Gateway-based call admission in distributed object oriented systems

Widell, Niklas; Nyberg, Christian

Published in:
Proceedings fifteenth Nordic Teletraffic Seminar, NTS-15, Lund University, August 22-24, 2000

2000

Gateway-based Call Admission in Distributed Object Oriented Systems

Niklas Widell and Christian Nyberg

Department of Communication Systems,
Lund Institute of Technology, Sweden,
e-mail: {niklasw, cn}@telecom.lth.se

Abstract

Many applications in telecommunications will depend on distributed systems to provide enough capacity. In a distributed system a service is split up into a number of modules (often called objects) that can be placed at different nodes or processors in a network. A service can be seen as a number of invocations of the objects in a certain order. There are a number of performance problems which have to be solved. How shall objects be distributed on the nodes? How shall the load be distributed among the nodes for a given object distribution? How shall the distributed system be protected from temporary overload situations?

In our paper we investigate these questions for distributed systems where requests for service arrive to a gateway from which they distributed to the nodes of the system. We assume that the object distribution is given and we concentrate on protecting the system using three different external load control mechanisms: Percent Thinning, Call Gapping and Tokens. Using simulations, we find that Tokens provide the best system protection.

1 Introduction

In the future the development of telecommunication technology will be characterized by:

- Integration of services in networks. The networks will to a larger and larger extent be based on the TCP/IP protocol stack.
- Many new services with very different demands on network resources like transmission and processing capacity will be developed. Examples of such services are video conferencing, multimedia services and Internet services.
- The demand for both personal and terminal mobility will increase.
- The role as service provider and operator will be separated in the future.

To be able to handle these developments, new service architectures have been or are being developed. Examples of these are the different versions of Intelligent Networks, where services and data are located at special nodes in the network, and the Telecommunications Information Networking Architecture (TINA), which provides a fully distributed service architecture. Both IN and TINA have been standardized by work groups with representatives

from all large operators and vendors. A problem with TINA is the large complexity of the architecture, which has halted its use. Instead, distributed architectures based on the TCP/IP stack are being developed and they are likely dominate in the future.

There are a number of performance issues which need to be solved for distributed architectures. Some of the most important are:

- How shall objects be distributed on the nodes? Observe that there may exist more than one instance of an object in a distributed system. From a performance point of view the best would be to have one instance of each object type in all nodes. Then a call would only have to be executed in one node. However, this is not feasible. For instance, having multiple copies of large databases is not feasible for administrative reasons
- How shall the load be distributed among the nodes for a given object distribution? Communications between nodes is fairly expensive in terms of processing, while at the same time distribution without inter-node communication is useless.
- How shall the distributed system be protected from temporary overload situations? We need overload protection on both internal (node) and external (system) levels.
- What traffic models shall be used? It has been observed that the packet traffic in networks might have a fractal behavior which is very different from the Poisson traffic used to model telephone traffic.

Performance models of distributed systems are thus needed to be able to study, among other things, response times, optimal allocation of computational objects and overload control algorithms. In this paper we will concentrate on some aspects of overload control.

Overload control of nodes in telecommunication networks has been an active research area for many years and many studies have been published. One example of this is [1], which is a survey of early results with many references. [2] compares the efficiency of different throttling mechanisms (especially call gapping and percent blocking) that can be used to reject calls. What is common to these early studies is that they concentrate on protecting one node with one processor from being overloaded. As pointed out above, distributed system architectures are likely to get more and more common. Thus there is a need to investigate the performance of these architectures.

Several papers have discussed load balancing and load sharing in computer networks. However, very few of these examine load balancing in distributed systems. [3] discussed load sharing algorithms for distributed systems. [4] examined how the network nodes should exchange load status information to be used in load balancing schemes. [5] investigated call admission policies for communication networks that support several services. [9] investigate how to assign a number of tasks to to a number of processors in order to minimize, for example, the maximum completion time. The ACTS project MARINER has published extensively on using market-based agent technology to protect distributed systems in Intelligent Networks, see [10].

This paper extends the work of [2] by comparing different throttling mechanisms used not only to protect one single processor but a distributed system built up by several processors. It also extends the work in [10], since there only one throttling mechanism (tokens) was used.

