speed s_i. If $s_i = 1, 1 \leq i \leq m$, then all nodes have equal processing speed. For the present model, it is assumed that it is the processors that are the performance bottlenecks. In addition, assume that the network is very fast compared to the nodes, and switching and transmission times are negligible. Therefore the nodes can be considered as fully connected. Each node is modeled as a single server queue. The queuing discipline depends on the scheduling mechanism used. However, for simple analysis a FIFO queue is used.

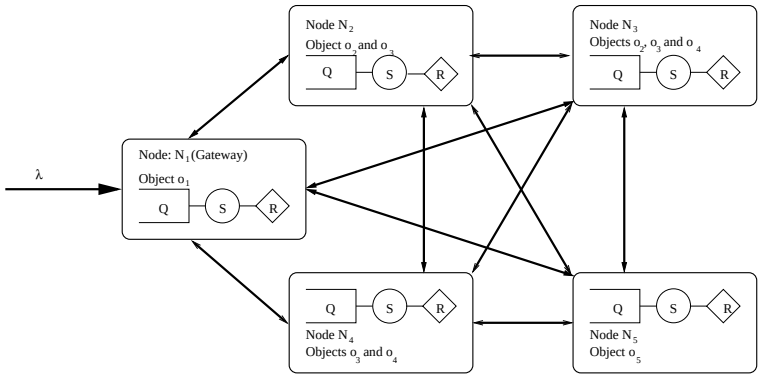

Figure 1: A five node network with five objects

Distributed in the network are software components called object of n different classes. In the performance model, the set of objects all instantiated from a single class i is labeled o_i. Objects in o_i each has a number of services called methods. By saying that o_i is located on node N_j, it is meant that the services of class i can be accessed at run-time at node N_J. Referring to figure 3, there are five objects sets (o_1 to o_5) distributed in the network.

For every method m of object o_i invocation time $t_{o_i,m}$ is used to model the time a method invocation executes before returning or making a new invocation to another object. This is a simplification, since the actual method invocation times may vary depending on the application. It is assumed that the behavior of each method invocation is independent of context, meaning that processing time and other resource usage does not depend on why or when the method was invoked.

The process of translating the parameters transfered during a method invocation between two objects is called marshalling for the client object and unmarshalling for the server object. If a method invocation must be transfered over the network, the marshalling/unmarshalling process is modeled as an extra service time t_m for the node with the client and on server object node side an unmarshalling time t_u. If both client and server objects are located on the same node, then $t_m = t_u = 0$.

The node at which a new task arrives is called a gateway. There can be as many gateways as there are nodes. The gateway represents the entry point for a task. All communication between system and user passes through the gateway.

Tasks of different types, T_i, arrive at the system according to some stochastic process, with a a mean rate λ_i . A task T_i is described by: a sequence of method invocations (SMI_i), a deadline d_i and a importance v_i. SMI_i describes which objects are used and in what order they are used by the task. d_i is the time the task must be finished for it to be considered as useful. v_i is the value of the task to the system, as defined by the implementer. A mission-critical task will have $v_i = \infty$. The value of a task may be used during overload in order to reject tasks with lesser value to give more valued tasks more resources.

A sample task is:

$T_{example} = ([o_1, o_2, o_1, o_3, o_4, o_3, o_1, o_3, o_5, o_3, o_1, o_2, o_1], t_{now} + 0.5sec, 1)$.

To clarify the interaction between objects, figure 2 presents a Message Sequence Chart (MSC) of $T_{example}$:

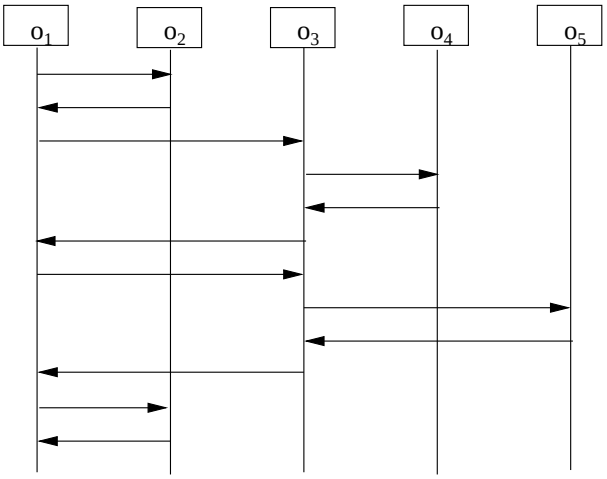

Figure 2: $\tau_{example} = [o_1, o_2, o_1, o_3, o_4, o_3, o_1, o_3, o_5, o_3, o_1, o_2, o_1]$

4 Overload Protection Philosophy

This section outlines why overload is dangerous to a real-time system and how overload protection can be used to improve the performance of such a system.

Fundamental queuing theory states that the load, ρ, of a system is related to arrival rate λ and the task mean service time of $\overline{x}$ as $\rho = \lambda\overline{x}$ [14]. For a single server queue the delay[1] experienced by a task is proportional to $\frac{1}{1-\rho}$. From this relationship it is obvious that keeping load levels down is crucial to prevent large delays. Since a real-time system depends on short and predictable response times to meet deadlines, loads close to one may be catastrophic for the system.

One main observation for overload protection is that in some cases it is better to prevent some tasks from entering a system, than to let all tasks enter and thereby let all tasks experience poor QoS. The objective of an overload protection mechanism is to maximize the amount of useful work which is performed by the system. Useful work is such that is performed to tasks that finish correctly before their deadline. Work done and resources used by tasks that miss their deadlines is wasted. Note that in a soft real-time system deadline misses are acceptable if they are rare, but not if they are common.

The traditional way in which performance has been viewed is from the system operator viewpoint, for instance a telecommunications company. The operator of a system wants maximum capital gain from a given investment, in order to fully utilize the system's available resources. This means that the operator should invest in only enough equipment to support the services offered, while at the same time have enough resources so that no business opportunities are lost due to insufficient resources.

However, with a telecom market moving towards increasing deregulation, and an Internet market where "all sites are equal", the system user viewpoint is becoming more and more important. A user wants quick error-free responses from the system. Good performance is when a requested service is delivered without unacceptable delays.

[1]The given relationship is for the Poissonian arrival process, however, for non-deterministic arrival processes the delay typically has a similar behavior

4.1 Types of Overload

The causes for overload can generally be divided into two classes: *resource failure* and *unbounded traffic source*. By resource failure overload it is meant overload that is caused by the system itself, such as by hardware or software malfunction. The failure of a processor, for example, may decrease the capacity so that the offered workload can no longer be finished in time. The failure of a piece of software may cause system instabilities by generating large amounts of erroneous traffic. Unbounded traffic source overload is caused by external sources, such as the users of a system. This type of overload is common in systems where the system itself has no control of the source of the tasks.

While the overload protection schemes discussed in this paper may be applied to both resource failure and unbounded traffic source overload, it is generally aimed at overload caused by the later.

It important to note that overload protection is meant to decrease the problems caused by short periods of overload. If overload continues, then the system is under-dimensioned and other actions are necessary to improve performance.

4.2 Overload detection

The detection of overload is a process full of potential pitfalls. The system load may change very quickly and the system itself may change due to the introduction of new task types with new objects, new traffic patterns and so on. The overload protection mechanism must both be ready for instant action, while at the same time being conservative in order to avoid oscillations in order to prevent mechanism instability. In a distributed environment it becomes even harder, since overload on one node not necessarily has any impact on the processing of tasks that do not use the overloaded node. In addition, there is a trade-off between transferring load information often, with resulting communication overhead, or relying on old and possibly inaccurate load information.

There are several system parameters that can be monitored during execution. The following are commonly monitored: resource load, deadline miss rate and system response times. In addition, a profiling system that can make fine-grained measurements on execution times can be used.

