Using Analytic Models for Predicting Middleware Performance

Dorina Petriu[1] Hoda Amer[2] Shikharesh Majumdar[1] Istabrak Abdull-Fatah[2]

[1] Dept. of Systems and Computer Eng.
Carleton University
Ottawa, ON, Canada, K1S 5B6
{petriu|majumdar}@sce.carleton.ca

[2] Nortel Networks
3500 Carling Avenue
Nepean, ON, Canada K2H 8E9
{hoda|istabrak}@nortelnetworks.com

ABSTRACT

The client-server paradigm is very popular in building distributed computing applications. Heterogeneity is natural in client-server systems, where components implemented using different technologies must interact and collaborate with each other. Inter-operability is provided through *middleware* on such a heterogeneous system. *Common Object Request Broker Architecture (CORBA)* is a standard for middleware-based distributed object computing systems. This paper focuses on analytic modeling of middleware-based systems that can be used in the performance engineering of CORBA-based client server applications. The paper applies the Layered Queuing Network model to two types of middleware architectures: *handle-driven* and *forwarding*. The model outputs are compared with measured values to determine the accuracy of the modeling technique.

Keywords

Analytic performance model, CORBA, client-server performance, Layered Queuing Networks.

1. INTRODUCTION

Concurrency, reliability and reusability are well-known attributes of *Distributed Object Computing (DOC*) systems. Heterogeneity is natural in DOC systems. When an existing system is upgraded or when new features are added to an embedded application, the new components are often implemented using a different technology in comparison to the legacy components. In order to remain economically viable it is crucial to reuse the existing components. Inter-operability is provided through *middleware* on such a heterogeneous system. *Common Object Request Broker Architecture (CORBA)* is a standard for such middleware-based DOC systems.

Scalability and latency are desirable properties of most distributed systems and are crucial for the proper functioning of performance demanding applications that include various telecommunication products. Most current practices in software design and implementation are based on a "design now and fix performance later" approach. That is, the functional design and implementation of the system are done first and the performance techniques are retrofitted at a later point in time. In many situations the prototype fails to meet the performance requirements resulting in an expensive redesign of the system. Software performance engineering techniques that advocate the integration of performance analysis with various steps in system design and implementation have been proposed [11]. Analytic performance models are often used in software performance engineering because of its lower cost in comparison to simulation and measurement-based approaches. Analytic models are also used in system selection studies and in capacity planning [7]. This paper focuses on analytic modeling of middleware-based systems that can be used in the performance engineering of CORBA-based client server systems.

Queueing network modeling is a popular and widely used technique for predicting the performance of computing systems. Although queueing network models have been successfully used in the context of traditional time sharing computers, they often fail to capture complex interactions among various software and hardware components in client-server distributed processing systems. Modeling techniques such as Method of Layers [10] and Layered Queueing Networks (LQN) [12], [9] are proposed for handling such complex interactions. We have used the LQN modeling technique in this paper for analyzing the performance of CORBA-based middleware systems. We have developed LQN analytic models for two different types of middleware interaction architectures that are discussed in the next section. Based on a Commercial-Off-The-Shelf (COTS) middleware product called Orbeline [8] (currently sold as Visibroker by Inprise) and a synthetic workload running on a network of Sun workstations using Solaris we have implemented performance prototypes for client-server systems. Using Solaris system calls buried into the prototype software we have measured system throughput and response times. The measured values are compared with the model outputs to determine the accuracy of the modeling technique.

The paper is organized as follows. The middleware interaction architectures and the synthetic client server applications are described in the following section. In order for the paper to be self-contained, the LQN modeling features used in the paper are briefly presented in Section 3. LQN analytic models for two types

of middleware-interaction architectures are described in section 4. A comparison of the measured results and the output of the analytic models is given in Section 5. Section 6 presents our conclusions.