2 Call Admission Strategy - External Load Control

If the call arrival rate is higher than what a system can handle, calls must be rejected in order to preserve system sanity. We assume that all calls arrive at one node, called gateway. At the gateway, calls can be rejected if the system is overloaded. If a call is accepted it may proceed to the distributed system which consists of a number of nodes. A call has to be served at several of these nodes. Also these nodes can reject a request if they are too heavily loaded in order to preserve system sanity. If a request is not rejected at the gateway or at any other node, it is eventually served. However, if it takes too long time to serve a call, it will be regarded as a failure, because that might trigger timeouts or make users of the system impatient which in its turn might cause unwanted system behavior. Thus we can divide all requests into four classes:

1. Requests that are rejected at the gateway.
2. Requests that are accepted at the gateway but are rejected by a node in the distributed system.
3. Requests that are not rejected at the gateway or at any other node but are not finished in time.
4. Requests that are not rejected and finished in time.

The goal of overload control is that the number of requests of class 4 shall be as large as possible. Naturally, the number of requests of class 2 and 3 shall be as small as possible since they are not successful and the processing time spent on them is wasted.

Three different throttling algorithms that can be used in the gateway namely Call Gapping, Tokens and Percent Thinning are investigated. The algorithms are described in more detail in the description of our model. Two different arrival processes are used, a Poisson process with constant rate and a Poisson process with a variable rate. It is reasonable to assume that request rates will show large variations in many new services triggered by different kinds of events.

3 System Model

In this section we develop a useful model for studying performance in distributed object oriented systems. The model is generic, but contains all the important parts from real architectures.

3.1 A Generic Model

Distributed object oriented systems tend to be immensely complex, with several thousands of objects and many physical nodes of varying capabilities. Figure 1 presents a model of a simple four node network, with a number of objects named A to E. The network is a set of connected nodes. Connections between nodes are considered to be links of high capacity, so that transmission times are negligible. Service requests arrive to a special gateway node with an intensity λ. The gateway does some processing to decide whether to accept a service

request or not. If a service request is accepted to the system it is forwarded to the nodes where the actual service execution takes place.

The nodes represent the physical processing hardware in model. Nodes are processors, that can execute tasks given to them. For our purposes, each node consists of three parts; a queue, marked Q in figure 1, a server and finally a router, marked S and R respectively. All arriving jobs that cannot be handled immediately are placed in the queue. The server model the actual processing of a job. The router takes a finished job from the server and sends it along to another node for more processing.

The router is the component in each node that decides where to send a job when it has finished executing. It is in this routing procedure that we can take into account load balancing and load sharing by making intelligent choices of where to direct a job when it is finished on one node.

3.1.1 Objects

An object in our model is a representation of an independent piece of software that can perform one or more functions as well as interact with other objects. Every function call takes a certain amount of time to execute depending on the complexity of the call. The sequence of function calls made by the objects is described by a service, see below.

As objects represent software, they can migrate between nodes in the network. The migration process makes it easy to reconfigure the system by moving objects around. However, migration is a very processing intensive operation and using migration to move objects around to help in an overload situation is not a very likely situation. On the other hand, in longer time perspectives migration is a very useful way to change system configuration to increase performance as traffic patterns change. As we are interested in studying the real-time behavior of the system, we assume that a steady state has been reached for the object distribution.

In object oriented programming it is common to use the term *class* as something from which objects are instantiated from. As we look at systems during runtime, what we see are instantiated objects or nodes where objects of a certain class can be instantiated. Some objects might be very long lived, such as databases, while others may last only during the lifetime of a

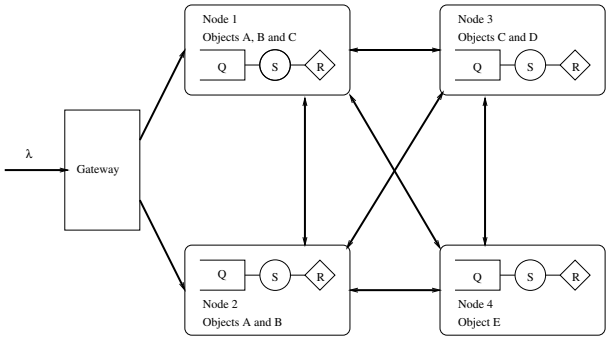

Figure 1: A simple four node network with a gateway

service. For our purposes we do not make any difference between long or short lived objects. Each object can perform only one action at a time. It does not matter whether all function calls to an object type goes to one particular object or to several different ones, it will still take the same amount of time to process them.