The resource load can be the load level of a processor, network element, memory or disk. The resource load is used as a basis for load balancing and object migration mechanisms. However, its applicability as a basis for overload detection in a CORBA environment is hampered by a number of factors. First, overload on node N_i may not have any impact on the processing of tasks that do not use N_i. Second, there is a question of *which* measured load that should be used to make the decision, since in a heterogeneous system the same measured load on two nodes with different capabilities may result in very large differences in delay. Third, since the tasks arriving at the system may have widely different service requirements, the variance of the service time may have large impact on the mean service time of a particular node. Finally, if the objective is to satisfy timing constraints, the process of relating a set of measured loads on a number of nodes to a particular task with its given processing requirements may be non-trivial.

The objective of an overload protection mechanism is to maintain a good throughput of tasks that meet their timing constraints. Thus, seeing how well this objective is met by the system is useful for two reasons. First, as a way to discover overload.

Second, as a measurement of how successful the overload protection mechanism is. An increase in the number of missed deadlines would be a good measure that the system is experiencing overload. The ratio of missed deadlines is commonly used in real-time systems as a measurement of success. Lu *et al* has a discussion of Deadline Miss Ratio as overload indicator, see [18]. The main problem with looking at dead-line miss ratio is that it only shows how many deadline misses there are, but gives little warning in the non-overloaded case as to how close the system is to being overloaded. It may therefore fail to adjust to overload until it is to late. To lessen this issue, the mechanism may look at the actual response times of the system and compare these with their respective deadlines.

The response time of the system is a result of load, but also of the variance of the execution times in the system. The main advantage of using system response time instead of load is that the former is what the user of the system sees. For example, a user of an e-commerce site has no knowledge of what the site looks like, how load is distributed, how many nodes are used etc. All that matters for the user is that the *received* QoS is acceptable to the user.

The Realize system [3] includes a Object Profiling System for real-time CORBA, that allows the ORB to monitor execution times of methods, and then feed back this information to the scheduler. The profiling service builds detailed models of how tasks use objects and how long time methods take to invoke.

4.3 Overload protection mechanisms

The protective actions to be taken in the event of overload depends on what type of resource that is to be protected and what type of traffic that arrives at the resource. In traditional telecommunications, the most common way to prevent overload from happening is to reject arriving calls according to some algorithm. However, this may not be sufficient in more complex systems.

In the case of resource failure overload, just rejecting tasks is usually counterproductive. Instead, the nodes should try to act to conserve resources by limiting the number of tasks sent.

In the case of unbounded traffic source, depending on the application and how overloaded the system is, the actions may take many forms. Users can be notified that the system is under heavy load and asked to return later (voluntary back-off), offered a simpler service (service adaption) or in the worst case, be rejected. If possible, the mechanism should be such that it discourages reattempts in a short time-scale. If, for example, users make immediate reattempts after failed tasks, the traffic to the system goes up and thus further increasing the load on the system.

Overload protection mechanisms are often designed to operate in three states. First, when there is no overload and the system can finish all arriving tasks without problems, all tasks receive full service. Second, when there is some overload, the system should try to focus on getting as many of the most valuable tasks finished in time, while potentially rejecting tasks with lower priorities. In this state the objective of the overload protection mechanism is to make sure that the throughput of useful tasks remain high. Third, when there is catastrophic overload, the system should concentrate on its own survival, executing only the most valuable tasks and being very restrictive on what tasks that may enter the system.

The point at which a task is rejected during execution may have great impact on performance. Since the objective of overload protection is to maximize useful work, it is important to reject a task before it wastes to much resources. In a distributed

system, this means that, if possible, a task should be rejected at the gateway to the system.

Also it is important that the overload protection mechanism requires low overhead. It must not require too extensive computational resources, making itself a reason for overload.

To measure the success of an overload protection mechanism, tasks can be divided into four classes.

1. Tasks that are rejected at the gateway, that is before they enter the system. This is often called external or network level overload protection.

2. Tasks that are accepted at the gateway but are rejected by a node inside the distributed system. This is often called internal or node level overload protection.

3. Tasks that are not rejected at the gateway or at any other node but are not finished in time, i.e. tasks that fail to meet their deadline.

4. Tasks that are successfully served and finished in time.

The purpose of overload protection is to maximize the number of tasks belonging to class 4. Naturally, the number of tasks of class 2 and 3 should be as small as possible since they are not successful and the processing time spent on them is wasted.

Tasks of class 1 and 2 can be considered as failed operations that must be taken care of by the application or user generating the tasks. Note that there is a risk that if a fault-tolerant system is being used, the system itself may do reattempts, and this interaction must be addressed.

5 Implementation in CORBA

This section gives some proposals for the introduction of overload protection in CORBA. The overload protection mechanism can then complement load balancing and object distribution mechanisms as ways to improve performance.

An overload protection function may be divided into two main mechanisms: a set of rules that decides which tasks to reject and a mechanism to clean up the system after a task has been rejected.

The set of rules for task rejection are application specific. Tasks may be admitted independently. For example, simple IN/CORBA services may generate only a single task. Some systems require entire groups of tasks to be successful in order to consider a user's interaction with the system as successful. For example, user interaction session with an e-commerce site requires all tasks to be successful for the session to be considered successful.

The general mechanisms to support the release of resources after a task has been rejected can be implemented by using existing CORBA services. The information of a task rejection can be distributed using exceptions. The Real-time CORBA specification specifies a TASK_CANCELLED exception that can be used. It is obvious that the amount of work necessary to release resources is proportional to the amount of resources used, and also the types of resources used. Stateless objects can be destroyed right away, while objects with states must be handled with more care.

6 Simulations

This section describes some classic overload protection mechanisms, a simple feedback loop to make the mechanisms dynamic, and a number of simulation cases that show the efficiency of the mechanisms.

6.1 Common overload protection mechanisms

There are two basic forms of overload protection mechanisms that have been used over the years in telecommunications: window based and rate based. These are further described below, with their respective strengths and weaknesses mentioned briefly.

6.1.1 Window based overload protection

A window based overload protection mechanism operates by limiting the number of active tasks to the window size W. If a new task arrives and there are presently fewer tasks executing than W, it immediately continues executing, otherwise it is rejected. The window size is related to the capacity of the system, with one window size giving one system response time.

The window mechanism is robust, since it handles transients well and uses resources efficiently. For the mechanism to work well, the counter of the number of active tasks must be kept updated at all times, for instance by using time-outs to discover failed operations.

The window mechanism used in this paper is similar to the Isarithmic mechanism described by Gerla and Kleinrock [15].

6.1.2 Rate based overload protection

A rate based overload protection scheme works by limiting the traffic to the system. The rate-based mechanisms are more susceptible to transients in arrival rates than the window based ones, and control loops updating the algorithm parameters must run on a tighter time scale.

Some variations include (both ar further described in Berger [7]):

Percent blocking The percent blocking algorithm admits tasks into the system with a probability $P(admit)$.

Call gapping A call gapping algorithm closes for a set amount of time t_{gap} (the gap size). After this interval the next task to arrive is allowed into the system and the throttle is closed again. By varying the gap size the mechanism can set an upper limit to the number of arriving tasks per time unit.

6.2 Algorithm control loop

A protection mechanism must be able to react to the changes in arrival rate or task mix in a quick, efficient and stable way. The reaction is typically a change in the parameters of the algorithm.

There are many ways in which algorithm parameters can be updated, using information fed back from the system. In this paper a simple update loop, based on the measured task execution time is used. The assumption is that it is the response time

of the system that is important from the user viewpoint. The loads of the internal nodes is of no concern to the user.

The algorithm classifies the response times t_{task} for tasks into three categories: good, fair or bad.