2. MIDDLEWARE INTERACTION ARCHITECTURES

One of the basic entities in a middleware system is one that maps the name of an object to an object reference or handle and is referred to as an agent in this paper. This handle is then used to invoke the method in the appropriate server object. Middleware interaction architectures refer to the way a request from the client is routed to the server. In a *Handle-Driven* (HORB) architecture when a client wants to request a service it sends the server name to the agent in the middleware system. The agent performs a name to object reference (IOR) mapping and sends a handle back to the client. The client uses the handle to contact the server and receive the desired service. A number of COTS ORBs that include Visibroker [3] and Orbix-MT [6] use such a handle-driven architecture. In a *Forwarding* (F-ORB) architecture the client sends the entire service request to the agent that locates the appropriate server and forwards the request to it. The server performs the desired service and sends a response back to the client. A detailed discussion of the performance of these architectures is presented in [2]. The H-ORB used in this research is the basic Orbeline product whereas the F-ORB is built using additional processes in conjunction with the Orbeline middleware [1].

2.1 Synthetic Application

A simple client-server application is built, in which two distinct services are using the ORB. A client executes a cycle repeatedly. In each cycle it makes one request to Server A and one to Server B. It performs a *bind* operation before every request. A synchronous communication mechanism is used: after a request is made, the client remains blocked until the response is received. When a service is requested form a particular server the server process executes a loop and consumes a pre-determined amount of CPU time. The synthetic application is used because it provides flexibility in experimentation with various levels of different workload parameters such as the service time at each server, and inter-node delay. Such an application is appropriate for investigating the accuracy of the analytic models discussed in this paper. Two copies of A called A1 and A2 as well as two copies of B called B1 and B2 are provided. The two copies of each server enable the system to handle more load and allow us to investigate the impact of load balancing that is provided by many commercial ORB products. Experiments were performed by executing the system on a local area network of Sun workstations running under the Solaris 2.6 operating system. Each of the four server processes is run on a separate machine. The agent is run on a separate workstation, and the remaining workstations are used for running the client processes. The synthetic application is characterized by a number of parameters that are briefly summarized.

Service Demands (SA, SB): The time required by server A and server B respectively to provide the requested service. Whenever a particular server A (or B) is invoked it consumes SA (or SB) units of CPU time.

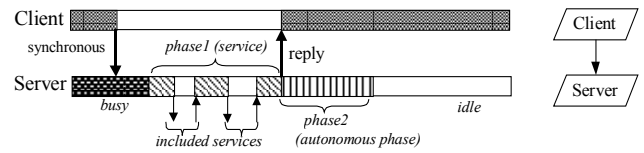
Inter-Node Delay (D): Clients, the agent, and servers may be separated from each other by a number of intermediate nodes. The inter-node delay between a client and a server, between a server and the agent, as well as between a client and the agent are characterized by D. In our experiments, a sender process sleeps for D units of time before sending a message in order to simulate an inter-node delay due to the store and forward operations performed by the intermediate nodes. However, in case of the H-ORB agent we did not have access to the source code, and the inter-node delay for the handle returning operation was simulated by making the client sleep for D units of time before receiving the message.

Message Length (L): The size of the message, L, sent by the client for providing a method name along with its argument list or sent by the server to return the results.

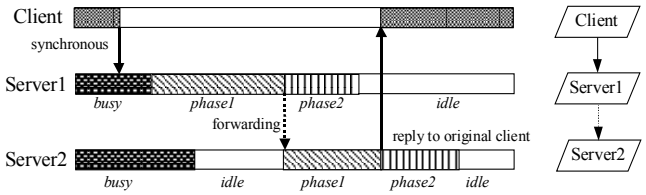
The measurement of end-to-end performance measures, such as response times and throughput, specific for different combinations of SA, SB, L and D, were repeated long enough to produce confidence intervals less than $\pm$5% of the mean at a confidence level of 95% for the performance measure of interest.

