Evaluating Overhead and Predictability of a Real-Time CORBA System.

Conference Paper · January 2004

DOI: 10.1109/HICSS.2004.1265651 · Source: DBLP

David C Levy
University of Sydney
128 PUBLICATIONS **621** CITATIONS

Evaluating Overhead and Predictability of a Real-time CORBA System

Jianfan Zou, David Levy
School of Electrical and Information Engineering
University of Sydney, Australia
{jeffz, dlevy}@ee.usyd.edu.au

Anna Liu
Microsoft Pty Ltd,
Australia
Annali@microsoft.com

Abstract

Over the last few years, real-time CORBA are increasingly used in development of distributed real-time applications, in which real-time CORBA and operating system constitute the computing and communication infrastructure upon which the applications execute. However the amount of effort put into studying and evaluating the real-time characteristics of real-time CORBA is still inadequate. In this paper we focus on examining the overhead and predictability of real-time CORBA, which are two of the most important concerns when building real-time applications. Firstly we introduce the desired features in real-time applications and possible sources of unpredictability. Secondly, we evaluate a real-time CORBA implementation : TAO v1.2 on Linux kernel v2.4.18 over 100M Ethernet. It is observed that: 1) the overhead of TAO is acceptable, compared with the equivalent application using TCP/IP socket directly; 2) TAO presents rather predictable behaviors under light network traffic, but unpredictability of CORBA invocations increases significantly as the network becomes overloaded. Moreover, we identify that the major source of unpredictability of CORBA invocations is in the TCP/IP stack of Linux rather than in TAO itself.

1. Introduction

Over the last few years, a number of real-time CORBA (Common Object Request Broker Architecture) products have appeared in the real-time middleware market and they are increasingly used in large-scale distributed real-time applications. These products include TAO from Washington University, St Louis [1], ORBexpress from Objective Interface Systems [2], e*ORB from PrismTech [3], Visibroker-RT from Borland [4], etc. In the earlier real-time CORBA products, various incompatible proprietary technologies were used to support real-time applications. Since OMG (Object Management Group) standardized Real-Time CORBA Specification and added it into CORBA specification 2.4, an increasing number of vendors have started to provide standard-compliant real-time CORBA products.

Real Time CORBA specification is an optional extension to standard CORBA [5] in the support of fixed priority distributed real-time applications which require high end-to-end predictability. It defined a set of additional interfaces and mechanisms to allow application developers to explicitly control and configure computing and networking resources in ORB (Object Request Broker) end-system. The following new features are supported:

- **Priority propagation:** The CORBA server can processes the requests from clients either at their client side thread priority, or at the priority pre-determined on the server side.
- **POA thread pool:** POA (Portable Object Adaptor) can be associated with a configurable thread pool.
- **Explicit connection management:** Clients can explicitly bind to the server with an pre-established connection. This connection can be a priority-banded and/or non-multiplexed private connection.

Real-time application developers need to ensure that a particular real-time CORBA product running on a specific operating system can provide real-time characteristics required in their projects. However, existing middleware evaluation projects often give little attention to real-time aspects. For real-time systems, overhead and predictability are two of the most important concerns. In this paper we will investigate and analyze the possible sources of unpredictability in real-time systems, and evaluate overhead and predictability of a real-time CORBA product: TAO with a series of experiments.

The remainder of the paper is organized as follows: Section 2 introduces the features of real-time systems and analyzes the possible sources of unpredictability in real-time computer systems. Section 3 experimentally evaluates overhead and predictability of TAO real-time CORBA and presents experiment results. TAO v1.2 on Linux kernel 2.4.18 over TCP/IP and Ethernet network is examined. Section 4 introduces related works and section 5 gives concluding remarks.

2. Predictability of distributed real-time systems

As defined by Donald Gillies in [6], "A real-time system is one in which the correctness of the computations not only depends on the logical correctness of the computation but also on the time at which the result is produced. If the timing constraints of the system are not met, system failure is said to have occurred." Two complementary features are essential for real-time systems.

- **Timeliness** requires the work to be completed within pre-defined time constraints.
- **Predictability** requires the behavior of the system to be highly predictable and deterministic.