The algorithm uses the following parameters: an interval length t, a user defined deadline $t_{deadline}$ to categorize finishing tasks, an increase condition c_i, a decrease condition c_d, counters for the task categories x_{good}, x_{fair} and x_{bad}, measured task system time t_{task}, update results variable R. c_i is the ratio of tasks classified as good required for the mechanism to allow more tasks into the system. c_d is the equivalent ratio of tasks classified as bad required to decrease the number of tasks allowed into the system.

1. Time is divided into intervals of length t seconds.

2. During interval N, x_{good}, x_{fair} and x_{bad} are updated for each finishing tasks as follows:

Measured t_{task}	Updated parameter
$0 \leq t_{task} \leq 0.5 \cdot t_{deadline}$	$x_{good} := x_{good} + 1$
$0.5 \cdot t_{deadline} \leq t_{task} \leq t_{deadline}$	$x_{fair} := x_{fair} + 1$
$t_{deadline} < t_{task}$	$x_{bad} := x_{bad} + 1$

3. At end of interval N, calculate parameters for interval $N + 1$. Set $R = 0$.

   (a) If $\frac{x_{good}}{x_{good}+x_{fair}+x_{bad}} > c_i$ then $R := R + 1$.

   (b) If $\frac{x_{bad}}{x_{good}+x_{fair}+x_{bad}} > c_d$ then $R := R - 1$.

   (c) Update algorithm parameter according to table (note that $R = 0$ if both (a) and (b) are true above):

Parameter	$R = 1$	$R = 0$	$R = -1$
$P_{N+1}(admit)$	$P_N(admit) + 0.01$	$P_N(admit)$	$P_N(admit) - 0.01$
$t_{gap,N+1}$	$t_{gap,N} \cdot 0.95$	$t_{gap,N}$	$t_{gap,N} \cdot 1.05$
W_{N+1}	$W_N + 1$	W_N	$W_N - 1$

   In all cases there are max and min values that the parameters cannot go beyond.

6.3 Simulation parameters

The sample system is a model of a five node network ($N_1 - N_5$) with six object sets $o_1 - o_6$. Two different tasks, T_1 and T_2 arrive at the gateway node each according to a Poisson process with rates λ_1 and λ_2 respectively.

The tasks are defined by:

$T_1 = ([o_1, o_2, o_1, o_3, o_6, o_3, o_1], t_{arr} + 0.5sec, 1)$

$T_2 = ([o_1, o_2, o_1, o_3, o_4, o_3, o_1, o_3, o_5, o_3, o_1], t_{arr} + 0.5sec, 1)$

where t_{arr} is the arrival time of the task.

Table 1 summarizes other simulation parameters used.

	Value
Marshalling time, t_m	1ms
Unmarshalling time,t_u	1ms
Increase condition, c_i	0.50
Decrease condition, c_d	0.05
Length of update interval	1s

Table 1 Parameter settings

Table 2 below contains object distribution and respective probabilities used for load balancing. A dash means that this type of object is not available on this node. The object distribution is assumed to be static, with no object migration or further replication taking place. A random load balancing scheme is used, with the probabilities given in the table.

For example, a task of type T_1 is executing in the system, and an object in o_1 is about to make a method invocation on an object in o_3 for the first time of the task. The random load balancing scheme with the probabilities shown distributes invocations to object set o_3 evenly on nodes N_2, N_3 and N_4. Say that N_2 is chosen. Then all further invocations to o_3 will also be sent to N_2.

Object	N_1	N_2	N_3	N_4	N_5	$t_{o,m}$
o_1	1.0	—	—	—	—	$1ms$
o_2	—	0.50	0.50	—	—	$4ms$
o_3	—	0.33	0.33	0.34	—	$2ms$
o_4	—	—	0.5	0.5	—	$8ms$
o_5	—	—	—	—	1.0	$10ms$
o_6	—	—	—	1.0	—	$3ms$

Table 2: Object distribution and probabilities used for load balancing.

6.4 Simulation cases

Three performance perspectives were investigated. First the impact of the simple overload protection mechanisms on the throughput was compared to the case when no protection was used. Second, the algorithms were tested for fairness. Fairness was considered to be how available resources were used when the mix of tasks arriving was changed. Third, the system's reaction to traffic transients was investigated.

In all cases a Poissonian input process was used.

6.4.1 Case A: Throughput

The throughput of successful tasks was investigated. Successful tasks are tasks that fall into category 4 as described in section 4.3 above. The throughput of successful tasks is what is usually called *Goodput* [7].

75% of the tasks were of type 1 and 25% of the tasks were of type 2.

6.4.2 Case B: Fairness

In this case the arrival rate λ was kept constant, but the task mix was changed. This meant that the load on the system changed, since the two task types had different processing requirements.

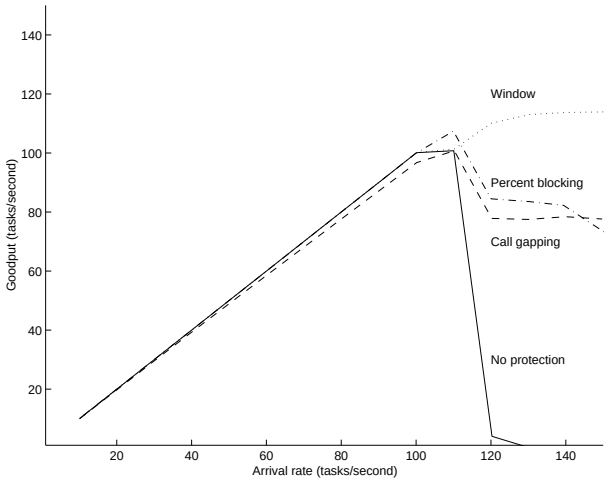

Figure 3: Throughput of succesful tasks ("Goodput") as function of arrival rate.

The objective of the investigation was to see whether the overload protection mechanism had any impact on the mix of tasks leaving the system.

Arrival rate was kept constant at $\lambda = 120s^{-1}$. The task mix was varied from 0% to 100% of task 1 and the rest of task type 2.

6.4.3 Case C: Traffic transient reaction

In this case the mean response time for the system was investigated when a traffic transient increase appeared. The task mix was kept constant, with 75% of the tasks were of type 1 and 25% of the tasks were of type 2. A period of 600 seconds were simulated. For the first 200 seconds, the arrival rate was kept at $\lambda = 100^{-1}$. At time 200s the arrival rate was changed to $\lambda = 120^{-1}$, which made the system overloaded. At time 400s the arrival rate was changed back to $\lambda = 100^{-1}$.

6.5 Simulation Results

6.5.1 Case A: Throughput

Figure 3 shows the throughput of successful tasks (goodput) as a function of total arrival rate λ. Throughput is severely degraded for the case with no overload protection (solid line), while call gapping (dashed line) and percent blocking (dash-dotted line) are very similar, with call gapping being slightly better than percent blocking at higher rates. The window mechanism (dotted line) gives the best throughput. All three protection algorithms are much better than a system without any protection.

The reason for the window mechanism being more successful is that it is less sensitive to small statistical fluctuations than percent blocking or call gapping. With a given window size, the number of tasks allowed into the system is kept at a level that the system can handle, given its timing constraints.

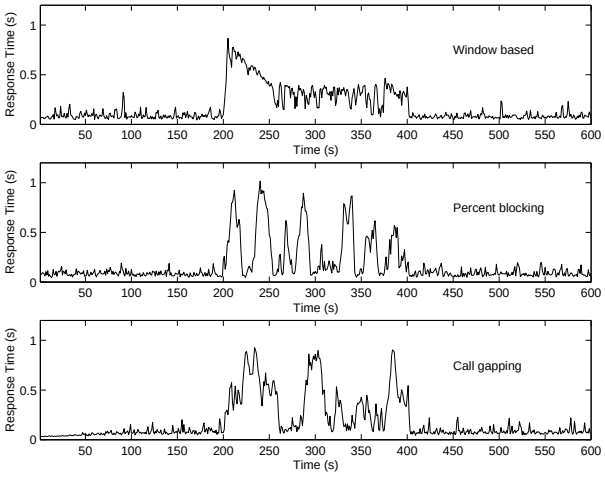

Figure 4: Mean response time for transient case

6.5.2 Case B: Fairness

The results from the simulations were the same for all three mechanisms. The throughput of successful tasks changed, but the ratio of arriving tasks of type 1 and exiting tasks of type 1 was constant.

