

This is an author produced version of a conference paper presented at Performance and control of network systems III : 20-21 September 1999, Boston, Massachusetts. This paper has been peer-reviewed but may not include the final publisher proof-corrections or pagination.

Citation for the published paper:

Niklas Widell, Maria Kihl, Christian Nyberg (1999). Measuring real-time performance in distributed object oriented systems. In: *Performance and control of network systems III : 20-21 September 1999, Boston, Massachusetts*. ISBN: 0-8194-3434-5.

Publisher: The Society of Photo-Optical Instrumentation Engineers (SPIE)

Copyright 1999 Society of Photo-Optical Instrumentation Engineers. This paper was published in Proceedings of SPIE, 3841 and is made available as an electronic reprint with permission of SPIE. One print or electronic copy may be made for personal use only. Systematic or multiple reproduction, distribution to multiple locations via electronic or other means, duplication of any material in this paper for a fee or for commercial purposes, or modification of the content of the paper are prohibited.

# Measuring Real-time Performance in Distributed Object Oriented Systems

Niklas Widell, Maria Kihl and Christian Nyberg

Department of Communication Systems  
Lund Institute of Technology  
BOX 118, SE-22100 Lund, Sweden

## ABSTRACT

The principles of distributed object oriented programming offer great possibilities for flexible architectures in multiple fields. In telecommunications, an architecture called Telecommunication Information Networking Architecture (TINA) has been developed using these very principles. It allows telecommunication services to be implemented using software objects that in turn can be executed in a location transparent way in a network. The location transparency offers great flexibility for service creation, but as the software must be executed somewhere in the network on nodes of finite capacity, performance problems can arise due to inefficient placement of objects causing either overloaded nodes or excessive and unnecessary inter-node communication. To ensure good performance, various measures of load control and load balancing must be taken. We discuss how to measure the performance of a distributed object oriented system and examine two load balancing algorithms that can be used in such systems.

**Keywords:** Performance, Distributed object oriented systems, TINA, Load balancing, Load control

## 1. INTRODUCTION

Recent development in the area of Distributed Object Oriented Computing has influenced the telecommunications industry to investigate how this technology can be used to support new telecommunication architectures. The concepts of distributed processing and object orientation offer great possibilities for rapid creation and good management of telecom services.

One major problem of the proposed systems is that the systems themselves can be performance bottlenecks. Objects are to be distributed on different physical nodes, which means that different object distributions will require different amounts of inter-node communication as well as different load on the nodes. Inter-node communication is expensive in overhead and it is an important goal to keep it as small as possible while maintaining the positive sides of distribution.

A common solution to the performance problem is simply to “buy faster machines”. As Jain<sup>1</sup> argues this often fails to solve the problem, or might even make the problem worse. Also, it might be interesting for economic reasons to solve the problem by using the available system more efficiently.

Good performance can be reached by several means. It can be faster hardware, efficient implementation of communication protocols, efficient implementation of the application itself, information caching and many others. In this paper we discuss distributed object oriented real-time systems from a communication performance point of view. We describe how object distribution affects performance, how performance can be measured and various ways how performance can be maintained by overload control, load balancing and other methods.

We will discuss three terms; load balancing, load sharing and load control. The difference between load balancing and load sharing is that the former strives to equalize the load among the nodes, while the objective of the latter is to ensure that no node stays idle while others are under heavy load. The objective of load control algorithms is to make the system perform efficiently during heavy load.

Research into load balancing and load sharing problems is extensive. However, so far it has mostly been directed at load balancing one step ahead, trying to find the best place to execute the next operation but not the one after

---

Send correspondence to Niklas Widell, E-mail; [niklasw@tts.lth.se](mailto:niklasw@tts.lth.se)

that. This may not be sufficient for more complex systems. Kremien *et al*<sup>3</sup> has a thorough analysis of load sharing algorithms. Kunz<sup>2</sup> describes how different workload descriptions impact on performance.