Predictability is the most desired features in real-time systems and it is impossible to guarantee timeliness without predictability. General speaking, predictability means that it should be possible to demonstrate or prove that requirements can be met under given assumptions ahead of time. For real-time systems the predictability with respect to the timing requirements is the first concern and we will concentrate on this aspect. In real-time area, determinism is often considered as an equivalent concept to predictability and the two terms will be used interchangeably in this paper. We enumerates the possible sources of unpredictability in the real-time systems as follows:

- **Operating systems:** thread scheduling and synchronization, concurrency control protocol for accessing shared resources, kernel preemptability, timer resolution, network protocol stack, etc.
- **Middleware layer:** connection management, message buffering, thread synchronization, thread and prioritization model, etc.
- **Network infrastructure:** QoS (quality of service) support. For instance, the traditional TCP/IP network does not support QoS whereas the ATM network does.
- **Application software:** The real-time applications often contain multiple concurrent tasks, which are related to others and form various dependencies, such as resource-sharing, precedence dependency. These dependencies may make it difficult and even impossible to analyze and validate schedulability of real-time systems.
- **Hardware:** Most of computers are designed to maximize average-case performance rather than bounded worst-case performance. A number of technologies, such as DMA (direct memory access), memory cache, instruction pipelining, may cause unpredictability in some cases.

In real-time systems containing multiple tasks with different priorities, it is expected that the high priority task has precedence in accessing computing resources over the low priority tasks. However, for various reason a high priority task may not be able to preempt the low priority tasks instantaneously at any time. The scenario in which the high-priority task ready to run waits while the low priority tasks are running is referred to as priority inversion. Although completely eliminating priority inversion is difficult and even impossible, minimizing its occurrence and priority inversion time is highly required in real-time systems. As priority inversion is one of major sources of unpredictability in real-time applications, the unpredictability caused by priority inversion in real-time CORBA ORB will be examined in the next section.

Provided all the schedulers of shared computing resources were priority-aware, priority inversion would be minimized. However, system predictability often has to trade off against system complexity. Normally only the resources assumed to be scarce are scheduled according to priorities. In most of real-time operating systems, CPU is the only resource scheduled according to priorities. Other resources such as network and hard disk usually serve requests in the best effort way and their schedulers are not unaware of priorities of the requests. Provided that these shared resources are competed for heavily, significant priority inversion and unpredictability will occur. Conversely, if competition for these resources is not intensive, priority inversion may be unnoticed in most cases.

Based on the analysis above, we can qualitatively identify whether some real-time system designs are appropriate or not. A good sample design is the use of 10Base5 10M thick coax Ethernet for communication between SIEMENS S5/S7 PLCs (Programmable Controller). As data volume exchanged between PLCs is small and the network traffic is low, the priority inversion and non-determinism caused by CSMA/CD protocol of Ethernet are bounded. An example of a bad design would be: A remotely controlled mobile robot equipped with cameras mixes the video frame streams and time-critical control messages over a shared communication link which is not priority-aware. The system may fails due to loss or lateness of the time-critical control messages when the communication link and network protocol stack are overloaded.

3. Evaluating predictability of real-time ORB

The section investigates the overhead of real-time CORBA invocations against TCP/IP socket programs and end-to-end predictability of real-time CORBA.

3.1 Overview of testbed configuration

Configurations of ORB end-system testbed are illustrated in Table 1 and Figure 1.

Table 1

Server machine	Dell Pentium IV 1.6G Hz
Client machine	Dell Pentium III 500 Hz
Network	100M Ethernet, point-to-point
Real-time CORBA	TAO 1.2
Operating system	Red Hat Linux 7.3, Kernel 2.4.18

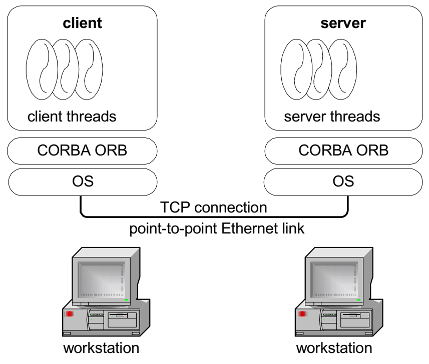

Figure 1

Unless stated otherwise the client and server are located on two different single-CPU PCs, as shown in Figure 1. TAO v1.2 on Red Hat Linux 7.3 with kernel 2.4.18-10 are evaluated. Both client and server run at real-time priority and enable POA policies "priority propagated" and "band bounded connection" to maximize end-to-end predictability.