6.5.3 Case C: Traffic transient reaction

In this case the transient characteristics of the system was investigated. Figure 4 shows the mean response times, measured over 1 second long intervals, for one realization of the simulation. All tasks that finish, whether they meet their deadline or not, are included in the mean response times calculated for each interval.

The diagrams show that the window based mechanism seems to be more stable than percent blocking or call gapping. All mechanisms show a large increase in the beginning of the transient. The time it takes for the mechanism to establish a new steady state depends on how fast the control loop is. In the window based case, the response time decreases as fast as the loop allows it until it ends up on a new mean response time, which is now slightly below the deadline at 0.5 seconds. Both call gapping and percent blocking never attain any new steady state during the transient. A better transient response for call gapping and percent blocking could probably be achieved with better tuned parameters.

7 Conclusions

This paper argues for the introduction of overload protection in CORBA systems with time constraints. The main reason to use overload protection is to prevent complete throughput degradation when the load is high. This can be very important when dimensioning systems in an environment where budgets are limited or where the accuracy of traffic estimates is poor.

Three classic simple overload protection mechanisms are added to a CORBA system in this paper. Simulations show that all three mechanisms prevent a complete throughput degradation that happens when no protection is used. To control the mechanisms a control loop is used. The control loop uses comparison of measured response times and the deadlines of the task to update the parameters of the mechanisms. Using the measured response time is useful for two reasons. First, it is what the user of the system sees, and therefore the experienced quality of the system is used to control the system. Second, CORBA systems can be very complex, with many nodes, many objects and quickly changing traffic patterns, where traditional load measurements are both difficult and error-prone.

Simulations show that a window based mechanism performs better than both percent blocking and call gapping. The window based mechanism has both better throughput at high loads and is less sensitive to sudden traffic increases.

8 Acknowledgements

This work has been partially sponsored by the Swedish Research Council for Engineering Sciences (TFR) under contract 271-97-203.

References

[1] E. D. Jensen,*Eliminiating the 'hard'/'soft' real-time dichotomy*, Computing and Control Engineering Journal, February 1997.

[2] O. Othman, C. O'Ryan and D. C. Schmidt, *The Design of an Adaptive CORBA Load Balancing Service*, Distributed Systems Engineering Journal, April, 2001

[3] P. M. Melliar-Smith, L. E. Moser, V. Kalogeraki and P. Narasimhan, *Realize: Resource Management for Soft Real-time Distributed Systems*, Proceedings for the IEEE Information Survivability Conference, SC, January 2000.

[4] O. Kreimen and J. Kramer, *Methodical analysis of adaptive load sharing algorithms*, IEEE Transactions on parallel and Distributed systems, Vol. 3, No. 6, Nov. 1992.

[5] T. Kunz, *The influence of different workload descriptions on a heuristic load balancing scheme*, IEEE Transactions on Software Engineering, vol. 17, no. 7, July 1991.

[6] U. Krner and C. Nyberg, *Overload Control in Communication Networks*, GLOBECOM '91, Phoenix 1991.

[7] A. W. Berger, *Comparison of Call Gapping and Percent Blocking for Overload Control in Distributed Switching Systems and Telecommunication Networks*, IEEE Transactions on Communications, vol. 39, no.4, April 1991.

[8] TINA Consortia's homepage: www.tinac.com

[9] ACTS Project AC333 MARINER, *Deliverable 9: Project Final Report*, Dublin, 2000.

[10] M. Kihl, *On overload control in a TINA network*, 6th IEE Conference on Telecommunications, Edinburgh, Scotland, 1998.

[11] S. Jordan and P. Varaiya, *Control of multiple service, multiple resource communication networks*, Proceedings of Infocom'91, Bal Harbour, Florida, 1991.

[12] C. McArdle, N. Widell, C. Nyberg, E. Lilja, J. Nystrm and T. Curran, *Simulation of a Distributed CORBA-based SCP*, IS&N 2000, Athens, Greece, 2000.

[13] M. Rumsewicz, *Load Control and Load Sharing for Heterogeneous Distributed Systems*, Proceedings of ITC 16, Edinburgh, Scotland, 1999.

[14] L. Kleinrock, *Queuing Systems Volume 1: Theory*, Wiley-Interscience 1975.

[15] M. Gerla and L. Kleinrock, *Flow Control: A Comparative Survey*, IEEE Transactions on Communications, VOl. 28, No. 4:553-574, 1980.

[16] Object Management Group's homepage: www.omg.org

[17] Object Management Group, *Real-time CORBA 1.0*, 1998.

[18] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. Son and M. Marley, *Performance Specifications and Metrics for Adaptive Real-Time Systems*, Proceedings of the 21st IEEE Real-Time Systems Symposium, 2000.

Performance Modeling of Distributed E-commerce Sites

Maria Kihl, Niklas Widell and Christian Nyberg

[2] Department of Communication Systems, Lund Institute of Technology,
P.O. Box 118, SE-221 00, Lund, Sweden
{maria,niklasw,cn}niklasw@tts.lth.se cn@tts.lth.se

Keywords:
E-commerce, performance analysis, queuing model, load control

Abstract

Today, e-commerce is a growing market in the Internet. Many of the problems that have been recognized for e-commerce sites are related to site performance and there is a tendency that customers leave and never come back to sites that perform poorly. In order to design and implement e-commerce sites that will meet the customers' expectations, it is important to have good performance models. In this paper we develop a queuing network model with several customer classes and load control mechanisms. Further, we show how this model can be used to investigate the performance of a site.

1 Introduction

E-commerce is becoming an important part of the Internet. Some examples of e-commerce services are web stores, e-traders, web auctions and Internet banks. Today, e-commerce sites usually consist of multiple web servers distributed on a LAN. By distributing the functionality, the web servers become scalable and more reliable. To manage the system, a so called dispatcher is used to spread the incoming requests according to some load balancing algorithm.

The main objective for the e-commerce providers is of course to achieve high revenue. A customer that connects to an e-commerce site starts a session that will last some time before the customer is finished. The session consists of several requests for varying data. If the customer successfully completes the session, the site owner may receive some revenue. During the whole session the customer expects short response times when requesting new data. If the response times become too long, the customer will end the session which means that the site looses potential profit.

One of the problems that has to be solved is how to ensure that customers receive good QoS all the time. A popular site may receive much traffic during some periods. For a web store this may happen during sales or when a new product is released. Internet banks usually have a high offered load at the end of each month. Therefore, it is crucial to develop analytic models aimed at investigating the performance of distributed e-commerce sites. The objective should be to, for given system parameters, model the site performance during varying arrival rates so that the site can be accurately designed. Further, the models should include load control mechanism in order to ensure a high performance also during overload situations.

Numerous papers have discussed and investigated design issues for distributed web servers. Dias *et al.* [10] were the first to describe a scalable web server that consisted of a set of logical front-end servers and a set of logical back-end servers connected by a switch. A good survey of scalable web server techniques can be found in Schroeder *et al.* [19]. Colajanni *et al.* [6][9] developed different load balancing algorithms for web sites. Pai *et al.* [16] investigated a locality-aware request distribution strategy (LARD) to balance the load among a number of back-end servers. A survey of different load balancing techniques can be found in Bryhni *et al.* [5].