This paper further extends work by Kihl<sup>6</sup> and Kihl *et al*<sup>7</sup> in which some initial studies of load control and load balancing in a distributed object oriented telecom architecture called Telecommunication Information Networking Architecture (TINA) were made. Parhar *et al*<sup>4</sup> also made some early investigations on performance TINA-based networks. Rumsewicz<sup>5</sup> discuss heterogeneous networks and performance.

The rest of this article is structured as follows: in section 2 we develop a model that can be used to study performance in distributed systems, in section 3 we discuss various means by which performance can be measured in our systems, in section 4 we discuss requirements for load balancing and load control, in section 5 we use our model to simulate two simple algorithms for load balancing and in section 6 we present the results of our finds. Section 7 contains conclusions of our study and some ideas for future work.

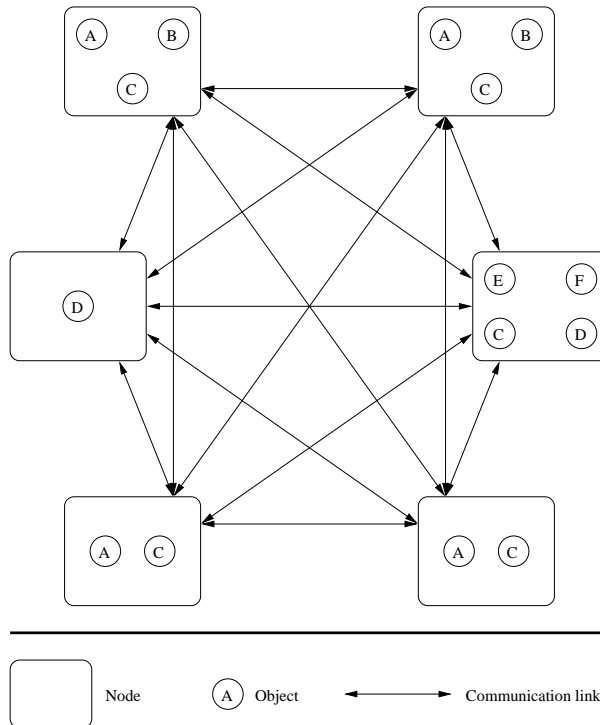
## 2. MODELING DISTRIBUTED OBJECT ORIENTED REAL-TIME SYSTEMS

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. At the end of the section we describe how our model maps to TINA. We also mention other areas where our model might be used to study performance of systems.

### 2.1. A Generic Model

Distributed object oriented systems can be immensely complex, with several thousands of objects and many physical nodes of varying capabilities. To study these systems we need to have a model that is simple enough to give insightful information of how suggested algorithms impact on performance, while at the same time having necessary complexity to capture all the relevant details of the real systems.

Figure 1 presents a top view of our model. The concepts in the model are explained below



**Figure 1.** A simple six node network with generic objects A to F

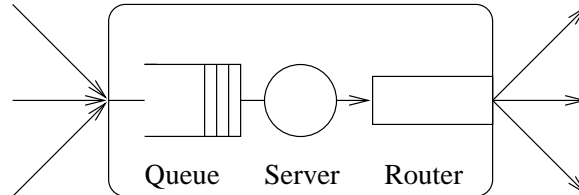
### 2.1.1. Network

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. We make no further assumptions concerning the network.

### 2.1.2. Nodes

The nodes represent the physical processing hardware in our simulations. Nodes are processors, that can execute tasks given to them. For our purposes, each node consists of three parts; a queue, a server and a router, see figure 2. 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. We describe the routing process in more detail below.

Communication between nodes is assumed to be very fast, and we can ignore switching and transmission times as they are assumed to be small. Thus, all nodes in a network can be assumed to be fully connected. The actual transport between two nodes is instead modeled as an added execution at sending and receiving node, which is a fair representation of all necessary protocol processing.



**Figure 2.** Internal node structure

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.

