Application Level Performance Optimizations for CORBA-Based Systems

Weili Tao,
Systems and Computer Eng. Dept.,
Carleton University,
Ottawa, CANADA.

Shikharesh Majumdar
Systems and Computer Eng. Dept.
Carleton University,
Ottawa, CANADA
+1 (613) 520-5654 code
majumdar@sce.carleton.ca

ABSTRACT
Middleware provides inter-operability in a heterogeneous distributed object computing environment. Common Object Request Broker (CORBA) is a standard for middleware proposed by OMG. Although inter-operability is achieved middleware often introduces overheads that impair system performance. This research is concerned with performance enhancement of CORBA-based systems by deploying appropriate techniques at the application level. The paper demonstrates that decisions made by the application software designer and programmer can have a large impact on the performance of a CORBA-based system. The paper presents a set of guidelines that can be used at the design and implementation levels for enhancing system performance. We focus on issues such as reduction of connection set up latency, appropriate techniques for parameter passing, impact of method placement on response time, performance implications of different ways of packing objects in servers and load balancing. Insights into system behavior that highlight the effectiveness of the guidelines as well as capture the relationship between the CORBA compliant middleware and overall application performance are presented.

Keywords: CORBA performance, middleware performance, performance optimization, design guidelines.

1. INTRODUCTION
This research focuses on *Distributed Object Computing (DOC)* which is currently one of the most popular paradigms for application implementation [19]. It combines the desirable properties of distributed processing such as concurrency and reliability with the well-known re-usability characteristics of object oriented technology. The ability to run DOC applications over a set of diverse platforms is crucial for achieving scalability as well as gracefully handling the evolution in hardware and platform design. Additional components are added to an existing system for handling an increase in workload or the incorporation of new features in an embedded application for example. Due to the continuous improvement in computing technology, the newly added components are often built using a technology that is different from that used for implementing the legacy components. Moreover, the new feature in the embedded system may require a special platform for execution. An effective middleware system is required to provide the glue that holds such a heterogeneous distributed system together, and achieve high system performance. Using such middleware software it is possible for a client and server written in different languages and implemented on top of different operating systems to communicate with one another [24]. The paper is concerned with the performance of systems using middleware that provide this communication infrastructure and inter-operability in a heterogeneous distributed environment. *Common Object Request Broker (CORBA)* is an OMG standard for middleware used widely by distributed system builders [14]. This research is concerned with such CORBA-based middleware systems. Although Commercial-Off-The-Shelf middleware products provide inter-operability they often introduce performance overheads that adversely affect the latency and throughput which make them unsuitable for performance demanding applications. Examples include telephone switches and process control systems, the proper functionality of which depends on the ability to respond to a given volume of incoming requests in a timely manner. When conventional implementation of CORBA is applied to such systems, middleware overheads often degrade performance to such an extent that system specifications are violated [16].

Research at the Real Time and Distributed Systems Lab of Carleton University is addressing this problem regarding middleware performance at three different levels. At the application-middleware interaction level, we have investigated innovative architectures for client-middleware-server interactions and their impact on performance [1, 18]. Performance enhanced middleware systems that exploit limited heterogeneity in systems and dynamically bypass overhead incurring standard CORBA operations through a "flyover" when a similar client-server pair communicates is discussed in [2, 25]. This research focuses on performance optimization at the application level. Engineering performance into software and system require the ability to specify and analyze application performance [22] as well as the

Permission to make digital or hard copies of part or all of this work or personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.
WOSP '02, July 24-26, 2002 Rome, Italy
© 2002 ACM ISBN 1-1-58113-563-7 02/07 …$5.00

availability of guidelines and techniques for the design and implementation of performance enhanced software. This paper focuses on performance optimization techniques at the application level for CORBA-based systems. The contribution of the paper includes a set of guidelines that can be used during application design and implementation. The guidelines are concerned with four different issues: connection setup latency, placement of methods into servers, packing of objects into servers, and load balancing. Based on a synthetic workload, a network of workstations, and the Orbix MT-2.3 middleware we have investigated the impact of these performance optimization techniques. The performance improvement is observed to vary with the technique: from a few percent to an order of magnitude. Insights gained into system behavior and performance are reported. Some of the observations are intuitive, but our experimental results help the designer to quantify the performance impact and therefore determine the importance of the performance optimization in the context of a given application. The other observations are counter-intuitive and are therefore important for system design and implementation. The results of this research are likely to be most useful in the context of general CORBA-based systems. Although most of the currently available COTS middleware products are based on this general CORBA specifications, real time CORBA has started receiving attention from researchers. Issues specific to real time CORBA are beyond the scope of this paper.