Several papers have developed and investigated mathematical models of web servers. Iyengar [11] analyzed a web server and found that it is necessary to have some admission control in order to obtain good performance. Bhatti and Friedrich [3] discussed QoS issues for web servers and developed a queuing model that gave varying priorities to different classes of customers. The model was used to investigate a simple admission control. Cherkasova and Phaal [8] developed a session based admission control, that rejected new customers during overload. Abdelzaher and Bhatti [1] used a different approach when they developed an overload control that instead of rejecting requests used content adaptation to decrease the load. During overload, the server delivered less resource intensive content to the users.

However, only a few papers have discussed and investigated the specific performance problems that arise in large-scale e-commerce sites. Bhatti *et al.* [4] presents experiments designed to estimate users' tolerance to long waiting times and other QoS parameters in e-commerce sites. Nielsen [15] presents the results from a study of users who have bought products on the Web. Bhargava and Bhargava [2] discussed QoS parameters for e-commerce sites. Meira *et al.* [12] investigated how e-commerce sites can distribute some services to so called e-representatives executed on cache servers.

Until now, only Menascé *et al.* [13][14] have developed queuing network models of distributed e-commerce sites. Their model includes a state transition graph called customer behavior model graph (CBMG) to describe a customer's behavior when visiting an e-commerce site. However, the model does not include any load control mechanisms and the authors do not discuss how to ensure that the customer QoS is preserved also during periods with high offered loads.

In this paper we develop a general queuing network model for distributed e-commerce sites. The model includes several customer classes with varying behavior. Also, the model contains both load balancing and overload control mechanisms. Finally, we show how this model can be used to investigate the performance behavior of an e-commerce site both during normal loads and during overload.

2 A distributed e-commerce site

Figure 1 shows the general structure of a medium to large scale distributed e-commerce site that uses a dispatcher to distribute HTTP requests. A distributed web site can be divided into a front-end system and a back-end system. The front-end system has direct communication with the customer. The front-end then communicates with the back-end, if necessary. The back-end is invisible to the customer.

The front-end part consists of a cache server, a firewall and a dispatcher. The cache server caches static web objects, such as plain HTML pages and pictures. It improves performance by being able to quickly serve the cached content, without forwarding requests to the actual web servers. This means that only requests for dynamic web pages reach the web servers. The firewall protects the system from intrusion. It works

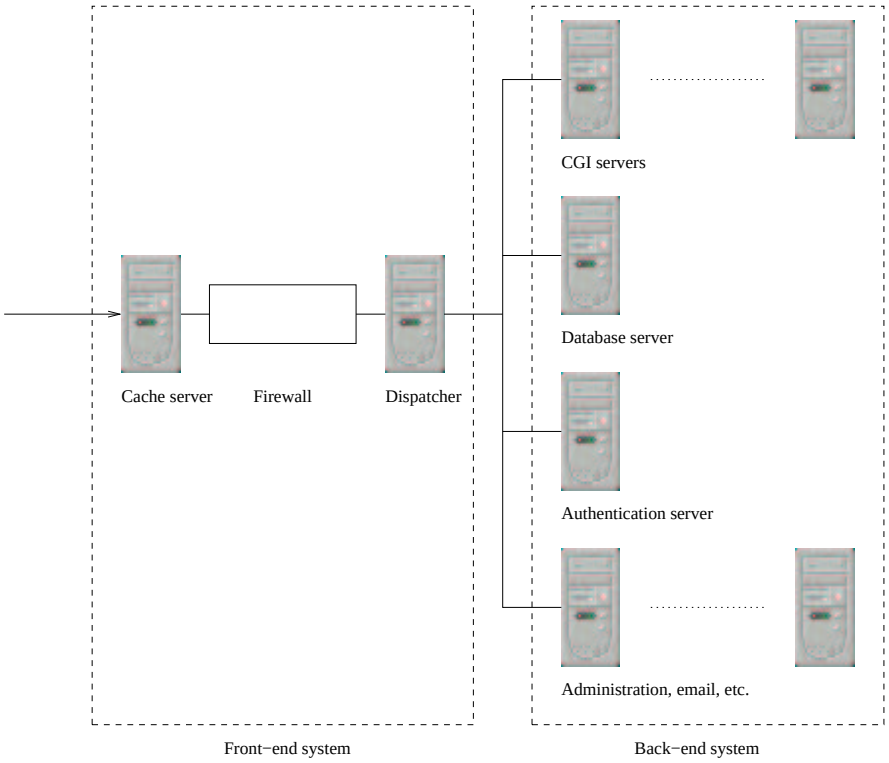

Figure 1: General structure of an e-commerce site

at wire speed and has no impact on performance. The dispatcher distributes incoming requests to the web servers using some algorithm. When a request belonging to a new session arrives at the dispatcher, a back-end web server is chosen to serve the session, typically by using round robin or random selection. The web server chosen will then be used for the remaining time of the session. Sessions can be identified in many ways, for example URLs, cookies or IP-addresses. The dispatcher may receive load information from the web servers to help in the distribution process. However, a simpler scheme is often used where the number of active TCP connections to a given server is limited to some threshold value.

The back-end servers on the inside of the dispatcher are typically connected with a fast LAN, such as a switched Fast Ethernet. The servers receive the HTTP requests from the customers. It answers the HTTP request by returning requested page or by running the requested server side scripts. If a script requires the services of one of the other back-end servers, the web server generates and sends a new request to the respective back-end server. Web servers are often replicated, so that all web servers have the same software. This means that more servers can easily be added if more capacity is needed. The database server has one or more databases containing customer data, inventory, catalogues and other data. The authentication server is used to verify the identity of a customer. In addition, there are often other servers that have other functions, such as e-mail or site administration. While they may be necessary for the functionality and maintenance of the site, they are usually not performance

bottlenecks.

The concept of customer session is of particular interest for e-commerce sites. A customer session is started when the customer first enters the site and lasts until the customer leaves the site. During the session, the customer generates a number HTTP GET requests to the web site. For an e-commerce site to be successful, it is obviously important that as many customers as possible complete their sessions. Only a customer that complete his/her session may generate some revenue. Therefore it is of utmost importance give ongoing sessions as high priority as possible in order to maximize the number of completed sessions.

One way to measure the performance of an e-commerce site is to measure the rate of so called *happy* and *angry* customers. A happy customer is a customer that leaves the site after finishing his/her session. A happy customer may generate some revenue for the service provider. However, a customer that leaves the site before the session is completed for example due to long waiting times or because a request is rejected in the middle of a session is instead called an angry customer. These customers have already spent some of their time in the site without being able to finish their sessions. Therefore they may hesitate to visit the site again.

Further, there are customers that are rejected before he/she has entered the site. These customers have not spent any time in the site. If the site informs these customers about the load situation and rewards them if they come back some other time, these customers may be classified as happy customers (or at least not angry customers).

3 Queuing network model of an e-commerce site

In this section a queuing network model is described that can be used to investigate the performance of an e-commerce site. The general network architecture is shown in figure 2. The web site consists of one front-end server, M web servers, one database server and one authentication server. The servers are connected by two high-speed LANs. In the following we will assume that the LANs are fast enough not to cause any delays and that they are unaffected by congestion.