3. LQN MODEL

LQN was developed as an extension of the well-known Queueing Network (QN) model, at first independently in [12] and [10], then as a joint effort [4]. The LQN toolset presented in [4], [5] includes both simulation and analytical solvers that merge the best previous approaches. The main difference between LQN and QN is that a server to which customer requests are arriving and queueing for service may become a client to other servers, thus giving rise to nested services. An LQN model is represented as an acyclic graph whose nodes (named also *tasks*) are software entities and hardware devices, and whose arcs denote service requests (see Fig. 2 and 3). The word *layered* in the name of LQN does not imply a strict layering of the tasks (for example, a task may call other tasks in the same layer, or skip over layers). The tasks are drawn as parallelograms, and the devices as circles. The nodes with outgoing and no incoming arcs play the role of *pure clients* (named also *reference tasks*, as they drive the system). The intermediate nodes with incoming and outgoing arcs play both the role of client and server, and the leaf nodes are *pure servers*. A software or hardware server node can be either a single-server or a multi-server (composed of more than one identical clones that work in parallel and share the same request queue). An LQN task may offer more than one kind of service, each modeled by a so-called *entry* drawn as a parallelogram "slice", (see *F-agent* in Fig. 3 for an example). An entry has its own execution times and demands for other services (given as model parameters). Although not explicitly illustrated in LQN notation, each server has an implicit message queue, where the incoming requests are waiting their turn to be served. Servers with more then one entry still have a single input queue, where requests for different entries wait together. The default scheduling policy of the queue is FIFO, but other policies are also supported.



a) LQN synchronous message



b) LQN forwarding message

Figure 1. Synchronous and forwarding LQN requests

There are three types of LQN request messages: synchronous, asynchronous and forwarding. Fig.1 illustrates the first and the last type, which are used in this paper. A *synchronous* message represents a request for service sent by a client to a server, where the client remains blocked until it receives a reply from the provider of service (see Fig.1.a). If the server is busy when a request arrives, the request is queued. After accepting a request for one of its entries, the server starts to process it by executing a sequence of one or more phases of that entry. At the end of *phase 1*, the server replies to the client, which is unblocked and continues its work. The server continues with the following phases, if any, working in parallel with the client, until the completion of the last phase. After finishing the last phase, the server begins to serve a new request from the queue, or becomes idle if the queue is empty. During any phase, the server may act as a client to other servers, asking for "included services". The *forwarding message* (represented by a dotted request arc) is associated with a synchronous request that is served by a chain of servers, as illustrated in Fig. 1.b. The client sends a synchronous request to Server1, which begins to process the request. At the end of its phase1, it forwards the request to Server2. Sever1 proceeds normally with the remaining phases in parallel with Server2. The client, however, remains blocked until the forwarded request is served by Server2, which replies to the client at the end of its phase 1. A forwarding chain can contain any number of servers, in which case the client waits until it receives a reply from the last server in the chain. A phase may be "deterministic" or "stochastic", and is subject to the following assumptions:

a) The total CPU demand of a phase (whose mean is given as a parameter) is divided up into exponentially distributed slices, each of which is delimited by a request to lower level servers. The mean execution time is the same for all slices.

b) Requests to lower level servers are geometrically distributed with a specified mean (given as a parameter) in a stochastic phases, and occur for a fixed number of times in a deterministic phase.

4. LQN MODEL FOR H-ORB AND F-ORB ARCHITECTURES

Fig. 2 represents the LQN model for the H-ORB architecture. The top node named *Clients* contains the reference tasks driving the system. The multiplicity of the node represents the population of the model, and is a parameter in our experiments. Each client loops endlessly and sends synchronous requests to the lower level tasks: 2 requests per cycle to the Agent, one to a server A and one to a server B. Each client task runs on its own CPU (the node *CPUc* which is a multi-server). The system contains two identical servers of type A, and two of type B, each with its own queue. Therefore, each server is modeled as a separate task.