Linux is an Unix-like multi-tasking operating system, which supports both time-sharing dynamic priority scheduling and real-time fixed-priority scheduling. The real-time priorities range from 1 to 99. The Linux kernel v.2.4 is not fully preemptive and does not support real-time resource access protocols, such as priority inheritance and priority ceiling. Linux is not suited for mission-critical hard real-time applications. However, there exists a number of real-time variants of Linux that can provide fully preemptive kernel and real-time resource access protocols, such as TimeSys Linux RT [7] and RTLinux [8]. The evaluation of these other real-time operating systems is of some interest in our future work.

TAO v1.2 is an open-source implementation of Real Time CORBA. Its ORB was design with the guidance of design patterns, such as Leader-Follower pattern and Thread Specific Resource pattern [9], and features the "Reactor per Lane" model. According to TAO's documents in [1], "Each Thread-pool Lane has its own reactor, acceptor, connector, and connection cache. Both I/O and application-level processing for a request happen in the same thread: there are no context switches, and the ORB does not create any internal threads."

The IDL definitions used in our evaluation experiments is given as follows.

```
module test {
   typedef sequence<octet> OctetSeq;
   struct message {
      long work;
       OctetSeq payload;
   };
   void method( inout message msg);
}
```

The parameter *work* specifies the amount of CPU intensive work the server performs. The parameter *payload* specifies the size of data transferred between client and server. The server receives the request and echoes back the data in payload to the client.

There are two types of CORBA invocation threads in the following experiments. Periodic invocation threads make CORBA invocations at a specific frequency and their deadline is equal to the corresponding period. For example, an H Hertz periodic invocation thread tries to make H invocations per second. The periodic thread executes task in the beginning of a period and sleeps for a specific time interval until the start of next period. A continuous invocation thread makes remote CORBA invocations immediately after the previous invocation completes.

3.2 Experiment results and analysis

Experiment 1: Overhead of CORBA invocation

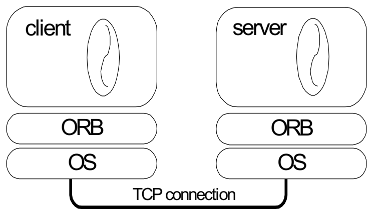

Figure 2

This experiment measures the throughput of one-way CORBA invocations and the latency of two-way CORBA invocations, then compares the results with those from an equivalent TCP/IP socket program. The experiment configuration is shown in Figure 2. Figure 3 shows the throughput of CORBA one-way invocations while Figure 4 shows the latency of CORBA two-way invocations. Figure 5 gives the overhead of CORBA two-way invocations in percentage, when compared to TCP/IP socket program.

