Simulation of a distributed CORBA-based SCP

McArdle, Conor; Widell, Niklas; Nyberg, Christian; Lilja, Erik; Nyström, Jenny; Curran, Tommy

Published in:
Telecommunications and IT convergence : towards service evolution : proceedings (Lecture notes in computer science)

2000

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

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.

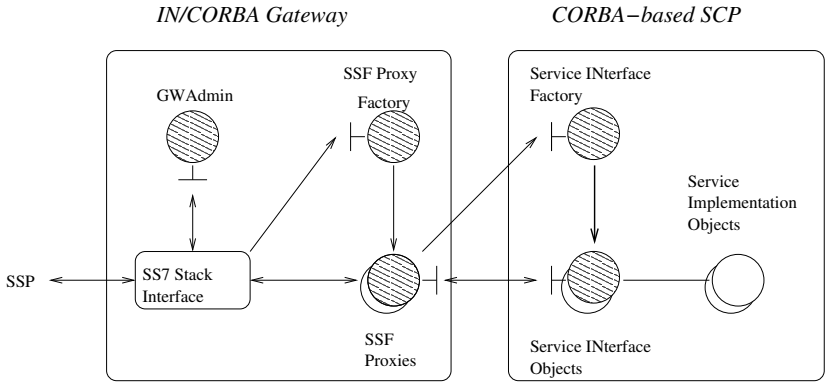

Figure 1: Main elements of the IN/CORBA Interworking Gateway

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.

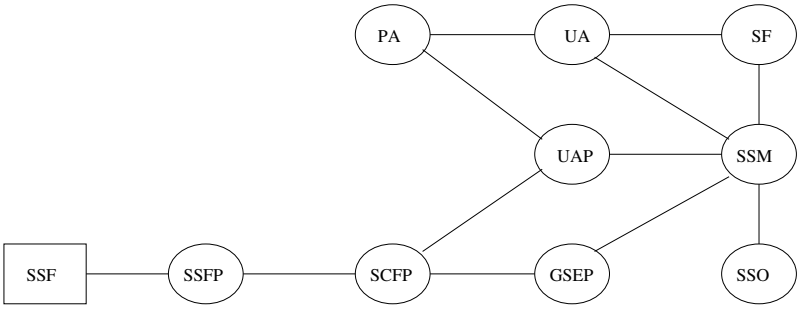

Figure 2: Computational Objects and the interactions required for a VPN Service

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.

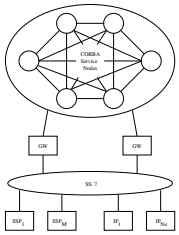

Figure 3: Network Scenario for CORBA-based SCP

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 required 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.

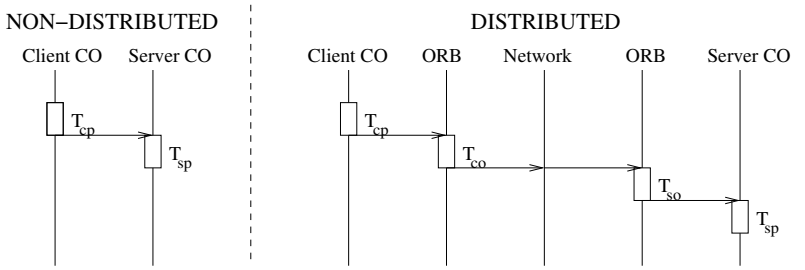

Figure 4: Execution times for messages passed between COs. In the right hand figure, two COs are executing on different processors (i.e. COs are distributed). The processing times T_{CP} (service processing time) and T_{CO} (marshalling time) give the total processing time at the client node associated 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}.

Marshaling, demarshaling 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.

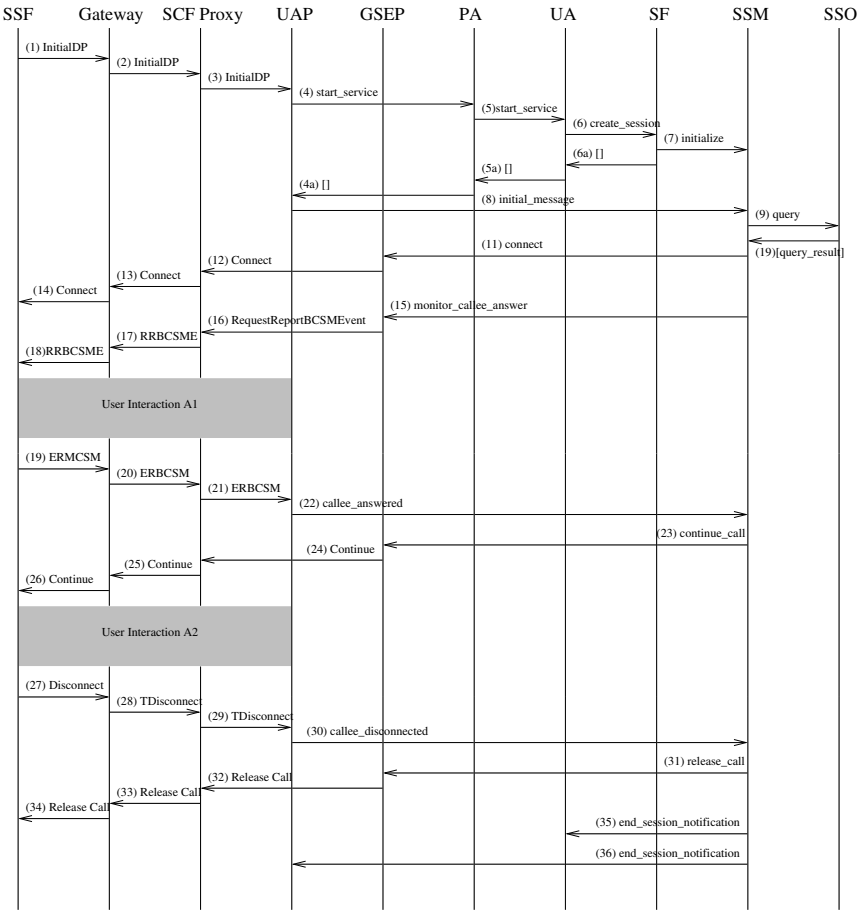

Figure 5: Message Sequence Chart for Test Service A

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.

Simulation Parameters	
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