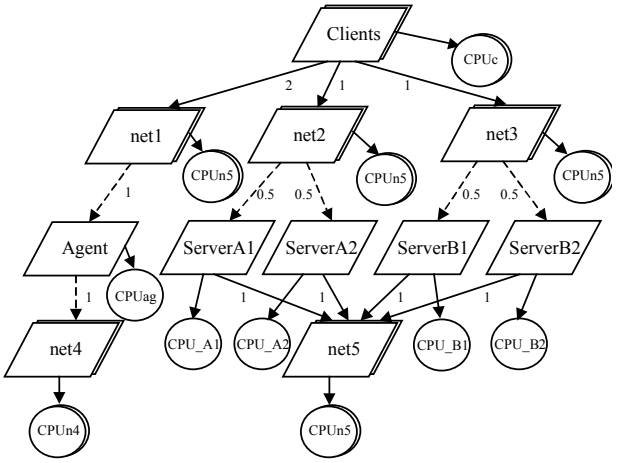


Figure 2. LQN model of the H-ORB architecture

The measured system was built such that, before sending a request to the Agent or a server, a client "sleeps" for a duration D to simulate the inter-node delay. In the LQN model, this delay is represented by the tasks *net1 to net5* (which are infinite servers). Note that delay servers such as *net1* are LQN tasks and execute on their own CPUs. Therefore, a request/reply sequence between a client and the agent follows a forwarding chain: the client sends a synchronous request to *net1* and blocks; *net1* forwards it to *Agent*, which forwards it to *net4*. The client remains blocked until the arrival of the reply from *net4*, the last task on the forwarding chain. A similar request path is used between the client and a server, with two differences. First, the network delay server *net2* (*net3*) chooses each server randomly with a probability of 0.5. This models the behaviour of the load balancing agent. The second difference is that the servers call *net5* synchronously, whereas the agent forwards to *net4*. This corresponds to the behaviour of the system that was implemented and measured: the servers were built to sleep before sending the reply to the client, whereas the agent (a COTS component) did not wait artificially for a network delay.

In this model, each task has a single entry, as it offers a single service, and each entry has a single phase. All phases are "deterministic", which means that the number of requests given on each arc specifies the exact number of requests performed. For example, a client makes exactly two requests to the agent, one to either A1 or A2, and one to either B1 or B2. However, the order of these requests is random in the LQN model, as opposed to the

measured system where the calls were made in a fixed order. Another difference between the real system and the LQN model concerns the CPU demand distribution. In the measured system, all execution times and network delays are deterministic, whereas in the LQN model the execution times between two requests are exponentially distributed.

In a *Forwarding (F-ORB)* architecture the client sends the entire service request to the agent that locates the appropriate server and forwards the request to it. The server performs the desired service and sends a response back to the client. The forwarding architecture was built by using an additional process named *F-agent* in conjunction with the Orbeline agent [1]. *F-agent* accepts a request from the client, calls the default COTS agent (named *D-agent*) to get the server handle, then sends the requests to the chosen server. *F-agent* (which was later cloned) is running on the

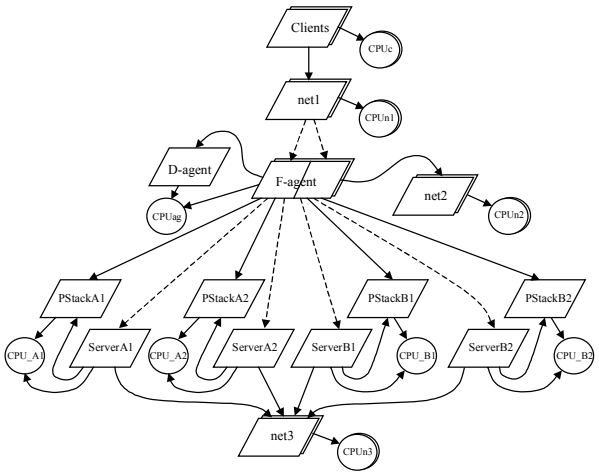


Figure 3. LQN model of the F-ORB architecture

same CPU as *D-agent*. The LQN model is given in Fig. 3.