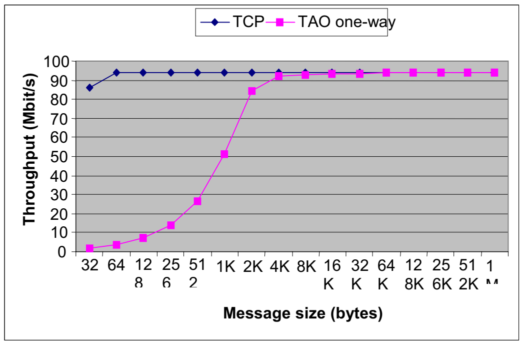

Figure 3

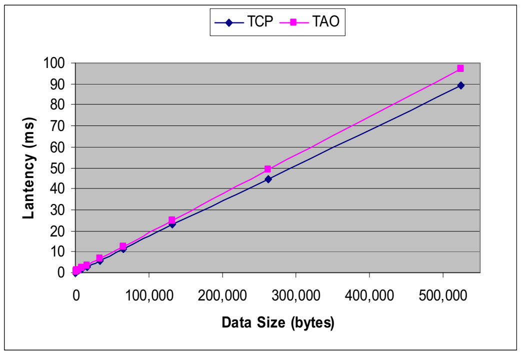

Figure 4

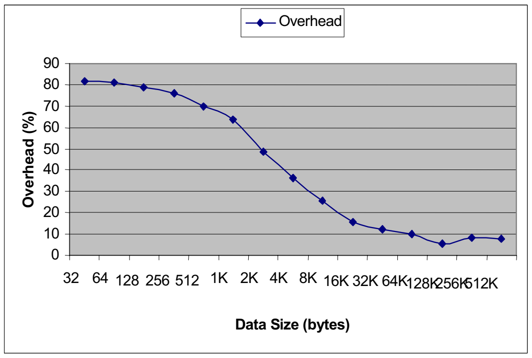

Figure 5

In Figure 3, Figure 4, Figure 5, it is observed that the overhead of CORBA invocations in percentage decreases as the data payload in CORBA invocation message increases. Figure 3 shows that, when the data payload is greater than 8K bytes, CORBA one-way throughput approaches throughput of TCP/IP socket. Figure 5 indicates that the overhead of two-way invocation drops to around 10 percent as data payload increases beyond 32K bytes, when compared to an equivalent application using TCP/IP socket directly. It is supposed that the overhead of CORBA invocations drops to a low level when data payload becomes the dominant part in CORBA message. All these results indicate that the overhead of TAO is acceptable for most of applications when using CORBA one-way and two-way invocation.

Experiment 2: Resolution of Timer

The distributed real-time systems based on CORBA often contain some periodic tasks which invoke a CORBA remote method and then sleep for a time interval until the next period begins. In Experiment 3 we will examine the impact of unpredictability of CORBA ORB on the periodic tasks. Timing accuracy of the kernel timer seems not to be able to affect predictability of real-time CORBA ORB. However, If the sleep time of a periodic task is not the same as we expected due to the imprecise kernel timer, for example, the task sleeps much longer, the job which need to be completed in the next period may not be able to start on time and complete before its deadlines. Since our focus is on the unpredictability of real-time CORBA ORB, we need to choose appropriate timer resolution to ensure that non-determinism caused by lower resolution timer can be ignored and only non-determinism caused by real-time CORBA ORB is exposed. This experiment serve the purpose by examining whether the actual invocation frequency of a periodic thread is the same as we expected under a particular kernel timer configuration.

This experiment reveals that the timer resolution of the operating system has significant impact on the timing of periodic tasks and can cause highly nondeterministic behaviors. Timing of default 100Hz timer and customized 1000Hz timer on Linux are compared in Figure 6.

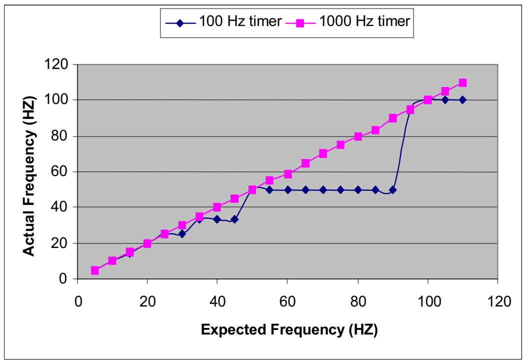

Figure 6

As shown in Figure 6, with 100 Hz timer, the invocation frequency (the number of invocations in one second) varies strangely as expected invocation frequency increases above 20Hz. The anomaly results from the default 10ms timer resolution in Linux. Given that the system call usleep() is called for a sleep of 5 ms , the Linux kernel will round up the actual sleep time to 10ms. The round-up also results in the meeting points between 10Hz timer and 1000 Hz timer in Figure 6. Since the timing of 1000 Hz timer is accurate enough, we will use 1000Hz timer kernel in the following experiments to avoid non-determinism caused by lower resolution kernel timer. Besides the approach which increases interrupt frequency of hardware timer to achieve higher kernel timer resolution, there are some other approaches to improve timer resolution. For example, KURT Linux [10] , TimeSys Linux [7] use one-shot hardware timer to provide microsecond-level timer resolution.