The layout of the paper is presented. The next section provides a short description of CORBA and related work on CORBA performance. A brief introduction into the performance optimization issues and the environment used in our investigations are presented in Section 3. Section 4 describes the results revealing the impacts of the proposed performance optimization techniques. Our conclusions captured in terms of a few design and implementation guidelines are presented in Section 5.

2. CORBA AND RELATED WORK

CORBA has been used in the context of various applications that include enterprise computing [20] and network management [6]. A number of papers have captured the performance limitations of several existing CORBA-based middleware products. Low level C++ wrappers for sockets were observed to outperform two implementations of CORBA: Orbix and ORBeline. The latency and scalability problems in conventional middleware products are discussed further in [5]. The authors observe high latency for these CORBA implementations that also give rise to poor system scalability. A performance comparison of a number of products with a multithreaded architecture, CORBAplus, HP Orb Plus, miniCOOL, MT-Orbix, TAO, and Visibroker is available in [16]. Using load balancing techniques at the middleware can lead to an improvement in system performance [15]. The integration of load balancing with name service is proposed in [4]. The performance of a group membership protocol based on CORBA is discussed in [13]. Achieving high performance by using innovative client-middleware-server interaction architectures is described in [1] whereas performance optimized ORBs that exploit limited heterogeneity in systems are described in [2, 25]. The limitations of the original CORBA standard in providing real time QoS, and in handling partial failures are described in [11]. Real time ORBs that need to provide low latency as well as predictability have started receiving attention from researchers (see [9, 8] for example). A more elaborate literature survey on CORBA performance is available in [12].

Most of the previous works have focused on construction of low overhead fast ORBs for achieving high performance. This paper focuses on enhancement of system performance through appropriate performance optimization techniques applied during application design and implementation.

3. APPLICATION LEVEL PERFORMANCE OPTIMIZATIONS

We have considered five different issues that concern application performance. These include the connection setup latency experienced by the client when it binds to a server, the relationship between different ways of parameter passing and performance, the impact of placement of methods in an object on the method dispatching latency, the impact of the object packing technique used on performance and load balancing. The performance optimization techniques useful in these contexts are to be considered during application design and implementation and are therefore important in the context of this paper. A brief description of the experimental environment and performance metrics used in the investigation of these techniques are presented next. The results of the experiment that demonstrate the performance impact of these techniques are presented in Section 4.

3.1. Experimental Environment

The experiments used in demonstrating the impact of the performance optimization techniques are carried out on a network of sun workstations running under Solaris 2.6. The workstations are connected by a 100 Mbps Ethernet-based LAN. The CORBA compliant middleware used is Orbix MT 2.3 [7]. A closed system running a synthetic workload is used. With such a workload it is possible to vary the different characteristics such as service times and message lengths that is crucial for answering "what if" questions.

Each client ran in a cyclic manner and invoked a single method in each cycle. In most experiments a client cycle was immediately followed by another. The only exception is the set of experiments described in Section 4.5 in which two successive cycles in a class of clients are separated by a think time. A server either returns immediately after being called (zero service time) or executed a for loop to consume a pre-determined amount of CPU time. Both fixed and exponential distributions have been used for service times. Unless mentioned otherwise the server execution times are assumed to be fixed at the mean in the following discussions. The parameters passed during a method invocation is controlled both in terms of the mix of data types used as well as the number of data units in the parameter list transferred between the client and the server. The number of workstations used varies from experiment to experiment. A single workstation is always used however for running the clients. Since clients consume very little CPU time and mostly wait for a server response or the completion of a thinking period, concurrency in client execution is not impaired by running them as separate threads on the same workstation. The following performance metrics are used in the evaluation of the various performance optimization techniques presented.

Connection Set up Latency (T): is the time duration between the invocation of a bind operation made by the client and the time at which the operation is completed. T is measured in milliseconds (ms)

Response Time (R): is the time duration between a method invocation made by a client and the arrival of its result at the client site. R is measured in terms of milliseconds.

System Throughput (X): is the sum of the throughputs of each client that is measured in number of client cycles completed per second.

Note that R and X are related by Little Law [10]: C = XR where C is the number of clients

Ratio: is a metric used in certain cases for capturing the relative performances of different options. It is the ratio between the response time achieved with a specific case and the response time of the base case identified in the text.

Response times and latencies are measured by inserting Solaris system calls for time measurement at appropriate positions in the client code. The experiments were run long enough to achieve an interval that is lower than ± 5 % of the mean at a confidence level of 95%.