### 2.1.3. Objects

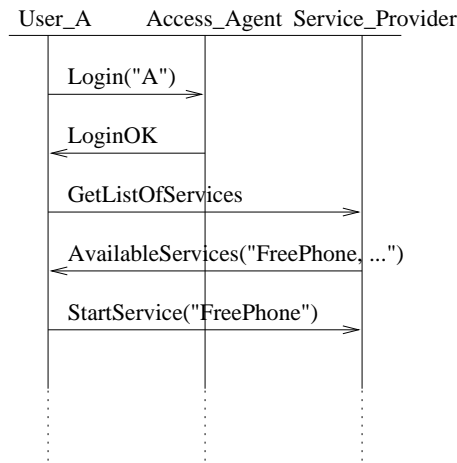
An object in our model is a representation of an independent piece of software that can perform one or more functions as well as interact with other objects. Every function call takes a certain amount of time to execute depending on the complexity of the call. The sequence of function calls made by the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

### 2.1.4. 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 3.** 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.

## 2.2. Mapping TINA to Our Model

The TINA architecture provides a set of concepts and principles to be applied in the development of software for telecommunication systems. The specifications of TINA are very comprehensive (for an introduction see Chapman *et al*<sup>9</sup>) and we shall here only explain how our model can be used to simulate TINA networks.

One important concept in TINA is *session*. A session is defined as a set of activities carried out with a specified goal. The TINA specifications specify three different session types:

**Access session** The access session deals with authentication and authorization of users.

**Service session** The service session provides users with an environment in which to execute services.

**Communication session** The communication session supports activities needed to establish communication stream connections between users.

A service specification defines the interaction that takes place in a service between software entities called Service Components (SCs). An SC is an abstraction that encapsulates data and processing. Each SC provides a set of interfaces that can be used by other SCs to access the capabilities of the SC. In our model each SC would map to one object.

Below the Service component layer is the Distributed Processing Environment (DPE, see Graubmann *et al*<sup>11</sup>). The DPE provides communication facilities for the Service Components to use. It hides the distributed nature of the system, making location transparent to the SCs. Load balancing and load control functions are according to the specifications to be implemented in the DPE.

## 2.3. Modeling Other Systems

The model proposed in this paper can be used to study a number of interesting systems other than TINA:

**CORBA** TINA is to a large extent based on CORBA (the Common Object Request Broker Architecture) developed by the the Object Management Group (see Vinoski<sup>12</sup>). The CORBA specifications as such have no real-time properties, but there is ongoing work to provide such facilities. CORBA is a very general architecture that can be used in many different fields, and while not all of them are extremely performance critical, using good load balancing and load control schemes can help in overall system performance.

**Distributed Web Servers** There is an increasing need for bigger and better web servers that can serve higher amounts of traffic. By using clusters of computers more traffic can be served. However, unless there is a good strategy on how to implement intra-cluster communication the same kinds of performance problems as in TINA can be expected.

### 3. MEASURING PERFORMANCE

This section contains a discussion about the concept of performance in distributed object oriented systems used for telecommunication applications.

Performance can be viewed from at least three perspectives, defined here in a very loose way:

**System user:** A user wants quick error-free responses from the system. Good performance is when a requested service is delivered without unacceptable delays. While the question of what is and what is not an acceptable delay is surely relevant, especially as this differs from system to system, performance from the system user perspective is the system's capability to deliver a service within an expected time frame.

**System operator:** 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 on one hand 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.

**System manufacturer:** The manufacturer of a system wants the system to have expected behavior at all times, even during heavy load.

In classic telecommunications a common measurement of performance is the call setup time, i.e. the time it takes from when the user requests a service until it is completely delivered. If a maximum acceptable time is set, it is easy to estimate performance by looking at the number of successful calls compared to the number of failed calls, where a failed call is a call that takes more time in setup than the given maximum acceptable time.