The communication between *F-agent* and servers deserves some discussion. First, *F-agent* sleeps D units of time before sending the message to the server, after which it sends the message pseudo-asynchronously. This means that *F-agent* is blocked until the message is received on the server side, but is not waiting for the server to perform its service. The LQN model contains for each server an additional task *PStack* running on the server's CPU. It models the protocol stack process responsible for receiving and sending messages. In the LQN model, the sending of a message by *F-agent* to a server is decomposed in three steps: i) a synchronous request from *F-agent* to *net2* for the inter-node delay; ii) a synchronous request from *F-agent* to *PStack* of the designated server; and iii) a forwarding request from *F-agent* to the server. The communication between a server and a client is also modeled in three steps: i) a synchronous request from server to *net3* for the inter-node delay; ii) a synchronous call from the server to its *Pstack* process for the execution of a send operation; and iii) the replay sent back by the server to the client (due to the fact that the server is the last in the forwarding chain). It must be noted that in the H-ORB model there was no need to represent the processor stack separately; its CPU demand for each receive and send operation was included in the CPU demand of the server process instead. *F-agent* has two entries in order to capture more closely the behaviour of the measured system: one entry chooses between the server *A1* and *A2*, the other between the server *B1* and *B2*. Each entry is represented in the graph by a parallelogram "slice". The incoming requests targeting a given entry arrive at the top of the parallelogram slice, and the outgoing requests leaving a given entry depart from the bottom. The arcs leaving from the sides (to *D-agent*, *net2* and *CPUag*) represent requests that are common to both entries.

The *F-agent* node is an LQN multi-server because the effect of cloning the *F-agent* is studied in some of the experiments. As in the case of the H-ORB model, each entry has a deterministic first phase only.

5. COMPARING THE MEASUREMENTS AND MODEL RESULTS

As mentioned previously, the synthetic application that was built and measured is characterized by the following factors: service demands (*SA, SB*), inter-node delay (*D*) and message length (*L*). We have conducted experiments that varied a single factor at a time. The results of the analytic model are compared in this section with measured values.

a) The effect of service demands. Fix D=10 ms, L=150 bytes and vary the service demands (SA, SB): {(10, 15), (50, 75) and (250, 375)} ms. The measured response times and those predicted by the model are given in Fig.4. As expected, the response time grows with the service demands. The errors are quite low for small demands, but grow up for a large number of clients, reaching –18% in the case where SA, SB take the largest values. The analytic model tends to predict shorter response times, especially when the servers are strongly saturated. We believe that the cause is a "convoy effect" that appears in the real system due to its strong deterministic behaviour. The client processes follow the same order of visits, and the service demands and delays are all deterministic. The clients move "in convoy" through the system, and thus the queueing delays are larger. The LQN model does not capture this effect due to its assumptions that the service times are exponentially distributed and the order of visits is random. We have performed a few simulation experiments that confirm this hypothesis.

b) The effect of inter-node delay. Fix (SA, SB) at (10, 15) ms, L=150 bytes and vary D: {250, 500, 1000} ms. The model and measured response times are given in Fig.5. As expected, the response time depends strongly on the inter-node delay. A 96.8% increase in response time is observed when D doubles from 250 to 500 ms, and a 290% increase when D quadruples from 250 to 1000 ms. Even for large populations, the errors are quite low for smaller D (–2.26% error for D=250 ms), and grow a little for larger D (–5.5% for D=1000 ms).

c) The effect of message size. Fix (SA, SB) at (10, 15) ms, D = 200 ms, and vary L: {150, 4800, 9600, 19200} bytes. The overheads for sending/receiving the messages, and for marshaling/unmarshaling the parameters grow with the message size, which leads to an increase in the response time. However, the increase is not substantial (only 11.9% increase in response time when L changes from 150 to 19200 bytes, for large populations). Fig. 6 shows the response time predicted by the model as a function of number of clients for all the L values, but only one measured case, due to the fact that the curves were too close. The errors are small (less than 1%).