4. PERFORMANCE OPTIMIZATION GUIDELINES

This paper is concerned with performance optimization techniques that can be deployed during system design and implementation. A number of issues that are important in this context were described at the beginning of Section 3. An investigation of each of these issues including the performance data resulting from the experiments conducted on the network of workstations is presented in a separate subsection.

4.1 Connection Setup Latency

A client needs to bind to a server object before it can invoke the desired method. A connection is set up between a client and its server whenever a bind operation is performed. In response to a bind call, the middleware activates the server if the server was found inactive. It also identifies the inter-object reference (IOR) or handle associated with the object and returns it to the client. The client can then use this handle for invoking the desired method. The connection setup latency is likely to have a strong impact on performance if the client binds to the different objects in the systems frequently. Binding is mandatory when the client uses an object for the first time. In spite of the availability of a cached object handle, the client may choose to bind to the object for avoiding a runtime exception in the event that the object instance has crashed after the previous bind operation. Wit replicated objects, the binding call always returns the handle of an object instance that is functional at the current time. The different factors that are expected to have a significant impact on the connection setup latency are presented.

- *State of the server*: Active (A) or Non Active (NA)
- *Distribution of client and server*: Same Node (SN) or Different Nodes (DN)
- *Collocation of client and server in the same address space*: Collocated (C) or Not Collocated (NC). This option is available to the designer only when the client and its server are allocated on the same node (SN). A feature for collocation available in Orbix is used to achieve this collocated configuration.

Various combinations of these factors lead to a number different cases that are listed in Table 1.

Table 1. Configurations for Connectio Setup Latency

Factors	Case I	Case II	Case III	Case IV	Case V
Server state	NA	NA	A	A	A
Distribution	SN	DN	SN	DN	SN
Collocation	NC	NC	NC	NC	C

The connection set up latency T measured with each of these configurations is presented in Figure 1.

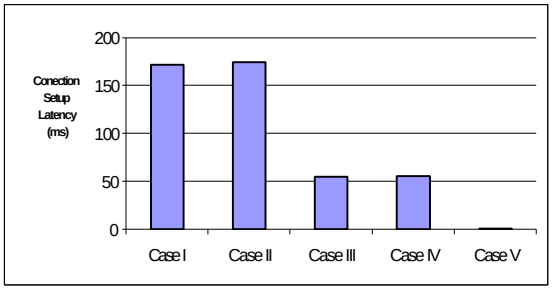

Figure 1. Connection Setup Latency Achieved with Different System Configurations

Activating a server is expected to incur an extra overhead. Our experimental results indicate that this overhead can be substantial on a real system. The latencies achieved with Case III and Case IV that correspond to a system with an active sever are approximately 32% of those achieved with Case I and Case II that incurred the extra overhead of server activation. Every active sever consumes a finite amount of system resources. Thus even though it may not be possible to keep every server on the system active for all the time, keeping the "popular" servers permanently active can lead to a substantial savings in connection setup latency. A further savings is achieved by locating the client and server on the same address space: the latency achieved with Case V is an order of magnitude lower than those achieved with Case III and Case IV. Note that in most systems client and servers are compiled into separate processes. Our experiments demonstrate that the non-intuitive design decision of client-server collocation can significantly improve system performance.

4.2 Impact of Parameter Passing

System throughputs achieved with various types of native data types passed as arguments are described in [3]. In this paper we investigate applications that pass more complex arguments involving structures that are often passed during a method invocation. Our experiments demonstrate that the way parameters involving such complex data structure are passed between the client and its server can have a significant impact on performance. Figure 2 displays three different ways of parameter passing involving an array of structures. Option I (see Figure 2a) is the most natural in which the array of structures is directly passed. In Option II (see Figure 2b) the array of structures is unraveled into two component arrays of primitive data types whereas in Option III two methods each with a separate parameter are invoked. We assume that the methods in three different cases are implemented in such a way that the same functional result is achieved with each option. The relative performances of the three options are presented in Figure 3 that displays the response time for a zero server execution time observed with each option normalized with respect to that achieved with Option I.

Option I: Method 1 with one parameter

(a) struct CD [size] {

char c; double d}

Option II: Method1 with two parameters

(b) char [size] c; double [size] d;

Option III: Method1 with parameter char [size] c;

(c) Method 2 with parameter double [size] d;

Figure 2. Different Ways of Parameter Passing (a) Option I (b) Option II (c) Option III.

Option II demonstrates the best performance. Its relative performance with respect to the most natural Option I increases with the array size. A performance improvement of over 100% is observed, for example, for arrays with 1000 or more elements. It is interesting to note that although at lower array sizes Option III is inferior in performance to Option I, Option III assumes the position of the second best performer at higher array sizes.