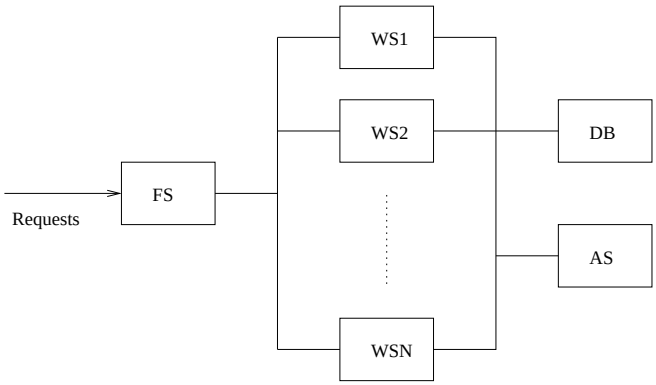

Figure 2: Network Architecture, with Front-end server (FS), Database server (DB), Authentication Server (AS) and N Web servers (WS1-WSM)

There are K classes of customers $(C_1, .., C_K)$. Each customer class has a unique behavior as will be discussed later. New customers arrive according to a Poisson

process with mean λ customers per second. A new customer belongs to class C_x with probability P_x. The fact that the Poisson process can be used to model customer arrivals at web sites has been shown by Paxson and Floyd [18].

3.1 Front-end server

A simple queue model of the front-end server is shown in figure 3. Incoming requests are placed in a queue. Requests are processed one by one in the server, in order to identify the requests. The requests are then sent to an overload control mechanism. If the request is admitted to the system, it is sent to a load balancing mechanism. The load balancer will send the request to one of the web servers. There are of course a number of load balancing and overload control schemes that may be used. In section 4, we describe the specific schemes that we have used in our investigations.

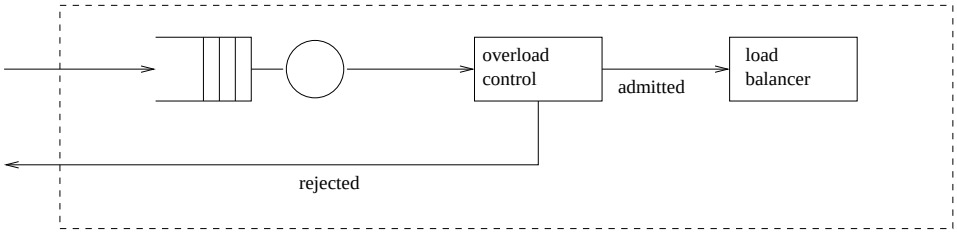

Figure 3: A model of a front-end server

The processing time for a request is deterministic with a value of x_f. We assume that the admission control and load distribution algorithm are simple enough not to cause any heavy processing in the server.

3.2 Back-end servers

The back-end servers are modelled as a queuing network that is shown in figure 4. The web servers and the authentication server are modelled as single server systems. Since the database usually is implemented as a cluster system, it may be modelled as a single queue with N servers.

Each back-end server performs a particular type of processing, called operation. The processing time for each operation type is denoted x_{WS}, x_{DB} and x_{AS} depending on the server that performs the operation. A request may need several operations before completed. In this case, the request will be sent between the servers that contain the operations needed to complete the request. We assume that the transmission times between the servers are close to zero. When a request has been completed, the answer is sent directly to the customer.

3.3 Customer behavior

A customer that visits an e-commerce site sends HTTP requests for varying types of data. There are a limited number of request types. For example, in a web store the request types are typically Search, Browse, Select, Add and Pay. A different set of requests can be identified for other types of e-commerce sites. In our general model we will assume that there are R types of requests, denoted $1..R$. When a customer has

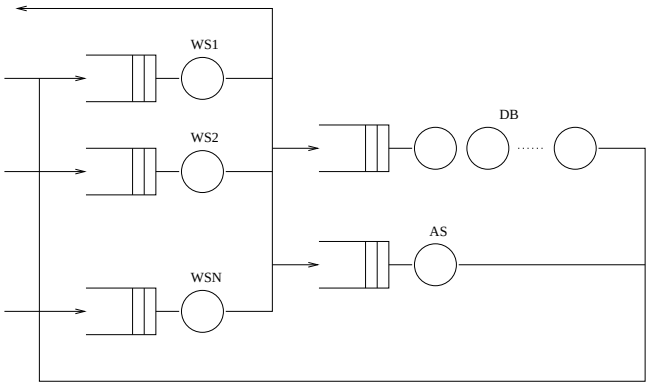

Figure 4: Back-end servers

sent a request to the site, he/she waits for an answer. When the answer has arrived, the customer either sends a new request or decides to leave the site.

It is important to have accurate customer behavior models in order to investigate the performance of e-commerce sites. A customer session can be modeled with a customer behavior model graph (CBMG), see for example [13]. In a CBMG, a customer can be in one of several states. Each state correspond to a specific request. We will use a simplified model of the CBMG, where the customer instead sends requests according to some probability distribution. A customer that belongs to class C_x sends a new request of type y with probability $p_x(y)$. With probability $p_x(leave)$ the customer instead leaves the site. Note that

$$p_x(leave) + \sum_{y=1}^{R} p_x(y) = 1$$

A customer that belongs to class C_x and that has sent a request of type y will wait S_{xy} seconds. If the answer has not arrived after this time, the customer becomes impatient and retries with a probability $p_x(retry)$. With probability $1 - p_x(retry)$, the customer leaves the site due to poor performance. A customer only retries once for each request.

When a customer has received an answer, there is some think time before the next request is sent. We use an exponential think time with mean D seconds since the distribution of the think time is of no importance for the steady-state performance of the web site.

3.4 Detecting timed-out customers

One problem in web sites is how to detect that a customer has left the site due to poor performance, a so called timed-out customer. A timed-out customer presses the stop button before receiving a reply from the site. If the site processes a request belonging to a timed-out customer, the processing is wasted.

Carter and Cherkasova [7] discusses and develops a method of detecting timed-out customers. We assume that timed-out customers can be detected. In the model this can be solved by adding a time-to-live parameter in each request. If a server detects

that the time-to-live has passed or will be passed during the processing, the request is not processed.

4 Simulation example

We will in this section give an example of an e-commerce site. We then use simulations and mathematical analysis to investigate the performance of the site. In particular we investigate the difference in performance when using a session based admission control instead of a request based control.

4.1 Network structure

The web site is shown in figure 5. There are two web servers. The database server has one processor. Customers arrive according to a Poisson process with mean l customers per second. There is one class of customers, that corresponds to an occasional buyer (see [13] for more details). An occasional buyer is a customer that mainly want to search for information. Studies show that most customers visiting a web site are occasional buyers (see, for example [15]).

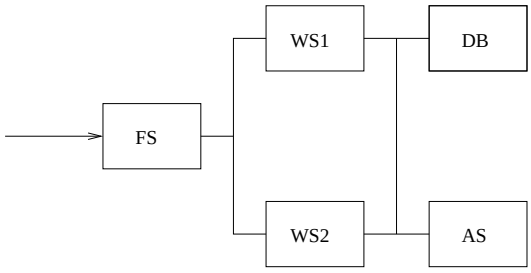

Figure 5: Back-end servers

4.2 Load balancing

As discussed before, load balancing algorithms for distributed web servers have been investigated in several papers. In this case we have homogenous servers, which means that the load will be evenly spread by a load balancing algorithm that uses round-robin or random selection. Therefore, we use a simple round-robin mechanism to decide a suitable web server for a customer. For heterogeneous server systems other types of load balancing must be used, for example weighted round-robin or content based routing [5].

4.3 Overload control

The overload control mechanism in this paper is designed as in figure 6. It consists of a gate, a controller and a finite waiting queue with L places.The controller uses measurements of the system to decide the rate at which requests can be admitted to the system. The gate admits customers according to this admittance rate. Requests that cannot be admitted at once, are placed in the waiting queue. If the waiting queue is full when a request arrives, the request is rejected.