Experiment 3: End-to-end predictability of CORBA invocations

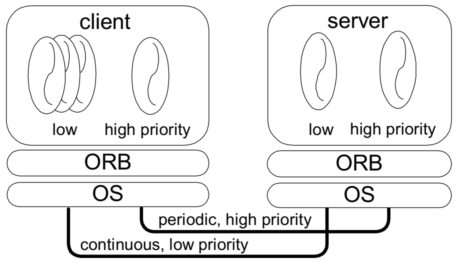

Figure 7

This experiment examines the variance of two-way CORBA invocation latency of the high priority periodic client thread, as the competition for some resources from low priority client threads increases. We examined experimentally the impact of increasing one or more components as follows:

- **Experiment (a):** the number of low priority client threads increased.
- **Experiment (b):** both the number of client threads and network traffic increased.
- **Experiment (c):** impact of background traffic from a separate process.
- **Experiment (d):** server side computation workload increased.

Experiment (a) measures invocation latency of the high priority CORBA client as the number of low priority CORBA client threads increases. The experiment configuration is shown in Figure 7. The server handles the requests using two threads with high and low priorities, which process the high and low priority requests from the client side, respectively. On the client side, a periodic invocation thread is at the high priority while the best-effort continuous invocation threads are at low priority.

The graph in Figure 8 plots the average, maximum invocation latency of the high-priority periodic thread as the number of low priority continuous invocation threads increase when *workload=100* and *payload=0*. Setting *payload=0* implies that the message size of CORBA invocations is small and causes less network traffic.

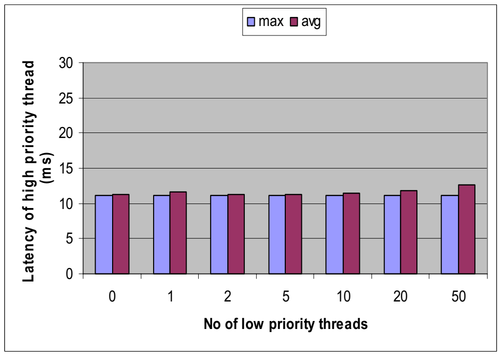

Figure 8

Experiment (b) measures invocation latency when network traffic is increased. The *workload* of CORBA invocation remains unchanged and the *payload* is set to *10K bytes*. The graph in Figure 9 plots the average, maximum invocation latency of the high priority periodic thread.

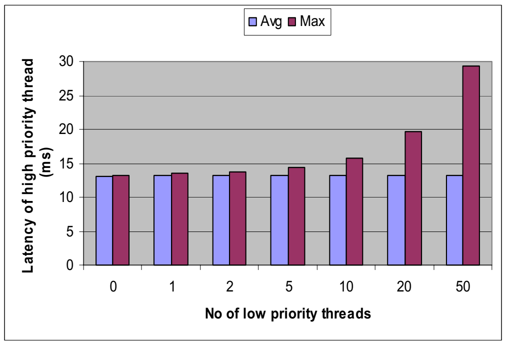

Figure 9

Figure 8 indicates that increasing the number of low priority client threads does not has significant impact on the high priority CORBA client in the case that network traffic is low. Figure 9 shows that priority inversion becomes noticeable as network traffic gets heavy.

Experiment (c) measures invocation latency of the high priority periodic client thread in the presence of heavy background network traffic. The low priority task is a program running in a separate process and generating network traffic as much as possible to occupy the network bandwidth.

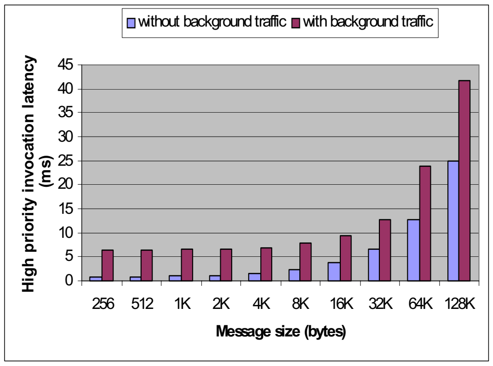

Figure 10

Figure 10 depicts the average invocation latency of the high priority periodic client thread with/without background network traffic. It is observed that the low priority program which produces a large volume of network traffic significantly affects the high priority periodic thread. Invocation latency variation of the high priority thread is large when there is heavy background network traffic.