3.1.2 Services

A service is a request by a user for the system to do something, for instance to set up a telephone call. In the model it is a sequence of tasks that a system can perform. Each task can be performed by one object type, so a service describes the order in which objects are called.

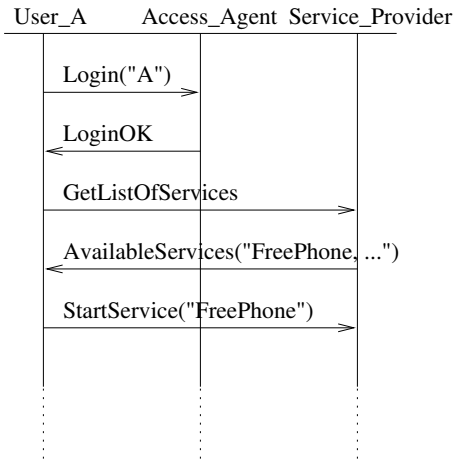

Figure 2: Message sequence chart for a service scenario

In figure 3 a typical generic service scenario is shown. The scenario is a generalized and simplified form of a TINA access and service session, see below. A user (represented by the object *User_A*) wishes to start a service, for instance FreePhone. Object *User_A* first interacts with an *Access_Agent* object that can authenticate the user, for instance by the user typing in a password. When authentication is ready, *User_A* requests a list of services from a *Service_Provider*. The *Service_Provider* returns a list of available services that *User_A* can choose from. Execution then continues with the system providing the service or services the user asked for, until the user logs out.

3.2 Algorithms

As mentioned above, we have studied three different algorithms: Percent Thinning, Call Gapping and Token. Below is a description of these.

3.2.1 Percent Thinning

For each arrival to the system, a random number is generated. This random number is compared to a given blocking probability. If the random number is above the blocking probability, the arriving session request is passed in to the system.

In a real system the blocking probability should be continually updated by some kind of control loop, to take into account fluctuation in arriving traffic. However, in this paper no such loop is used.

3.2.2 Call Gapping

Arriving session requests are only allowed into the system at certain times. Only the first arriving service request will be allowed into the system during a given interval. If more than one requests arrive during a interval, the following ones are rejected.

In a real system the interval length should be continually updated by some kind of control loop, to take into account fluctuations in arriving traffic. However, in this paper no such loop is used.

3.2.3 Tokens

The gateway has a number of tokens, that are handed out to arriving session requests. If an arriving session request finds a free token, it immediately continues execution. Otherwise, it is rejected. When a session is fully served, the token is handed back to the gateway. Thus, the number of tokens in use is the same as the number of sessions presently in the system.

The token as such should not be thought of something that "travels with" the session as execution brings through the system. A better representation is that there is a limit to the number of initial objects that can be in use at the same time (object A in simulations below). If a session is lost (blocked by an internal node), the initial object discovers this by some kind of time-out, and the token can be returned to the gateway.

4 Simulations

This section describes a number of simulations that we have run to study the behaviour of the different algorithms. We first present the parameters used, and then describe the different simulation cases.

4.1 Simulation parameters

The following table lists the input values used in the simulations:

Parameter	*Value*	*Comments*
Nodes	5	Node 1 is gateway node
Objects	5	Named *A* - *E*
Session sequence	*A-B-A-C-D-C-A-C-E-C-A-B-A*	
$\overline{x}_A$	0.01s	per session step
$\overline{x}_B$	0.01s	per session step
$\overline{x}_C$	0.01s	per session step
$\overline{x}_D$	0.05s	per session step
$\overline{x}_E$	0.12s	per session step
$\overline{x}_{marshall}$	0.01s	per session step
$\overline{x}_{unmarshall}$	0.01s	per session step
$\overline{x}_{session}$	0.52s	"empty system"
$T_{timeout}$	2.25s	time before timeout

The table below contains object distribution and respective routing probabilities. A dash means that this type of object is not available on this node. Routing inside the network was random.