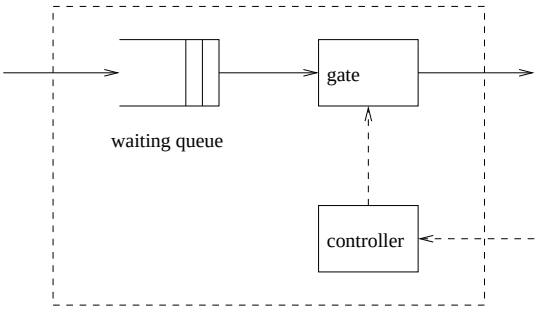

Figure 6: Overload control mechanism

We have investigated two types of load control mechanisms in this paper, one request based and one session based mechanism. In the request based case, all HTTP requests has to go through the load control mechanism. This is the mechanism that is most common today. In the session based case, only requests from new customers goes through the load control mechanism. Once a customer has been admitted to the site, he/she should be able to finish the ongoing session.

In both cases, the gate uses a dynamic window mechanism (see, for example Pham and Betts [17]) to admit new customers to the site. The window mechanism uses a controller parameter, W, which decided the maximum number of requests (in the request based case) or sessions (in the session based case) that are processed at the same time in the site. W is updated by looking at the request response times in the server cluster. If a request receives a response time higher than Thigh, W is decreased by one. If instead 20 requests have had a response time lower than T_{low}, W is increased by one. W is always between 1 and W_{max}.

If there are W_{max} ongoing requests/sessions when a new request arrives at the gate, the request is placed in the waiting queue. If the waiting queue is full, the request is rejected and a notification message is sent to the customer informing about the current overload. A customer that is rejected leaves the site.

When a customer is placed in the waiting queue, it is possible to estimate the waiting time for this customer. If the estimated waiting time is longer than a threshold, a notification message should be sent to this customer informing about both the waiting time and the number of customers already waiting. This means that the customer can make a decision to wait or leave the site and try again some other time. According to the study performed in [4], the customers regard this as an acceptable behavior of the site.

4.4 Customer behavior

The customer behavior depends on the type of e-commerce site that is investigated. This simulation example represents a web store, which means that the customers typically send requests for browse, search, select, add to cart and pay. This means that we have five types of requests ($R = 5$). In the following we number the requests as follows: Browse=1, Search=2, Select=3, Add=4 and Pay=5.

The customers send requests according to a probability distribution, $p(i)$, where i is the type of request. We have used the following probability distribution: $p(1) = 0.37, p(2) = 0.36, p(3) = 0.15, p(4) = 0.015$ and $p(5) = 0.005$. With probability

$p(leave) = 0.1$, the customer has completed his/her session and therefore leaves the site. This probability distribution correspond to the CBMG of an occasional buyer in Menascé *et al* [13].

4.5 Simulation parameters

Table 1 shows the values of the parameters used in the simulations. We have assumed that a customer never becomes impatient. This makes it easier to understand the behavior of the load control schemes. Further, the mean think time between requests is rather short, however this parameter will not affect the steady-state behavior of the system.

	Value
Processing time in front-end server, x_f	1 msec
Processing time in a web server, x_W	10 msec
Processing time in the database server, x_{DB}	5 msec
Processing time in the authentication server, x_{AS}	10 msec
Probability of a retry, $p(retry)$	0
Mean think time, D	5 sec
Average waiting time between requests, $S_y(y = 1..5)$	infinity
Length of waiting queue, L	10
Lower threshold for response times, T_{low}	7 sec
Upper threshold for response times, T_{high}	8 sec
Maximum window size, W_{max}	500

Table 1: Simulation parameters

Table 2 shows the different back-end servers required to complete each type of request. As can be seen in the table, a request may need to be processed more than once in a particular back-end server. This is due to the dynamic CGI scripts that have to be processed before the requested web page can be returned. For each CGI script some data has to be read or written in the database or the authentication server. For example, when a customer sends a search request, several CGI scripts need to be executed. We assume that a search query in average consists of two words (see, for example [12]), therefore two item lists have to be constructed and merged. In the browse request, only one item list has to be constructed.

Request	Back-end servers	Processing time (ms)
Browse	Web, Database	15
Search	Web, Web, Database, Database, Web	40
Select	Web, Database	15
Add	Web, Database	15
Pay	Web, Database, Web, Authentication	35

Table 2: Request content

5 Results

Here follows the results from the investigations. We simulated the system with varying arrival rates. Due to the unexpected results that were obtained we find it necessary to also perform a mathematical analysis of the system. Therefore, we here present some

preliminary analytical results. However, further mathematical analysis is necessary in order to understand the behavior of a distributed e-commerce site.

5.1 Simulation results

Figure 7 shows the average throughput, measured in completed sessions per second, for the two load control schemes. As can be seen, the throughputs are surprisingly similar. If one think of traditional load control theories, the throughput in the request based case should decrease when the load increases. In the request based scheme, customers may be rejected also when they are in the middle or at the end of their session. If such a customer is rejected, capacity will be wasted and the throughput should thereby decrease. However, this is not the case here. Further, there is no obvious reason why the throughputs are almost equal. The next section contains a mathematical analysis of this unexpected behavior.

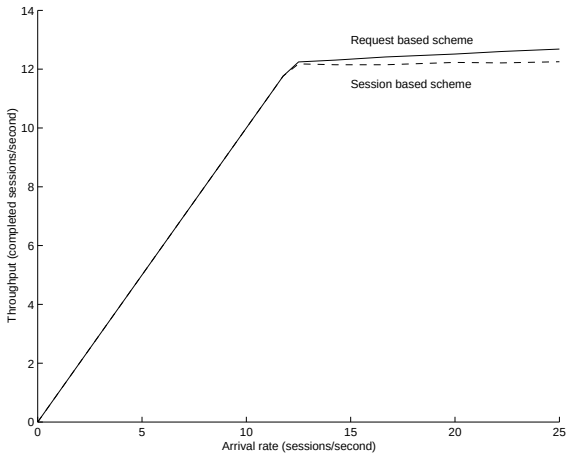

Figure 7: Average throughput

However, if one looks at the percentage of angry customers as shown in figure 8 one can see that there is a clear difference between the two schemes. In the request based scheme the percentage of angry customers increases rapidly when the load increases. This is due to the customers that are rejected in the middle of there sessions. In the session based scheme on the other hand, customers may only be rejected in the beginning of their session. Therefore, they are not defined as angry customers (see section 2),

Another major difference between the two schemes can be seen in figure 9. Here, the average session length, measured in number of requests, for completed sessions is shown. In the request based scheme, the average session length decreases when the load increase. Since all requests have the same probability of becoming rejected, the scheme will prioritize short sessions. In the session based scheme, all customers that have been admitted will complete their sessions. Thereby, the average session length stays unchanged when the load increases.

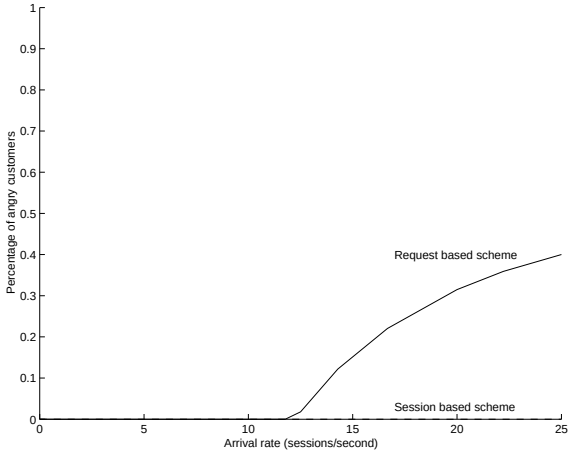

Figure 8: Percentage of angry customers

5.2 Mathematical analysis

This section contains an short mathematical analysis of the behavior of the two load control schemes. The system that is analyzed is a simple server with an infinite queue. However, the results may be generalized to more complex systems. The objective of the analysis is to show that, during certain conditions, the throughput will be the same for the two control schemes. We hope to present a more extensive mathematical analysis of the system in a future paper.