However, call setup times lose their meaning to some extent when more and more services are added. First, as services can differ greatly in complexity, it is difficult to have a set time that all services are required to follow. Another problem is that the services may not be easily divided into "atomic calls". For instance, in TINA there are no calls as such, instead as mentioned earlier we have the concepts of access, service and communication sessions. An access session is the entire unit of time that an user is "logged on" to the system. During an access session, a user can start one or more service sessions, each of which is an instance of a particular service, such as Video Conference. Each service session can then use several communication sessions, each of which represents one voice or video bit stream. An ordinary telephone call, where user A calls user B, would then consist of user A logging on and starting a access session. User A then starts a telephony service session which notifies user B that he has a call waiting. If user B accepts the call, then the service session sets up a communication session so that A and B can talk. While talking, both users can then start other service sessions to for example share a document or use a system database. In this scenario a call is something very complex and highly user and service dependent.

In an object oriented environment it might be tempting to measure the the round trip time for a function call made to another node. The round trip time is then the time it takes from the call is made until a reply is received. A problem with this approach is that the remote object might make calls to other objects in turn, and this makes round trip time highly dependent on load balancing and distribution, and does not say very much about the load status in the network.

One way could be to use special performance monitoring objects. These would make simple out and back function calls to surrounding nodes and measure the round trip time for these. If the monitor object has an intelligent way of making the calls the resulting communication and processing might not become overly heavy. One method might

be to issue queries at a rate proportional to the rate at which nodes are communicated with, thus having more up to date knowledge about the load level on nodes that are used often. A similar scheme has been used in Intelligent Networks (IN) load control (see Jennings *et al*<sup>8</sup>) where light weight intelligent agents (“ants”) are sent between the Service Switching Points and Service Control Points to measure round trip times and network load levels.

One reason to use special load monitor objects is that the implementations of underlying protocol layers have large impact on the performance of a system. The monitor objects would have a implementation independent view of the network and see the same delays and loads as the objects used in services do.

#### 4. A DISCUSSION OF LOAD BALANCING AND RELATED CONCEPTS

In this section we discuss various ways to achieve good performance in a system by using load balancing, load sharing and load control. Rumsewicz<sup>5</sup> notes that the two former are heavily used in the computer science research area, while the latter is common in teletraffic and telecommunications research. The reason for this division of research is that the systems under study have different expectations on them.

The different expectations placed on systems stem from different basic properties of computer communication traffic and telecommunication traffic. In computer communications every job must be finished, but some waiting is (usually) acceptable. However, in traditional telecommunications the real-time aspects are more important. If a offered job cannot be executed right away, it better be dropped as soon as possible to not waste any processing power that might allow the accepted jobs to be finished on time.

For distributed object oriented systems such as TINA to be used in telecommunications both load balancing and load control schemes must be used to ensure good performance. If a load balancing algorithm is good enough, then the load control algorithm is only required to save the system in emergency situations.

##### 4.1. Load Balancing and Load Sharing

Load balancing is a way to increase performance by distributing the offered load among available nodes, thus making the best possible use of the available resources. In distributed computer systems we want the load divided among the nodes in such a way that system response times are kept as low as possible. Two factors influence load balancing. First the object distribution, giving the physical location where each object is executed. Second the algorithm that is used to decide which object to use if there are more one available that can be used.

Object distribution can be a very difficult operation. It is largely an optimization problem, where the goal is to optimize system performance by placing the objects in such a way that system response time is kept low. The ideal choice would of course be to have instances of every object on every node. This is not practical in reality for a number of reasons. Objects may have specific requirements on the nodes on which they are executing. For instance, a database object might require extra memory capacity. At the same time, some objects might not require heavy processing therefore making the system cheaper by putting them on nodes with less capacity.

While objects usually can migrate between nodes in the network, it is highly likely that some kind of steady state can be said to hold during long intervals. When object locations are “fixed”, then the next problem is to chose between the different instances when one of them are called. If an instance on a heavily loaded node is selected, then the response time will probably be longer than if a node with less load had been chosen. The problem is of knowledge, how can the load balancing algorithm know which nodes are heavily loaded and which are not? It is not as simple as just to ask them, since the very question will require processing time at both this and the remote nodes, further adding to the load situation. By asking load status more seldomly, the communication overhead would be less. However, as load status on a node can change quickly, the load information may well be out of date by the time it is used.

