Evaluating the Performance of CORBA for Distributed and Grid Computing Applications

T. Es-sqalli, E. Fleury, J. Guyard, S. Bhiri
RESEDA Project, LORIA, UHP Campus Scientifique, BP 239
F-54506 VANDOUEVRE-LÉS-NANCY cedex France

Abstract

Distributed computing applications rely on transmissions of messages between processes. However, as workstations are not intended to manage this kind of communications, it is necessary to use some communication tools known as Message Passing libraries. Currently, MPI (Message Passing Interface) and PVM (Parallel Virtual Machine) are the most used.

In addition, the need for interoperability among the rapidly proliferating number of hardwares and softwares led to the definition of CORBA (Common Object Request Broker Architecture), which standardizes the execution support of distributed object applications. This architecture offers a good interoperability and encapsulation support. Moreover, its method invocation mechanism, offers several features and can be used as a good communication model. Consequently, the use of such architecture for distributed and grid computing applications seems interesting and promising. However, CORBA is always criticized on the performances level.

By taking account of all these elements, a performance evaluation of CORBA seems necessary. The topic of this paper is to present our benchmark results of some CORBA implementations (ORBacus and TAO) and to compare them with those of the two message passing libraries MPI (MPICH and LAM) and PVM, by using a Network of Work Stations (NOW).

1. Introduction

CORBA [6] is a distributed object system that defines services and supports which an application needs in order to be carried out in a distributed environment. The communication platform, or ORB (Object Request Broker), is the middleware that establishes the client-server relationships between objects. Using an ORB, a client can transparently invoke a method on a server object. The client does not have to be aware of where the object is located, what programming language and operating system are used, or any other system aspect that does not belong to the object interface. This isolation between client and server makes the great force of CORBA and allows it to cover a large application field. Distributed and grid computing applications may benefit from those advantages. In this context, two frameworks of the use of CORBA seem interesting:

- In terms of services: the challenge for the parallelism community is to make various computational parallel tools and servers accessible by remote users within the framework of a large distributed environment. To achieve this goal, a good interoperability and encapsulation support is necessary. CORBA offers such a support.

- In term of communication support: the method invocation mechanism on remote objects proposed by CORBA offers several features and can be used as a good communication model for development of applications, in particular those based on the message passing programming model. In addition, CORBA offers several communication services [8]: event, notification and messaging services [7]. These services could be used as a communication core in some parallel programming models.

However, if the framework for the use of this standard in distributed and grid computation seems obvious, the problem is about its communication performance. CORBA is always criticized on this point of view.

Certainly, the first implementations of CORBA were not powerful since quality of service was not integrated in the specification of this standard. However, robust implementations of CORBA start to exist and evaluation of their performances seem consequently interesting and necessary. We carried out several benchmarks of the two CORBA implementations: TAO [11] and ORBacus [9]. The choice of these two CORBA implementations is justified by the fact
that they are among the most powerful free CORBA implementations. In addition, they provide an interface with the C++ language with which we made our tests. To evaluate these results, we carried out the same kind of test on some message passing libraries: MPICH [5] and LAM [12] which are two implementations of MPI [4], and PVM [13].

In this paper, we will present these benchmarks tests. But before, we start by giving a short introduction to CORBA architecture.

2. CORBA

The Common Object Request Broker Architecture (CORBA) is an emerging open distributed object computing infrastructure being standardized by the Object Management Group (OMG) to answer to the need for interoperability among the rapidly proliferating number of hardware and software products available today. Simply stated, CORBA allows applications to communicate with one another no matter where they are located or who has designed them [6].

CORBA automates many common network programming tasks such as object registration, location, and activation; request demultiplexing; framing and error-handling; parameter marshalling and demarshalling; and operation dispatching.

CORBA defines the Interface Definition Language IDL and the Application Programming Interfaces API that enable client/server object interaction within a specific implementation of an Object Request Broker ORB.