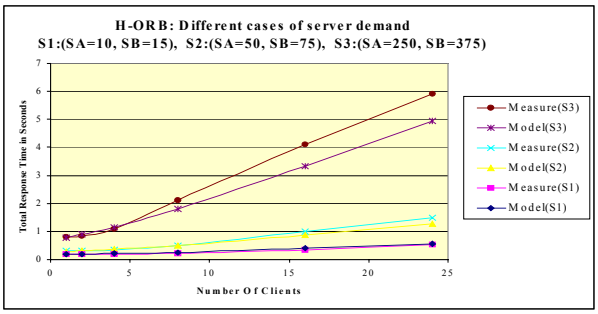


Figure 4. H-ORB: Measured and model response time Vs load for different server demands

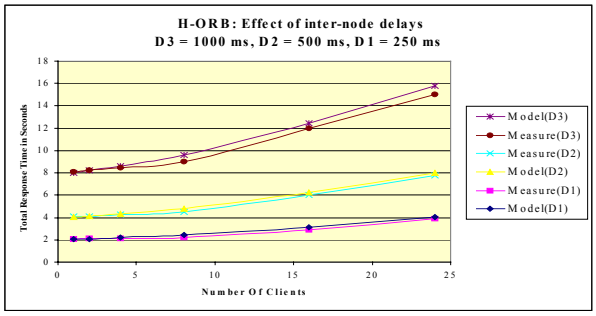


Figure 5. H-ORB: Measured and model response time Vs load for different inter-node delays

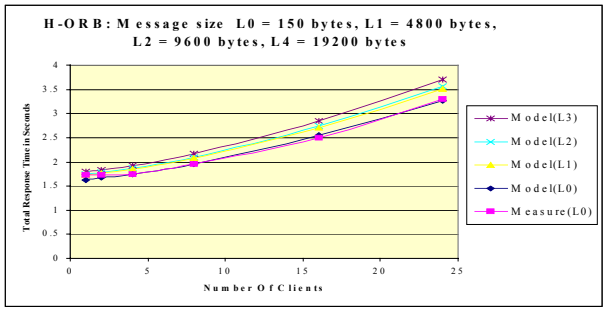


Figure 6. H-ORB: Measured and model response time Vs load for different message sizes

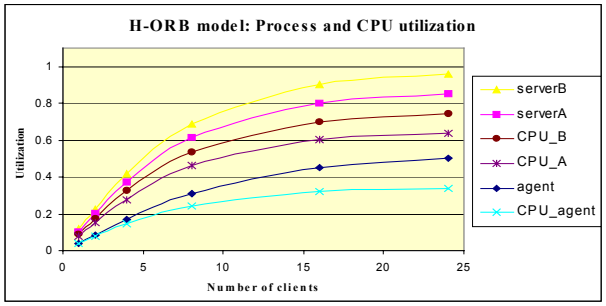


Figure 7. H-ORB: process and CPU utilization for SA=10, SB=15 D=10, L=150

Fig.7 shows the utilization of different processes and processors for the base case (SA=10ms, SB=15ms, D=10ms, L=150 bytes). The server processes B and A have the highest utilization (95.7% and 85%, respectively). Their dedicated CPUs have a lower utilization (74.4% and 63.8%, respectively). The agent and its processor have the lowest utilization. The system is characterized by a software bottleneck, where the CPUs are not used at their full capacity because some of the software servers approach saturation faster. We consider this a mild case of software bottlenecking because the difference between the software server utilization and their CPU is quite small. A more severe case of software bottlenecking (where all the CPUs are utilized at a very low level) is illustrated in Fig.10 and will be discussed in the next subsection.