Object	*Node 1*	*Node 2*	*Node 3*	*Node 4*	*Node 5*
A	1.0	—	—	—	—
B	—	0.50	0.50	—	—
C	—	0.33	0.33	0.34	—
D	—	—	0.5	0.5	—
E	—	—	—	—	1.0

The used arrival process was Poissonian, with the following arrival intensities.

CONSTANT Constant arrival rate, with $\lambda = 10s^{-1}$. This arrival intensity causes overload in nodes 1 and 5.

TRANSIENT First 40 seconds with $\lambda = 5s^{-1}$, then $\lambda = 20s^{-1}$ for 30 seconds, and finally $\lambda = 5s^{-1}$ again for another 40 seconds.

All the cases above have the same total number of requests over the simulation interval. For all arrival intensity cases the same realization of the input process was used for the different algorithms.

5 Results And Conclusions

This section contains the results from the simulations, as well as a discussion of these results.

The success rate given is the ratio of fully successful jobs to the total number of jobs arriving to the system.

5.1 Constant arrival intensity

The table below contains the best results gathered for the given arrival intensity. The parameters for each algorithm was set to give the highest possible success rate.

Algorithm	*Success rate*	$\overline{x}_{session}$	*Comment*
Token	71%	$1.73s$	# of tokens was 14
Percent Thinning	63%	$1.36s$	Allow percentage was 68%
Call Gapping	67%	$1.32s$	Gap interval length was 0.08s

5.2 Changing arrival intensity

At first, simulations were run with the same algorithm parameters as in the previous cases.

Algorithm	*Success rate*	$\overline{x}_{session}$	*Algorithm parameters*
Token	62%	$1.23s$	# of tokens was 14
Percent Thinning	42%	1.30	Allow percentage was 68%
Call Gapping	34%	1.33	Gap interval length was 0.08s

Obviously, this comparison is not fair to neither Percent Thinning nor Call Gapping, as the algorithm parameters could not change due to changing input traffic. Therefore, we also investigated how these algorithms would behave if the parameters were set in an "optimal" way. The following table contains the results from these simulations:

Algorithm	*Success rate*	$\overline{x}_{session}$	*Algorithm parameters*
Percent Thinning	58%	$0.88s$	100%/34%/100%
Call Gapping	60%	$1.06s$	$0.001s/0.13s/0.001s$

The same arrival intensities as in the previous cases. However, Call Gapping and Percent Thinning both uses optimal values for interval lengths and blocking percentage when intensities change. This is similar to having a perfect control system that follows the imput process exactly.

Note that parameters are given for low/high/low intensities.

5.3 Conclusions

Given the results above, it can be seen that the token algorithm performs very well indeed, even when the other two are given "optimal" parameters in terms of interval length and blocking percentage.

While calculating the best values to use for algorithm parameters, Call Gapping proved to be very sensitive to even small changes in interval length.

In line with Berger, Call Gapping performs better than Percent Thinning, except in the transient case with constant algorithm parameters. This might be another sign that Call Gapping is very sensitive and that the interval length must be set very carefully.

6 Future Work

This paper presents a snapshot of some work in progress in the research of distributed object oriented systems. Many more areas remain to be investigated.

As services are likely to have varying processing requirements, tokens could possibly be refined to something like a currency system, where different services need different amounts of tokens to be allowed into the system.

References

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

[2] 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.

[3] 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.

[4] 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.

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

[6] A. Parhar and M. 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.

[7] N. Widell, M. Kihl and C. Nyberg, *Measuring real-time performance in distributed object-oriented systems*, Proceedings of Performance and Control of Network Systems III, Boston, 1999.

[8] M. Kihl, N. Widell and C. Nyberg, *Load Balancing Algorithms for TINA Networks*, The 16:th International Teletraffic Congress, Edinburgh, 1999.

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

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