First, let p be the probability that a HTTP request is accepted in the request based mechanism and let $\tilde{p}$ be the probability that a session is accepted in the session based mechanism. Assume the following:

1. The probability that the number of requests in a session, denoted n, is $q^{n-1}(1-q)$ for $n \geq 1$, i.e. we have a geometrical distribution.

2. The mean processing time, denoted $\overline{x}$, is the same for all requests.

Now we will show that if p and $\tilde{p}$ are given values so that the load is the same in the two cases, the probability that a session is accepted will be the same for both mechanisms.

Let us first calculate the load on a server for both mechanisms. For the session based mechanism, assume that a session consists of n requests. Then the probability that i of these are served is $p^i(1-p)$ for $0 \leq i \leq n-1$ and p^n for i=n. The mean number of requests that are served in a session is $p(1-p^n)/(1-p)$, removing the condition on the number of requests in a session gives the mean number of requests in an arbitrary session:

$$\sum_{n=1}^{\infty} \frac{p(1-p^n)}{1-p} \cdot q^{n-1}(1-q) = \frac{p}{1-pq}$$

The load on the server for the request based mechanism will thus be

$$\frac{\lambda \overline{x} p}{1-pq}$$

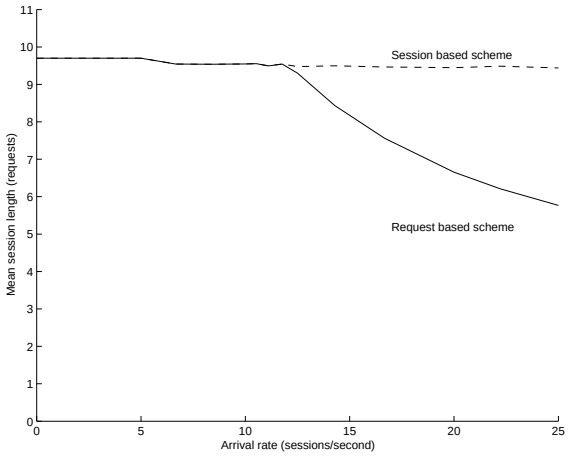

Figure 9: Mean session length

where is the mean processing time of a request and is the arrival rate of new sessions.

For the session-based mechanism the load on the server will be

$$\frac{\lambda \overline{x} \tilde{p}}{1-q}$$

since is the mean number of TCP connections in a session. This means that if and are given values so that the load is the same in both cases, we must have

$$\tilde{p} = \frac{p(1-q)}{1-pq}$$

The throughput of the request based method can be calculated if it is observed that the probability that all requests in a session are served is if a session consists of requests. Removing the condition on the number of requests in a session gives the probability that an arbitrary session is not aborted in advance:

$$\sum_{n=1}^{\infty} p^n q^{n-1}(1-q) = \frac{p(1-q)}{1-pq}$$

which means that the throughput will be

$$\frac{\lambda p(1-q)}{1-pq}$$

The session based mechanism has throughput which means that the throughput is the same for the session-based and request-based mechanism since $\tilde{p} = (1-q)p/(1-pq)$ if the load is equal. This explains the similar throughputs for the two load control schemes.

6 Conclusions

Today, many e-commerce sites have performance problems due to inaccurate dimensioning and lack of proper load control mechanisms. In order to design high capacity e-commerce sites that can meet customers' QoS expectations it is therefore necessary to develop and analyze mathematical models of these type of systems.

We have in this paper developed a queueing network model for a distributed e-commerce site. With the model it is feasible to investigate the performance of a site. Customers visiting the site belongs to one of several classes that each have a specific behavior. Further, the model contains both a load balancing and an overload control mechanism.

The queueing network model is used to investigate two types of overload control schemes: one request based scheme and one session based scheme. In the request based scheme, all requests may be rejected in the case of overload. In the session based scheme on the other hand, only requests belonging to new sessions may be rejected.

First, the system is investigated with simulation. We show that the throughput of completed sessions is almost equal for the two control schemes. However, in the request based scheme the number of so called angry customers increase rapidly when the load increases whereas in the session based scheme the number of angry customers stays low. Further, we show that the request based scheme prioritizes short sessions since the session rejection probability increases when the session length increases.

Also, we mathematically analyze a simple server system. We show that when the session length is geometrically distributed and when all requests have the same processing time, the two control schemes will give the same throughput.

References

[1] T.F. Abdelzaher and N. Bhatti, *Web content adaptation to improve server overload behavior*, Computer Networks, Vol 31 (1999), pp 1563-1577.

[2] A. Bhargava and B. Bhargava, *Measurement and quality of service issues in electronic commerce software*, Proc. of Symposium on Application-Specific Systems and Software Engineering and Technology, 1999, pp 26-33.

[3] N. Bhatti and R. Friedrich, *Web server support for tiered services*, IEEE Network, Sept/Okt 1999, pp 64-71.

[4] N. Bhatti, A. Bouch and A. Kuchinsky, *Integrating user-perceived quality into web server design*, Computer Networks, Vol. 33 (2000), No. 1-6, pp 1-16.

[5] H. Bryhni, E. Klovning and . Kure, *A comparison of load balancing techniques for scalable web servers*, IEEE Network, July/Aug 2000, pp 58-64.

[6] V. Cardellini, M. Colajanni and P.S. Yu, *Dynamic load balancing on web-server systems*, IEEE Internet Computing, May/June 1999, pp 28-39.

[7] R. Carter and L. Cherkasova, *Detecting timed-out client requests for avoiding livelock and improving web server performance*, Proc. of 5th IEEE Symposium on Computers and Communications, 2000, pp 2-7.

[8] L. Cherkasova and P. Phaal, *Predictive admission control strategy for overloaded commercial web server*, Proc. of 8th International Symposium on Modeling Analysis and Simulation of Computer and Telecommunication Systems, 2000, pp 500-507.

[9] M. Colajanni, P.S. Yu and D.M. Dias, *Analysis of task assignment policies in scalable web-server systems*, IEEE Trans. Par. Distr. Systems, Vol. 9, No. 6, June 1998.

[10] D.M. Dias, W. Kish, R. Mukherjee and R. Tewari, *A scalable and highly available web server*, Proc. of IEEE Computer Conference, 1996, pp 85-92.

[11] A. Iyengar, E. MacNair and T. Nguyen, *An analysis of web server performance*, Proc. of Globecom'97, 1997, pp 1943-1947.

[12] W. Meira Jr., D. D. Menascé, V. Almeida and R. Fonseca, *E-representatives: a scalable scheme for e-commerce*, Proc. of 2nd International Workshop on Advanced Issues of E-commerce and Web-based Information Systems, 2000, pp. 168-175.

[13] D. Menascé, V. Almeida, R. Fonseca and M. Mendes, *Business-oriented resource management policies for e-commerce servers*, Performance Evaluation, Vol 42(2000), pp 223-239.

[14] D. Menascé and V. Almeida, *Scaling for E-business*, Prentice Hall, 2000.

[15] J. Nielsen, *Why people shop on the web*, http://ww.useit.com/alertbox/990207.html, 1999.

[16] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel and E. Nahum, *Locality-aware request distribution in cluster-based network servers*, Proc. of ACM 8th International Conference on Architectural Support for Programming Languages and Operating Systems, 1998, pp 205-216.

[17] X. Pham and R. Betts, *Congestion control for Intelligent networks*, Computer Networks and ISDN Systems, vol. 26, 1994, pp 511-524.

[18] V. Paxson and S: Floyd, *Wide area traffic: the failure of Poisson modeling*, IEEE/ACM Transactions on Networking, Vol. 3, No. 3, June 1995, pp 226-244.

[19] T. Schroeder, S. Goddard and B. Ramamurthy, *Scalable web server clustering technologies*, IEEE Network, May/June 2000, pp 38-45.