In above experiments we observe that heavy network traffic can cause significant priority inversion even if the program generating heavy network traffic is in a separate process from the process using TAO ORB. It is well-known that CSMA/CD protocol of Ethernet can cause non-determinism in networks. However, in our experiments the client and server computers are connected together with a point-to-point duplex 100M Ethernet link. This configuration largely excludes non-determinism caused by the Ethernet frame conflicts. Hence, we concluded that the major source of priority inversion lies in the TCP/IP stack rather than in Ethernet or TAO real-time ORB itself.

Experiment (d) examines priority of inversion when server side workload increases.

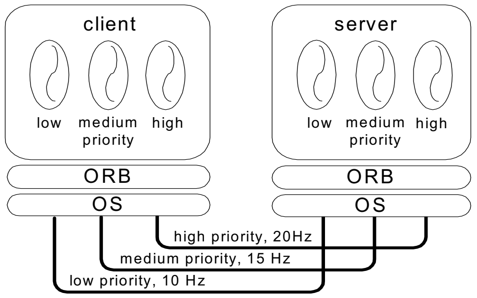

Figure 11

As shown in Figure 11, the server handles the requests using three thread lanes with high, medium, and low priorities. Each lane contains one thread. On the client side, three periodic threads make invocations on a CORBA object on the server at the frequency of 20Hz, 15Hz, and 10Hz with high, medium and low priority, respectively. The deadlines of the periodic invocations are exactly equal to their period. The POA policy “client priority propagation” and “band bounded connection” are enabled. Each CORBA request carries 256 bytes data payload, which will be echoed back in the reply.

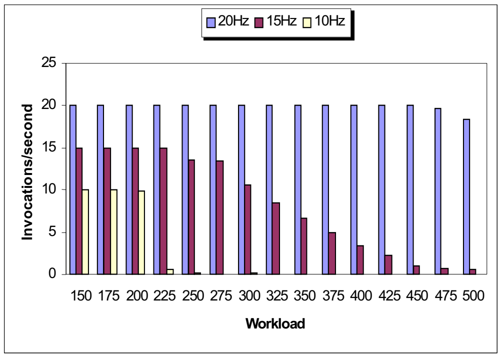

Figure 12

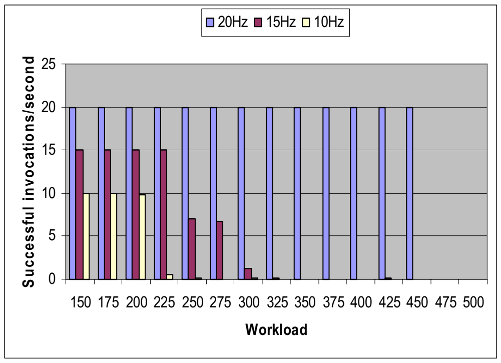

Figure 13

Figure 12 plots the number of actual invocations made per second for three periodic threads and Figure 13 shows the number of invocations that complete within their deadlines. By comparing results in Figure 12 to those in Figure 13, it is found that not all completed CORBA invocations can make their deadlines when CPU workload is heavy. It is observed that all deadlines are met when server side workload is low, e.g. *workload=150*. As the workload is increased, the lowest priority thread misses the deadline first, the medium priority thread follows, with the highest priority thread beginning to miss their deadline last. This behavior indicates that end-to-end predictability is well preserved based on the thread priorities.

3.3 Experiment summary

In above experiments we evaluated the overhead of CORBA invocation of TAO on Linux and the predictability of CORBA ORB of TAO under different resource contention conditions. We also investigated the influence of kernel timer resolution on predictability of periodic tasks. These experiments show that the overhead of TAO real-time ORB is acceptable and TAO ORB can preserve end-to-end predictability and minimize priority inversion in various resource competition conditions except for when the network is overloaded. We further identify that TCP/IP protocol stack of Linux kernel 2.4.18 can cause significant priority inversion when the network is overloaded.

3. Related works

The Distributed Object Computing Group in Washing University, St Louis and California University, Irvine, have published a series of papers about TAO's real-time performance evaluations. [11] [12] [13] are closely related to the topic of this paper.