5.1 The F-ORB model

In the case of F-ORB, there is another factor to consider: the cloning degree C (i.e., the number of copies) for the *F-agent* process. It has an important effect on performance because it removes a strong software bottleneck that appears for C=1, as explained below.

a) Effect of F-ORB cloning degree. Fix the service demand (SA=10 ms, SB=15 ms), the inter-node delay (D=10 ms) and the message size (L=150 bytes) and vary the *F-agent* cloning degree C: {1, 4, 8}. The measured response times and those predicted by the model are shown in Fig.8. The response time for 24 clients is cut by 49% when the number of clones goes from 1 to 8, even though no hardware resource is added to the system. This is due to the fact that at C=1 the system exhibits a strong software bottleneck.(see Fig.10). *F-agent* reaches saturation very early, for N=4 clients, preventing the hardware resources from being utilized at a reasonable level (*CPUagent* is utilized 54.8%, *CPU_B* only 26% and *CPU_A* only 22%). By adding *F-agent* copies to the system, the software bottleneck is removed and moves to hardware. At C=8 and 24 clients, the utilization of all CPUs is almost doubled (*CPUagent* is utilized 100%, *CPU_B* 53.5% and *CPU_A* 44.7%) and the system bottleneck moves to *CPUagent* (see Fig.11). As in the case of the H-ORB, the F-ORB model also tends to predict lower response times than the measured values, due to the same reasons (i.e., deterministic behaviour of the real system). At large populations, the error is -12.8% for C=1, -12.2% for C=4 and –3.7% for C=8. The model correctly identifies the system bottleneck.

b) Effect of message length. Fix the service demand (SA=10 ms, SB=15 ms), the inter-node delay (D=200 ms) and the number of clones (C=8) and vary the message size L: {4800, 9600, 19200) bytes. As in the case of the H-ORB, the overhead for sending/receiving the messages, and for marshaling/ unmarshaling the parameters grows with the message size. However, the response time increase is not substantial (only 2% increase when L doubles from 4800 to 9600 bytes, and 5.6% increase when L quadruples from 4800 to 19200 bytes). Fig. 9 shows the model response time Vs. the measured values L=9600. At a large population, the errors grow with the message size (from -6.22% for L=4800 bytes to -14.6% for L=19200 bytes).

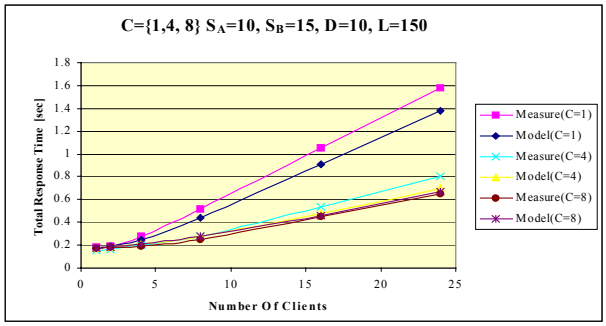


Figure 8. Measured and model response time Vs load for different cloning degrees of F-agent (C=1, 4, 8)

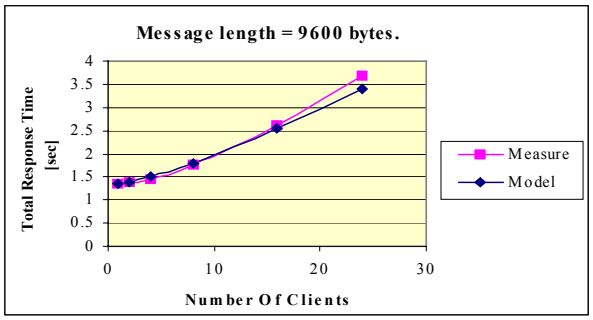


Figure 9. Measured and model response time Vs load for a message size of 9600 bytes