The rationale behind system behavior is explained with the help of Table 2 that shows the data alignment specified in CORBA's CDR format that is used by the ORB for transferring data over the wire. Since data types such as long and double have to start at specific byte boundaries padding bytes may be used between two different types of data elements. For example, if a double is to follow a char and the char starts at position 0, the double will start at position 8 (see Table 2) resulting in padding bytes between the two elements. Padding bytes do not have any semantic value but consume bandwidth. By unraveling the structure, Option II leads to an efficient packing of the elements of the same type together and reduces the number of padding bytes. The concomitant savings are two fold: in terms of communication time as well as in terms of CPU time consumed in marshalling and unmarshalling. For higher array sizes Option III performs better than Option I. Although Option III incurs an additional method invocation overhead, this additional overhead is offset at higher array sizes by the reduction in the number of padding bytes. Combining two methods into one makes intuitive sense during system re-design and modification aimed at performance improvement. System modification for performance enhancement is common in the software industry and was observed by the author in the context of telephone switches built by Nortel. The relative performances of Option I and Option III demonstrate that an unexpected result may accrue if this re-designing operation is not performed carefully. This is discussed further in Section 5.

Table 2 . CDR Alignment for Primitive Fixed Length Types.

Starting Byte Boundary	Data types
Multiples of 1	Char, octet, boolean
Multiples of 2	Short, unsigned short
Multiples of 4	Long, unsigned long, float, enumerated types
Multiples of 8	Long, long, unsigned long long, double, long double

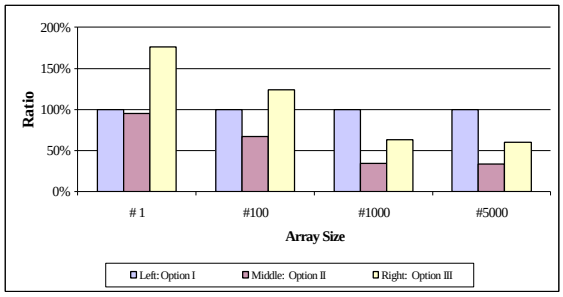

Figure 3. The Effect of Parameter Passing on Response Time (Zero Server Execution Time).

The experiments were repeated with nested structures as argument for the called method. The results confirm that the best performance is achieved by calling a single method and unraveling the parameters with an array for each primitive data component of the structure being sent as a separate parameter. Due to space limitations the numeric results are not included (see [23] for details).

4.3 Method Placement

An object can contain multiple methods. Whenever a method is invoked, the request is steered to the appropriate code corresponding to the method. The delay experienced in

dispatching the request is referred to as the method dispatching latency. Other studies have reported the dependence of this dispatching latency on the position of the method declaration in the interface for the object [3, 17]. An investigation of this relationship is presented in this section. A designer can be faced with the question of how many methods are to be packed in one object. For example, given a number of methods it may be possible to encapsulate all the methods in one object or to pack a single method in a separate object. Does the packing decision have a significant impact on performance? We have compared two systems Model I and Model II. In Model I there are 50 methods packed into a single object whereas in Model II there are 50 objects each containing a single method. The dispatching latency for a method with a given position in the interface definition is plotted in Figure 4 for both Model I and Model II. It is measured as the response time obtained with a zero server execution time. The dispatching latency is observed to increase linearly with the position of the method in the interface. The slope of the line is small. A similar observation has been made with a system with 500 methods (see [23] for details). Model I performs better than Model II. This suggests that a smaller method dispatching latency is achieved when all the objects are packed into a single object. Note that this observation is dependent on the method demultiplexing strategy used by the ORB product and is likely to vary from one product to another

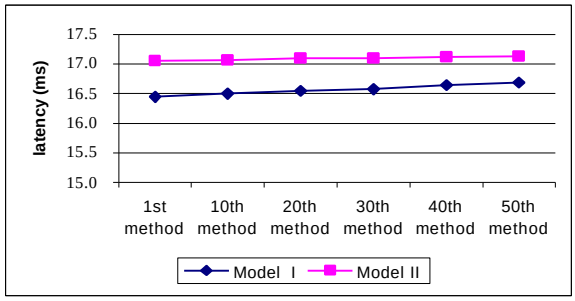

Figure 4 . Method Dispatching Latency (Zero Server Eexcution Timnme)

4.4. Packing of Objects and Distribution of Servers

Consider an application consisting of multiple client-server pairs. The impact of packing objects into servers and distribution of servers on separate nodes when multiple CPU's are available are discussed in this sub-section. How many methods should be placed in an object? Should a separate server process be used for each object? How should these servers be distributed among the computing nodes? In order to investigate these questions we have constructed four different prototypes for a system. Each client invokes a separate method and for the data presented in Figure 5, a method execution consumes 0.03 ms of CPU time. For the sake of simplicity we assume that each method is invoked by a distinct client. Each prototype corresponds to a particular mode that is described next.