IDL is the language used to describe the interface that client objects call and that object implementation provides. The ORB is the middleware that establishes the client-server relationships between objects. Using an ORB, a client can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call and it becomes in charge of finding an object that can implement the request, passing it the parameters, invoking its method and returning the results.

In typical client/server applications, developers use their own design or a recognized standard to define the protocol to be used between the devices. Protocol definition depends on the implementation language, network transport and a dozen other factors. ORBs simplify this process. With an ORB, the protocol is defined through the application interfaces via a single implementation language-independent specification, the IDL. Moreover, ORBs provide flexibility. They let programmers choose the most appropriate operating system, execution environment and even programming language to use for each component of a system under construction. More importantly, they allow the integration of existing components.

![Figure 1. Application of type “ping-pong”](image)

With CORBA, users gain access to information transparently, without having to know what software or hardware platform it resides on or where it is located. The communications heart of object-oriented systems, CORBA brings true interoperability to today’s computing environment.

3. Benchmark results

In this section, we will present all benchmarks which we carried out. Before presenting the results of each one and their analysis, we start by describing it. In addition, let us recall that all these tests were carried out by using SUN Ultra 5 stations inter-connected by Ethernet networks with a throughput of 10 Mbit/s.

3.1. Application of type “ping-pong”

3.1.1 Description

The first type of benchmark carried out is a classical “ping-pong” between a client and a server entities. As shown in figure 1, this test consists on sending and receiving of various data sizes between these two entities. To avoid system and network disturbances, we repeat the data exchange process several times. The obtained results will be related to the exchanged data size. Client and server entities will be implemented as objects in CORBA and tasks in the message passing libraries.

The concept of client and server is not concrete enough since the server application doesn’t present services which the client application doesn’t have. Indeed, these two entities present the same services of sending and receiving data. The difference is that on the level of CORBA, server applications make public the references (IOR) of the objects which they create. Hence, objects created by the client applications can reach them. On the level of the message passing libraries, tasks created by clients applications search for task identifiers of those created by server applications. Thus, it is the same type of application which will be implemented on both sides.

289
3.1.2 Benchmark results and analysis

For this type of test, considering exchanged data size varies from 1 byte to 4MB and the obtained results present different performances phases, we choose to present these results by progressives parts. We will denote by a "small data sizes" those going from 1 byte to 1KB, "medium data sizes" those from 1KB to 200KB and finally "big data sizes" those which are more than 200KB.

Figure 2 shows the performances of the various tested components for small data sizes.

![Figure 2. Communication times for a "ping-pong" application for small data sizes](image)

We see that for these data sizes that MPICH followed by PVM presents the best communication times compared to those of ORBacus, which is in the third place, then LAM and TAO. However, the range of data between 2KB and 16KB contains several thresholds points from which the ranks change and the performances reverse. Indeed, Table 1 shows that starting from 4KB, MPICH, which was the most powerful, starts to degrad and to become the worst.

![Table 1. Communication times for a "ping-pong" application for medium data sizes](image)

The same scenario is repeated for PVM which starts to degrade from 3KB. Contrary to the behavior of MPICH and PVM behaviors, LAM, ORBacus and especially TAO become more powerful when increasing the data size. ORBacus maintains the best performances with TAO until 32KB from which LAM becomes better than ORBacus. For TAO, starting from 10KB, it becomes the most powerful and finishes in first rank with LAM. Figure 3 shows the various performances for big data sizes.

![Figure 3. Communication times for a "ping-pong" application for big data sizes](image)

As conclusion for this type of tests, we can say that although MPICH followed by PVM present the best performances for small data sizes, they start to degrade quickly and lose their places to ORBacus which maintains a good performance with TAO until 32KB. From this value, LAM becomes more powerful than ORBacus. For the big data sizes, LAM and TAO present the best performances.