One further factor which complicates things is that once a load balancing algorithm has chosen one object to use, that object will be used for the rest of the session. This means that the effects of a particular choice may have effect on the network not only now but also at later times. However, trying to predict these future effects is very difficult and requires much knowledge of internal traffic conditions.

## 4.2. Load Control

If load balancing is not enough to achieve good performance, and often it is not at times of heavy traffic, more drastic measures are needed. This is where overload control comes into the picture. In traditional telecommunication systems, overload can be handled by blocking newly arrived calls or possibly rerouting them to a less congested part of the system. However, in more modern systems this is not so simple. First, usage of resources is more flexible as service requirements can be very different, making it hard to decide for instance how many can be let into the system. Second, the number of objects involved in a single service request can be large and it might very well be that an object on a overloaded node is not reached until far into the service setup phase. If a service request is blocked late in the setup phase, then a lot of processor time has been wasted. Third, blocking has an economic side as well as the telecom operator might be more interested in blocking the low income service types than the ones that generate huge revenues.

In telecommunications, load control is introduced to save the system from overload conditions, that is to maintain system sanity in the case of massive offered load. So while load balancing makes the system perform well, load control guarantees system survival, a requirement in real-time systems such as those used in telecommunications.

Load control can be of two kinds, either internal or external. Internal load control is local to a single node, and prevents that node to be overloaded by blocking incoming calls. External load control tries to look at the performance of the entire network by blocking traffic arriving at the entire network. While internal protection does help, it is by no means sufficient to support good performance for the entire system. External load control is more difficult to implement, and usually requires some communication to be efficient. In a distributed system a combination of internal and external load control is ideal.

## 4.3. Properties of Good Performance Algorithms

Rumsewicz<sup>5</sup> discusses a number of requirements that should be fulfilled by load control and load balancing systems. He identifies the following areas, slightly abbreviated:

1. Making no *a priori* use of relative processor performance, it is up to the algorithm to adjust for these.
2. Making no *a priori* assumptions about the mix of offered services or their different requirements on processing power.
3. Making no *a priori* assumptions about placement of functionality in the network.
4. Does not require synchronization of the control action undertaken by each processor in the system.
5. Incorporates telecommunications practices such as giving preference to calls which have had significant processing spent on them over newly arrived calls.
6. Attempts to maximize the call completion rate instead of maximizing the number of tasks performed by the system.
7. Is simple to implement and therefore more likely to be practically useful.

To the requirements given by Rumsewicz, we would like to add the following:

8. A high level of fault tolerance. If a fault occurs in the network, the remaining nodes must quickly be able to reconfigure so that the system does not go out of control.
9. Provide protection both on the node level and on the network level.
10. For load balancing it is important that the algorithm makes good routing choices, meanwhile it is equally important that it does not make very bad routing choices.
11. The algorithm should have minimum overhead. This concerns both ease of computation (the simplicity covered under 7 above) and little inter-node communication.



12. The algorithms must be fair. By fairness we mean that services shall not be starved of resources on a common basis.
13. The algorithm should be application transparent. Some communication between application and control is necessary, for instance to prevent the application from making repeated attempts to contact a congested node. The way that the algorithms are implemented should not effect the application.

It is obvious that adaptability is one of the main requirements, whether it is adaptability to changing physical conditions, such as adding new processing capabilities, or changing traffic patterns, such as the impact of the introduction of new services.

## 5. SIMULATIONS

In this section we describe some simulations performed using the described simulation model. The objectives were to study the impact of different object distributions and the use of different load balancing algorithms. Two algorithms are studied: random and ant-based.

### 5.1. Algorithms

We shall first describe the two algorithms used in the simulations.

#### 5.1.1. Random Load Balancing Algorithm

When there are several nodes which have the same type of object, a node is chosen randomly with all nodes having equal probabilities.