- *Mode I:* All methods are placed in a single object.
- *Mode II:* Each method is placed in a separate object. All the objects are packed into a single server.
- *Mode III:* Each method is placed in a separate object. Each object is packed into a separate server. All the servers are allocated to the same node.
- *Mode IV:* Each method is placed in a separate object. Each object is packed into a separate server with each sever running on a separate node.

A system with a variable number of clients and servers is experimented with. The number of workstations used for the servers is dependent on the mode used. The maximum number of servers used in this experiment is 15 that is determined by the number of workstations that are available in our measurement network. The response time observed for a method invocation in each of these modes is presented in Figure 5. As expected Mode IV that maximizes the concurrency of server execution by allocating each server on a separate node gives rise to the best performance. The performance differences among the different options increase with the number of clients in the system. At higher number of clients Mode III performs the worst whereas Mode I and Mode II demonstrate a comparable performance.

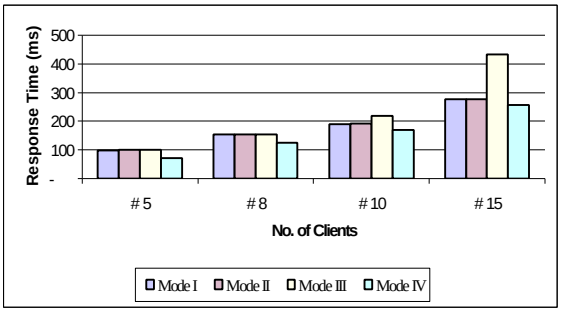

Figure 5. Effect of Server Packing and Distribution on Performance (Method Execution Time = 0.03 ms).

One of the reasons for the inferior performance of Mode III is the additional context switching overhead among multiple processes incurred in this option. The data presented in Figure 5 indicates that on a single CPU system, packing multiple objects, irrespective of their functionalities, into a single server is attractive for optimizing system performance. This is contrary to an intuitive placement of functionally unrelated objects into separate servers. The impact of method placement into objects on performance is some what secondary. A similar observation is made for systems with higher server execution times [23].

4.5. Load Balancing

Load balancing refers to the equitable distribution of workload among multiple instances of system resources. Three types of load balancing strategies are described in [15]. In *network-based* load balancing IP routers and domain name servers can steer successive requests for a resources to different IP addresses each of which corresponds to a separate instance of the desired resource. Load balancing is performed at the *operating system* level by migrating processes from a heavily loaded node to a node that is currently experiencing a lower load [21]. Load balancing can also be performed at the *middleware* level by a

CORBA name service agent that returns separate handles for different client requests to different instances of a replicated object. Providing a load balancing service in CORBA is also discussed in [4, 15]. Load balancing, however, is not a mandatory component of an ORB. In the absence of such a centralized service, load balancing may be performed at the application level for dedicated applications in which multiple clients and servers cooperate with one another to achieve a common objective. Examples include telephone switches and various embedded systems. This section describes a preliminary investigation of application level load balancing.

Five strategies are investigated to determine the effectiveness of application level load balancing. The performance evaluation is done in the context of a system characterized by two classes of clients *busy* and *non-busy*. A busy client operates cyclically; in each cycle it invokes a method and repeats the cycle as soon as the results of the method invocation is received. A non-busy client is similar to a busy client except that its two consecutive cycles are separated by a think time. The system consists of S replicated servers and the method invocation can be served by any one of the S instances. The total number of clients in the system are C and half of them are assumed to be busy and half non-busy. The think time is fixed at 500 ms.

The strategies used in this investigation of load balancing are briefly described.

- *Strategy 1 (Unbalanced System):* Half of the servers is used by the busy clients whereas the other half are used by the non-busy clients. This represents the pathological case in which the load is poorly distributed.
- *Strategy 2 (Balanced System):* Each server instance is used by one busy and one non-busy client. This is one of the best possible static load balancing strategy that can be deployed during system design. However, an exact a priori knowledge of the workload generated by the clients is required.
- *Strategy 3 (Half Dynamic Self Round Robin):* The selection of a server is dynamically performed at method invocation time by each busy client. A busy client selects a different server instance for each successive method invocation. Each non-busy client is statically bound to a fixed server instance. The non-busy clients are divided equally among the servers. The server handles are assumed to be acquired by each client during system initialization.
- *Strategy 4 (Full Dynamic Self Round Robin):* Both busy and non-buy clients select a server instance dynamically. Each client uses a local Round Robin strategy and routes successive method invocations to different servers. The server handles are assumed to be acquired by each client during system initialization.
- *Strategy 5 (Dynamic Central Round Robin):* Instead of performing routing decisions locally, the system uses a central dispatcher node for determining the server instance to be used by a client. Each client contacts the dispatcher that returns a server handle to the client. The dispatcher uses a Round Robin strategy for selecting the server to be used by a client.