Paper [11] evaluates priority inversion of TAO on the Linux 2.4 kernel with different ORB and POA configurations supported by real-time CORBA specification. Invocation/second and percentage of deadline misses of periodic tasks are measured as metrics of priority inversion when server-side workload is increased. However, their definition of the term “deadline miss” is less strict and different from the convention in the real-time system area. Therefore the results are not convincing enough for hard real-time system designers.

Paper [12] evaluates TAO's predictability on various OS platforms by examining two-way CORBA invocation latency and its deviation. Paper [13] compares TAO's predictability with other standard CORBA products. However these two papers did not discuss any features specific to the real-time CORBA specification.

Real-time CORBA Trade study [14] from Boeing Phantom Works evaluates three different real-time CORBA products: HARDPack, ORBexpress and TAO on various OS platforms. The project not only assesses the CORBA performance, but also other attributes of software quality, such as platform and language availability, pricing policy, and documentation, etc. The study measures one-way and two-way invocation latency with various message sizes and data types and reports the average, minimum and maximum operation execution times as well as standard deviations. However none of the features specific to the real-time CORBA specification was examined.

4. Concluding Remark

The paper shows that: 1) the overhead of TAO is acceptable, compared with applications using TCP/IP socket directly; 2) TAO real-time CORBA ORB exhibits predictable behaviors and can preserve end-to-end priorities across the network even when multiple CORBA client threads with different priorities simultaneously compete for a CORBA server, provided that there is no heavy competition for network resources. The paper also shows that heavy network traffic can cause significant priority inversion in real-time CORBA applications, where TCP/IP protocol stack is the main source of priority inversion because Linux TCP/IP stack can not distinguish between networking activities caused by the tasks with different priorities.

Since TAO is found not to be a main source of unpredictability, to improve predictability further, the priority-aware or QoS-aware TCP/IP stack and network are required. Before these technologies become more popular, we have to impose extra constraints in the design of distributed real-time systems. For example, we can physically separate the communication links for bulk data transport from those for mission-critical message transport with real-time constraints to ensure the network resources are not overloaded.

5. Acknowledgments

We gratefully acknowledge CSIRO, Australia for sponsoring our research in middleware. Especially, we would like to thank Jeffrey Gosper for his support and direction in this project.

6. References

[1] TAO Real-time CORBA, URL: http://www.cs.wustl.edu/~schmidt/TAO.html.
[2] ORBexpress Real-time CORBA, Objective Interface Systems, Inc., URL: http://www.ois.com
[3] e*ORB Real-time CORBA, PrismTech, URL:http://www.prismtechnologies.com/English/Products/CORBA/eORB/
[4] Visibroker-RT Real-time CORBA, Borland, URL:http://www.borland.com/visibroker_rt/
[5] The Common Object Request Broker: Architecture and Specification, Revision 2.6.1, Object Management Group, 2002.
[6] Comp.RealTime News Group FAQ, URL:http://www.realtime-info.be/encyc/publications/faq/rtfaq.htm#realtime_definition.
[7] TimeSys Linux, TimeSys Inc., URL: http://www.timesys.com/
[8] RTLinux, Finite State Machine Labs, Inc., http://www.rtlinux.com/
[9] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture, Volume 2, Patterns for Concurrent and Networked Objects*: John Wiley & Son Ltd, 2000.
[10] KURT Linux, URL: http://www.ittc.ku.edu/kurt/.
[11] I. Pyarali, D. C. Schmidt, and R. Cytron, "Achieving End-to-End Predictability of the TAO Real-time CORBA ORB," presented at Proceedings of the 8th IEEE Real-Time Technology and Applications Symposium, San Jose, CA, USA, 2002.
[12] D. C. Schmidt, M. Deshpande, and C. O'Ryan, "Operating System Performance in Support of Real-time Middleware," presented at Proceedings of the 7th IEEE Workshop on Object-oriented Real-time Dependable Systems, San Diego, CA, USA, 2002.
[13] D. C. Schmidt, S. Mungee, S. Flores-Gaitan, and A. Gokhale, "Alleviating Priority Inversion and Non-determinism in Real-time CORBA ORB Core Architectures," presented at Proceedings of the Fourth IEEE Real-Time Technology and Applications Symposium, Denver, Colorado, USA, 1998.
[14] *Real-Time CORBA Trade Study*, Boeing Company, 2001.