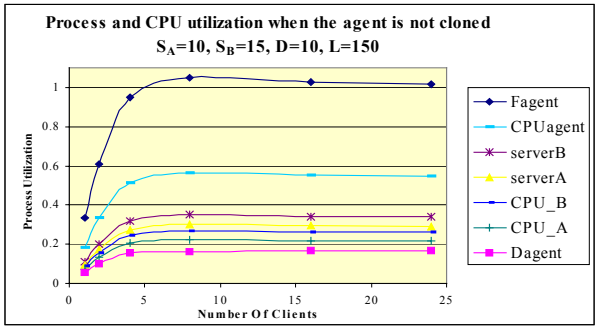


Figure 10. Software bottleneck example: F-ORB with a single F-agent copy

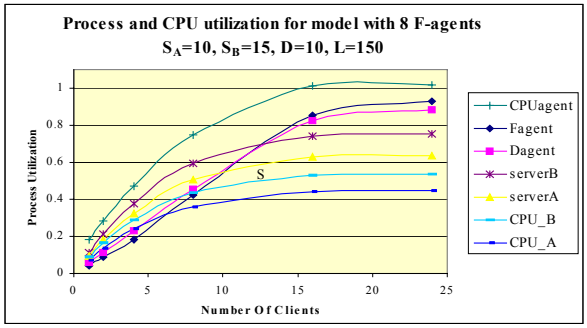


Figure 11. Hardware bottleneck example: F-ORB with eight F-agent copies

6. CONCLUSIONS

This paper focuses on analytic modeling of middleware-based systems that can be used in the performance engineering of CORBA-based client server applications. The paper applies the Layered Queuing Network model to two types of middleware interaction: *handle-driven* and *forwarding* architectures. We have found that the LQN model predictions are reasonably acceptable, with most of the errors less than 12%. The bottleneck in the system was also correctly predicted. The largest error of –18% occurred when both the service demand at the servers and the population was large. In such cases, the analytic model tends to predict shorter response times because it cannot capture a "convoy effect" that appears in the real system due to its strong deterministic behaviour (the processes follows the same pattern of requests and the service demands are deterministic). The LQN model cannot capture this effect due to its assumptions that the service times are exponentially distributed and the order of visits is random. Despite those differences, the LQN model can be quite useful in the performance engineering of CORBA-based application, due to its speed, facility of use and reasonable accuracy.

ACKNOWLEDGMENTS

This work was partially supported by grants from the Natural Sciences and Engineering Research Council of Canada (NSERC) and Communications and Information Technology Ontario (CITO).

REFERENCES

[1] Abdul-Fatah, I., "Performance of CORBA-based client-server architectures", Master Thesis, Carleton University, Dept. of Systems and Computer Engineering, (June 1998).

[2] Abdul-Fatah, I., Majumdar, S., "Performance Comparison of Architectures for Client-Server Interactions in CORBA", *Proc. IEEE 18th Conf. on Distributed Computing Systems*, 2-11, (May 1998).

[3] Borland Inprise, "Visibroker: CORBA Technology from Inprise", http://www.borland.com/visibroker, (1999).

[4] Franks, G., Hubbard, A., Majumdar, S., Petriu, D., Rolia, J., Woodside, M., "A toolset for Performance Engineering and Software Design of Client-Server Systems", *Performance Evaluation*, 24(1-2), 117-135, (Nov. 1995).

[5] G. Franks, "Performance Analysis of Distributed Server Systems", Ph.D. Thesis, Carleton University, Ottawa, (December 1999).

[6] Iona Technologies, *Orbix Programmers' Guide*, Dublin, (1997).

[7] Menasce, D.A., Almeida, V.A.F., Dowdy, L.W., *Capacity Planning and Performance Modeling*, Prentice Hall, (1994).

[8] PostModern Computing Technologies Inc., *ORBeline Reference Guide*, Mountain View, CA 94043, (1994).

[9] Ramesh, S., Perros, H.G., "A Multi-Layer Client-Server Queueing Network Model with Synchronous and Asynchronous Messages", *Proc.of First Int. Workshop on Software and Performance*, Santa Fe, 107-119, (Oct. 1998).

[10] Rolia, J.A., Sevcik, K.C., "The Method of Layers", *IEEE Trans. on Software Engineering*, 21(8), 689-700, (Aug. 1995).

[11] Smith, C.U. *Performance Engineering of Software Systems*, (1990).

[12] Woodside, C.M., Neilson, J.E., Petriu, D.C., Majumdar, S., "The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software", *IEEE Trans. on Computers*, 44(1), 20-34, Jan. 1995.