#### 5.1.2. Ant Based Load Balancing Algorithm

In the ant based algorithm special objects on all nodes make queries to the surrounding nodes at random intervals. The round trip times for the queries are collected. The assumption is that the loads on a neighbor node is inversely proportional to the round trip time for an ant sent to that node. The algorithm uses weights  $W_i$  to decide what the routing probabilities will be.

The following procedure is used for the ant jobs (note that the limiting values given are for the simulations described later):

1. An ant job is created on node *origin* and is put in its queue for protocol wrapping.
2. The ant job is sent to node *target*, where it receives first protocol unwrapping and then immediate protocol wrapping.
3. The ant job is sent back to *origin* where it receives its final protocol unwrapping.
4. The round trip time  $t_{rtt}$  is measured from the time the ant job was created until it has its final protocol unwrapping back at node *origin*.
5. The weight for node *target* calculated at node *origin* is updated by low pass filtering the inverse of  $t_{rtt}$ :

$$W_{target,new} = 0.9 * W_{target,old} + 0.1 * \frac{1}{t_{rtt}}$$

Boundary values for  $W_{target}$  are  $10 < W_{target} < 400$ .

Ants are generated by the nodes at random intervals. The length of the intervals are drawn from an exponential distribution with mean value 0.2 seconds. The reason for using random intervals is to spread out the ants over time evenly. The target node for an ant is selected using a simple round-robin scheme.

The routing table in every node is recalculated every second. If  $J$  is the set of nodes that has object  $j$ , then the probability  $P_i$  of selecting node  $i$  is:

$$P_i = \begin{cases} \frac{W_i}{\sum_{k, k \in J} W_k} & \text{if } i \in J \\ 0 & \text{otherwise} \end{cases}$$

Note that no ants are sent by a node to itself, it can therefore not calculate a weight  $W_{self}$  for itself. In the following simulations  $W_{self}$  was fixed to  $W_{self} = 300$ .

## 5.2. Simulation Parameters

Kihl *et al*<sup>7</sup> describes a simulation model where the TINA equivalent of POTS (Plain Old Telephony Service) was studied. Due to the complexity of TINA we have instead decided to look at a simpler scenario which bears similarity to TINA-based access and service sessions.

### 5.2.1. Network

A four node network was used. All nodes had the same relative speeds. The protocol wrapping and unwrapping times were set to 1ms.

### 5.2.2. Objects and service description

Four object types were used in the simulations, labeled A to D. Execution times for each object type is in the table below.

<i>Object</i>	<i>Execution time</i>	<i>Service steps</i>
A	4ms	1, 3, 5, 9
B	20ms	2
C	4ms	4, 6, 8
D	20ms	7

The sample service used in the simulations has nine steps, see figure 4.

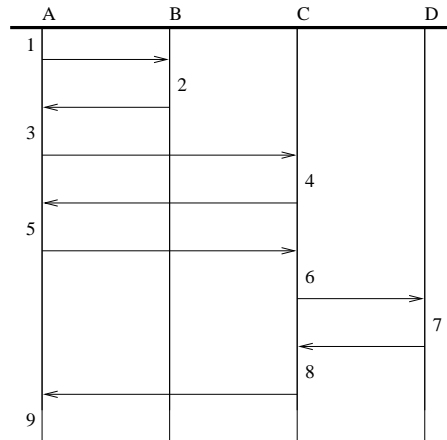


Figure 4. Sequence chart for simulated service

### 5.2.3. Object distributions

We studied one balanced and one unbalanced case. The table below describe how the objects were distributed in each case.

Balanced distribution		Unbalanced distribution	
<i>Node</i>	<i>Objects</i>	<i>Node</i>	<i>Objects</i>
1	A and C	1	A and C
2	A and C	2	A
3	B	3	B and C
4	D	4	D

#### 5.2.4. Arrival process

The arrival process to the system is a Poisson process. The arrival rate is 28 new service requests in average each second.

### 6. RESULTS FROM SIMULATIONS

We here present four simulation cases using the parameters given above.