In order to focus on the parallelism aspect in our tests, we develop two other applications which are presented as a generalization of a classical "ping-pong". The first application consists of the creation of several clients interacting at the same time with only one server, while the second application consists of the creation of several client/server couples interacting between them at the same time.

3.2 Application of type several clients/one server

3.2.1 Description

This type of application consists of several "ping-pong" between several clients (located on different work stations) and only one server. Interaction between clients and server is realized by sending and receiving various data sizes. This type of test aims to model the behavior of the server when it has to manage several requests at the same time. Figure 4 illustrates such type of application.
3.2.2 Benchmark results and analysis

The curves obtained will be three dimensional since the results will be related to the exchanged data size and the number of clients. However, for reasons of clarity, we choose to present them only in two dimensions: by fixing the number of clients or the exchanged data size.

Figure 5 and Table 2 present communication times for an application of type 25 clients/one server for data sizes between 1 Byte and 1 MB.

<table>
<thead>
<tr>
<th>Data size (Byte)</th>
<th>1</th>
<th>512</th>
<th>65536</th>
<th>262144</th>
<th>1048576</th>
</tr>
</thead>
<tbody>
<tr>
<td>lam</td>
<td>18.2</td>
<td>66.2</td>
<td>1254.8</td>
<td>5731.8</td>
<td>21762.03</td>
</tr>
<tr>
<td>mpich</td>
<td>7.3</td>
<td>34.9</td>
<td>1394.3</td>
<td>6483.1</td>
<td>32111.14</td>
</tr>
<tr>
<td>orbacus</td>
<td>26.1</td>
<td>83.7</td>
<td>983.1</td>
<td>4737.1</td>
<td>22623.77</td>
</tr>
<tr>
<td>pvm</td>
<td>14.0</td>
<td>69.4</td>
<td>2106.6</td>
<td>8038.9</td>
<td>30832.38</td>
</tr>
<tr>
<td>tao</td>
<td>17.1</td>
<td>30.9</td>
<td>1151.8</td>
<td>5351.0</td>
<td>18893.1</td>
</tr>
</tbody>
</table>

Table 2. Communication times for an application of type 25 clients/one server

By comparing these results with those of a classical "ping-pong", we notice the robustness of TAO server compared to the others ones. If during the "ping-pong" application, LAM and TAO finish together in first rank, for the application of type 25 clients/one server, TAO becomes more powerful than LAM and present better performances. Moreover, ORBacus, which was less powerful than LAM in a classical "ping-pong" presents the same performances as LAM for an application of the type 25 clients/one server.

In order to clarify this idea and to better see the behavior of the servers according to the number of clients, we choose to present some results in another way by fixing data size and varying the number of clients. Figure 6 shows communication times for data size equal to 1MB and number of clients varying from 1 to 25. This figure illustrates the remarks evoked before: the robustness of TAO, the similar performances of ORBacus and LAM and finally the degradations of the performances for MPICH and PVM.

Since this type of test is a generalization of the first one, we notice the similarity of these curves with those presented for a classical "ping-pong". In a first step, MPICH followed by PVM present the best communication times. Then in a second step their performances start to degrade and become the worst. In addition, ORBacus obtains the best performances for medium data sizes (16KB to 300KB). For big data sizes, we find TAO more powerful than the others followed by LAM and ORBacus which carry out similar communication times, then in last ranks MPICH and PVM.
Table 3 shows communication times for an application of the type 10 client/server couples for data sizes from 1 Byte to 1 MB. Figure 8 illustrates these results.