Experiments are performed for two different systems. The first system is characterized by C= 4 and S =2 whereas the second is characterized by C = 8 and S = 4. The experimental results for the first system with a server execution time of 10 ms are presented in Figure 6.

The highest system throughput is achieved with Strategy 2 in which the load is statically balanced a priori. Both the self round robin strategies perform comparably and occupy the rank next to Strategy 2. The dynamic self round robin strategies perform comparably with one another and occupy the rank next to Strategy 2. It is interesting to note that the worst performance is achieved by Strategy 5. The extra overhead incurred in getting the object handle from the dispatcher is one of the reasons for the inferior performance of the Dynamic Central Round Robin. The relative performance of the strategies with respect to Strategy 1 are captured in Figure 7 for various server demands. The rankings among the policies remain the same for different server execution times. With the exception of Strategy 5, the throughput for a given strategy such as Strategy 2 decreases monotonically with server execution times: the highest throughput is achieved for a server execution time of 10 ms and the lowest for a server execution time of 120 ms. However, the performance differences among the strategies seem to diminish at high server demands. Both the servers in systems characterized by high server demands are busy most of the time and comparable performance is achieved by all the load balancing strategies. Performance of a larger system with C = 8 and S = 4 is presented in Figure 8. Strategy 2 demonstrates the highest performance improvement. Strategy 5 takes the second position in terms of performance. The advantage of a centralized load balancing strategy seems to be observable on larger systems in which the handle fetching overhead is offset by the performance improvement produced by load balancing..

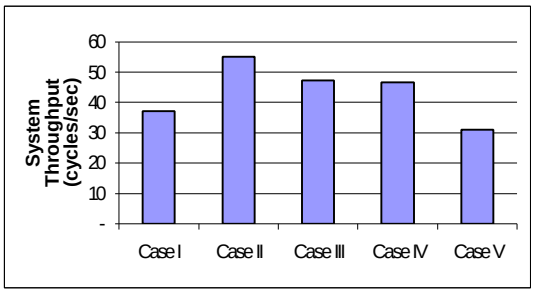

Figure 6. Performance of Load Balancing Strategies (C=4, S=2, Server Execution Time =10 ms, think time = 500 ms).

With the exception of Strategy 5, the throughput for a given strategy such as Strategy 2 decreases monotonically with server execution times: the highest throughput is achieved for a server execution time of 10 ms and the lowest for a server execution. The data presented in Figure 7 and Figure 8 correspond to fixed service times. A similar observation regarding the relative performances of the strategies are made when server execution times and think times are exponentially distributed (see [23] for details).

5. Conclusions

Heterogeneity is natural in distributed object computing systems. CORBA provides inter-operability in such heterogeneous environments. Additional overheads incurred in the CORBA compliant ORB deteriorate system performance. Achieving high performance in CORBA-based systems by using innovative client-middleware-agent architecture and by exploiting limited heterogeneity in systems are discussed in the literature. This paper focuses on performance optimizations at the application level. Based on an empirical approach we have investigated the impact of a number of design and implementation decisions on performance. The insights gained from the results of our experiments conducted on a network of sun workstations running Solaris 2.6 and using the Orbix MT-2.3 middleware have resulted in a set of guidelines for the system designer and implementers. These guidelines are briefly summarized.

- *Connection Setup Latency:* The state of the server has a strong impact on connection setup latency. In resource constrained systems in which it is not possible to keep all the servers active maintaining an active state for at least the popular servers that are used often can lead to a large reduction in the connection setup delay.
  Different modules such as clients and servers are typically compiled into separate processes. When the client and server are allocated on the same node, collocating the client-server pair in the same address space is recommended: an order of magnitude improvement in performance is achieved when the client-server collocation is used. Note that an Orbix binding function was used in name resolution. The importance of the factors such as server activation and client-server collocation identified in this study as well as whether or not additional factors need to be considered in the context of a system using CORBA naming service for name resolution warrant investigation.