<i>Case</i>	<i>Algorithm</i>	<i>Distribution</i>	<i>Setup Time</i>
A	Random	Balanced	0.148ms
B		Unbalanced	0.330ms
C	Ant based	Balanced	0.151ms
D		Unbalanced	0.225ms

All simulations were long enough to have small confidence intervals.

It is obvious that different object distributions have an impact on performance, largely due to the intensive inter-node communication needed to set up a service. While delays imposed by communication can be decreased by efficient implementation of communication protocols, they cannot completely be eliminated and it is therefore important to keep communication costs low. Thus it seems that good object distribution mechanism is an important feature in these systems if good performance is to be achieved.

In a balanced distribution the random algorithm (Case A) works well, however algorithm fails completely when the object distribution is unbalanced (Case B). This is reasonable since the algorithm does not use any knowledge about the nodes relative load levels, thus not being able to adapt to an unbalanced distribution.

The ant based algorithm shows no performance gain in the balanced case (Case C), as its adaption ability is of no use, even causing a slight disadvantage by generating communication overhead. Depending on how the node sets the weight on itself ( $W_{self}$ ) it may or may not route more jobs to itself, thereby gaining some advantage in lower communication costs. To tune  $W_{self}$  by hand is difficult and using an adaptive algorithm to do this would be preferable. In the unbalanced case (Case D) the performance gain is obvious.

While the ant based algorithm shows promise, it also has the drawback of needing to tune several parameters. These include how often ants are to be generated, the ratio between weight and round trip time, how often to recalculate the routing tables, the node's own weight, if the network is large, which nodes to send to, etc.. If these problems can be solved the algorithm might give valuable information of how the system behaves from the objects viewpoint.

### 7. CONCLUSIONS AND FUTURE WORK

In this paper we have discussed performance issues in distributed object oriented networks. We have pointed out several areas where work needs to be done if performance bottlenecks are to be avoided in these kinds of systems. We have collected a set of requirements that can hopefully help with overcoming these bottlenecks. We have also developed a simulation model for these systems and described how it can be used to simulate TINA and similar systems.

The model was used to simulate two simple load balancing algorithms, random and ant based. The ant based algorithm is adaptable and sees the network as the objects see it. Therefore it might therefore be useful for performance related tasks in the system.

One area for future work is further research in how to make the ant based algorithm more intelligent and examine how it can help in both load balancing and load control.

## REFERENCES

1. R. Jain, *Congestion Control in Computer Networks: Issues and Trends*, IEEE Network Magazine, vol 4, No 3, May 1990.
2. T. Kunz, *The Influence of Difference Workload Descriptions on a Heuristic Load Balancing Scheme*, IEEE Transactions on Software Engineering, vol 17, no 7, July 1991.
3. O. Kremien and J. Kramer, *Methodical Analysis of Adaptive Load Sharing Algorithms*, IEEE Transactions on Parallel and Distributed Systems vol 3, no 6, November 1992.
4. 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.
5. M. Rumsewicz, *Load Control and Load Sharing for Heterogeneous Distributed Systems*, 16th International Teletraffic Congress, Edinburgh, Scotland 1999.
6. M. Kihl, *On overload control in a TINA network*, 6th IEE Conference on Telecommunications, Edinburgh, Scotland, 1998.
7. M. Kihl, N. Widell and C. Nyberg, *Load balancing algorithms for TINA networks*, 16th International Teletraffic Congress, Edinburgh, Scotland 1999.
8. B. Jennings and Å. Arvidsson, *Cooperating Market/Ant based Multi Agent Systems for Intelligent Network Load Control*, To appear at the IATA Workshop, Stockholm, August 1999.
9. M. Chapman and S. Montesi, *Overall Concepts and Principles of TINA*, TINA-C, www.tinac.com, 1995.
10. L. Kristiansen (editor), *Service Architecture*, TINA-C, www.tinac.com, 1997.
11. P. Graubmann and N. Mercouroff (editors), *Engineering Modelling Concepts (DPE Architecture)*, TINA-C, www.tinac.com, 1994.
12. S. Vinoski, *CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments*, IEEE Communications Magazine, February 1997.