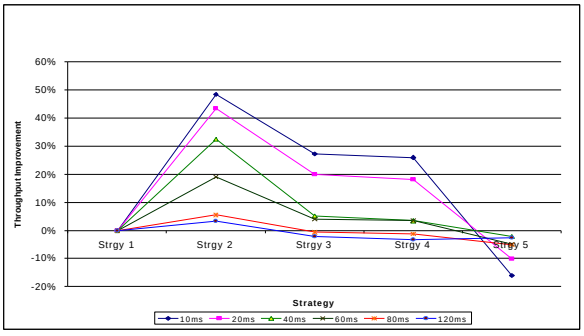

Figure 7. Performance Improvement for Different Server Demands (C=4, S=2, think time = 500ms)

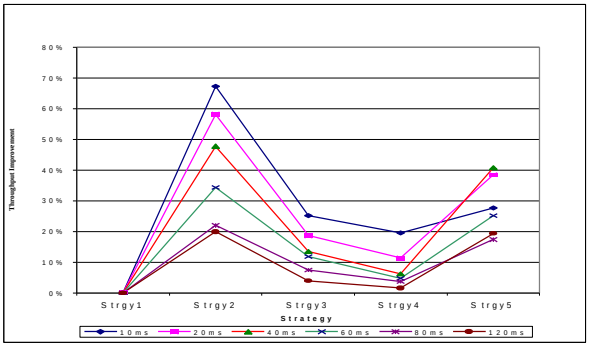

Figure 8. Performance Improvement for Different Server Demands (C=8, S=4, think time = 500 ms).

- *Parameter Passing:* Complex data structures are often passed between a client and its server. Significant performance differences are observed among different ways of parameter passing. The performance differences are observed to increase with the size of the parameter list. Unraveling complex structures into the component primitive types is observed to give rise to the best performance. More than 100% improvement in performance is observed with the system investigated in Figure 3. Although the technique is observed to produce a performance benefit for all parameter sizes a larger performance benefit is expected to accrue for communication bound systems characterized by larger messages exchanged between the client and the server.
- *Combination of Methods and Performance Recovery:* A secondary, nevertheless important observation is made in the context of parameter passing discussed in Section 4.2. Additional passes through existing code are some times made for reducing system overheads for improving performance. Combining multiple methods into one is often performed during such performance recovery operations for reducing the method invocation overheads. The empirical data in Figure 3 demonstrates a non-intuitive result: the system with two method invocations performs better than the system with a single method invocation passing an array of structures when the size of the array is large. The occurrence of such a non-intuitive result indicates that a careful consideration of the application is required before performing such performance operations.
- *Method Placement:* Compared to the other factors method placement is observed to have the least impact on performance. The method dispatching latency is reduced by packing a larger number of methods into an object. The position of the method in the interface has a small impact on performance: methods called more frequently should thus be defined as close to the beginning of the interface declaration as possible. Note that this observation is dependent on the search method used by a particular ORB

product and may/may not be significant in the context of different products.

- *Packing of Objects:* When all the objects are allocated on the same CPU the designer should avoid placing the objects on separate servers. Collocating the objects into the same server process or packing all the methods into one object demonstrated a higher performance in comparison to the system in which a separate process is created for each object.
- *Load Balancing:* Static load balancing based on a priori knowledge of client-server workload characteristics demonstrated the best performance for the workloads used in our experiments. Although the availability of such a priori knowledge is possible for dedicated applications it is unlikely to be available during earlier stages of system design. Moreover, in many applications a client may not remain in one "busy" or "non-busy" class and may change its behavior depending on system state. The dynamic strategies are useful in such a situation. The local strategies in which each client selects a server instance independently using a round robin strategy is preferable for smaller systems. For more complex systems using a centralized dispatcher-based global strategy can significantly improve system throughput.

Although most of the principles underlying system performance discussed in this paper are general and are expected to hold in the context of different middleware products, the degree of the performance impacts of the factors may vary with the middleware product used. Extending this study on systems using different middleware products is warranted. We have used fixed and exponential distributions for the generation of service times. Using distributions that give rise to more variable service times forms an important direction for future research. Only two classes were used in the preliminary investigation of load balancing presented in this paper. Validation of the conclusions for systems with more than two classes of clients under different workload types is required. The results presented in this paper are based on synthetic workload. Applying the guidelines to real applications is worthy of investigation.

ACKNOWLEDGMENTS

This work was supported by CITO and Nortel networks. Much of the motivations for this work came from Brian Carroll and Mark Christopher of Nortel Networks in Ottawa.

References

[1] I. Abdul-Fatah and S. Majumdar, "Performance Comparison of Architectures for Client-Server Interactions in CORBA", *IEEE Trans. On Parallel and Distributed Systems*, Vol. 13, No.2, February 2002, pp. 111-127.

[2] I. Ahmad and S. Majumdar, "Achieving High Performance in CORBA-Based Systems with Limited Heterogeneity", *Proc. IEEE International Symposium on Real Time Object Oriented Computing, M*agdeburg, Germany, April 2001, pp. 350-359.

[3] V. Amar, "Benchmark Results", www.beust.com/virginia/benchmark, July 2000.

[4] T. Barth, G. Flender, B. Freisleben and F. Thilo, "Load Distribution in a CORBA Environment", *Proc. International Symposium on Distributed Objects and Applications (DOA'99),* Edinburgh, UK, September 1999.

[5] A.S. Gokhale and D.C. Schmidt "Measuring and Optimizing CORBA Latency and Scalability Over High-speed Networks", *IEEE Transaction on Computers*, Volume 47, No. 4, April, 1998.

[6] P. Haggerty, K. Seetharaman, "The Benefits of CORBA-Based Network Management", *Communications of the ACM*, Vol. 41, No. 10, October 1998, pp. 73-80.

[7] Iona Technologies, *Orbix Programmers' Guide*, Dublin, Ireland, 1997.

[8] F. Kon, M. Roman, P. Liu, J. Mao, T. Yamane, L.C. Magalhaes, and R.H. Campbell "Monitoring, Security, and Dynamic Configuration with the dynamic TAO Reflective ORB", *Proc. IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware 2000),* Pallisades, New York April 3-7, 2000. pp.121-143.

[9] B. Li and K. Nahrstedt, "QualProbes: Middleware QoS Profiling Services for Configuring Adaptive Applications", *Proc. IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware 2000),* Pallisades, New York April 2000, pp.256-272.

[10] J.D.C. Little, "A Proof of the Queueing Fdormula L = lambda W", *Operations Research*, Vol. 9, 1961, pp. 383-387.

[11] S. Mafeis, D.C. Schmidt, "Constructing Reliable Distributed Comunication Systems with CORBA", *IEEE Communications Magazine*, Vol. 35, No. 2, February 1997, pp. 72-78.

[12] S. Majumdar, W.-K. Wu, "CORBA Middleware: a Performance Perspective" *Contract Report for Department of National Defense*, Department of Systems and Computer Engineering, Carleton University, October 2000.

[13] S. Mishra and X. Liu, "Design, Implementation and Performance Evaluation of a High Performance CORBA Group Membership Protocol", Proc. *7th International Conference on High Performance Computing*, Bangalore, India, December 17-20, 2000.

[14] Object Management Group, "The Common Object Request Broker: Architecture and Specification", 2.3 ed., June 2000.

[15] D.C. Schmidt, O. Othman, C. O'Ryan, "Strategies for CORBA middleware-based Load Balancing", Distributed Systems Online, Vol.2, No.3, March 2001, http://dsonline.computer.org/0103/features/oth0103_print.htm.

[16] D.C. Schmidt , "Evaluating Architectures for Multi-threaded CORBA Object Request Brokers", *Communication ACM*, vol 41 No. 10, October 1998, pp. 54-61.

[17]] D.C. Schmidt and A. Gokhale, "Evaluating the Performance of Demultiplexing Strategies for Real-time CORBA", *Proc. GLOBECOM '97 conference*, Phoenix, AZ, November, 1997.

[18] E-K. Shen, S. Majumdar and I. Abdul-Fatah, "High Performance Adaptive Middleware for CORBA-Based Systems", *Proc. ACM Principles of Distributed Computing (PODC) Conference*, Portland, July 2000, pp. 119-207.

[19] E. Shokri, P. Sheu, "Real-Time Distributed Computing: An Emerging Field", *IEEE Computer*, June 2000, pp. 45-46.

[20] J. Siegel, "A Preview of CORBA 3", *IEEE Computer*, May 1999, pp. 114-116.

[21] M. Singhal, N. Shivaratri, *Advanced Concepts in Operating Systems*, Mc-Graw Hill, 1994.

[22] C.U. Smith, *Performance Engineering of Software Systems*, Addison-Wesley, 1990.

[23] W. Tao Application Level Guidelines for Building High Performance CORBA-Based Systems, M.C.S. Thesis, Dept. of Computer Science, Carleton University, Ottawa, Canada, May 2002.

[24] S. Vinoski, "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments", *IEEE Communications Magazine*, Vol. 35, No. 2, February 1997, pp. 46-55.

[25] W.-K. Wu, S. Majumdar, "Engineering CORBA-Based Systems for High Performance", Proc. 2002 International Conference On Parallel Processing (ICPP), Vancouver, August 02 [to appear].