<table>
<thead>
<tr>
<th>Data size (Byte)</th>
<th>1</th>
<th>512</th>
<th>65536</th>
<th>262144</th>
<th>1048576</th>
</tr>
</thead>
<tbody>
<tr>
<td>lam</td>
<td>6.8</td>
<td>10.0</td>
<td>589.1</td>
<td>2357.8</td>
<td>9145.44</td>
</tr>
<tr>
<td>orbacus</td>
<td>5.6</td>
<td>8.7</td>
<td>578.7</td>
<td>2305.7</td>
<td>9286.5</td>
</tr>
<tr>
<td>tao</td>
<td>6.2</td>
<td>7.8</td>
<td>548.8</td>
<td>2150.2</td>
<td>8477.7</td>
</tr>
<tr>
<td>pvm</td>
<td>5.7</td>
<td>16.5</td>
<td>888.1</td>
<td>3560.7</td>
<td>14016.55</td>
</tr>
<tr>
<td>mpich</td>
<td>3.3</td>
<td>13.2</td>
<td>642.1</td>
<td>3006.0</td>
<td>12382.35</td>
</tr>
</tbody>
</table>

Table 3. Communication times for an application of type 10 client/server couples

One notices the difference in performances between the set gathering TAO, ORBacus and LAM and the second set containing PVM and MPICH. This difference will be accentuated when increasing data size.

Figure 8 clarifies the difference of the performances between these two sets. We find TAO more powerful than the others, followed by LAM, ORBacus, MPICH and finally PVM.

3.3. Application of type several client/server couples

3.3.1 Description

This type of application is also presented in the form of several “ping-pong”. These “ping-pong” are carried out between several clients and several servers. All clients are carried out on a work station and all servers on another one. So this application consists in creating at the same time of several interactions between several client/server couples and not between several clients and only one server as in the last application.

In addition, to facilitate the message passing programming mode (MPI and PVM codes), we chose to gather all clients in a group and all servers in another one.

The results for this type of application will be related to the number of client/server couples and the exchanged data size. Figure 7 illustrates such type of test.

The aim of this type of tests is to model the behavior of the ORB when it has to manage several communications between several objects at the same time and to compare this behavior with those of MPI and PVM daemons.

3.3.2 Benchmark results and analysis

The obtained curves will be presented on three dimensions since results will be related to the exchanged data size and the number of client/server couples. As for application of type several clients/one server, and for reasons of clarity, we choose to present them on two dimensions: by fixing the number of client/server couples or the exchanged data size.
times. Therefore, we carried out this type of test without using groups. This resulted in almost the same performances as with construction of groups. Figure 9 illustrates the results obtained for MPICH with and without creation of groups for 5 client/server couples.

![Figure 9. Comparison of MPICH performances for 5 client/server couples application with and without construction of groups](image)

These results show that the creation of group is not the factor responsible for the bad performances of MPICH and PVM libraries for this type of tests.

In order to answer the initial objective of these tests, we present some results in a different way, by fixing data size and varying the number of client/server couples from 1 to 10. We choose to have these results in such a way to compare the behavior of the ORB with those of PVM and MPI daemons when increasing the number of client/server couples. Figure 10 shows communication times for data size equal to 1MB and number of client/server couples varying from 1 to 10.

![Figure 10. Communication time for an application of type several clients/server couples according to the number of client/server couples for data size of 1MB](image)

We notice the difference between the set of TAO, ORBacus and LAM and that of PVM and MPICH. This figure confirms the performances of the preceding figure (8). We can conclude that the ORB, in particular TAO, is more robust than PVM and MPI daemons because it supports better several communications simultaneously.

We remind that the performances of TAO, during all the tests, are obtained thanks to its architecture. Indeed, TAO is a real time CORBA implementation which uses an efficient communication platform called ACE (Adaptive Communication Environment) [10] (see figure 11). This platform provides a set of reusable software components on several architectures and opened towards several communication interfaces: sockets, shared memory, pipe, etc.

![Figure 11. TAO architecture](image)
4. Conclusion

In this paper, we showed the interest in term of communication performances of the use of CORBA for the development of distributed and grid computing applications. If the advantage of this architecture in term of interworking and interoperability seems obvious and undeniable, the problem is on the level of its communication performances. For this reason, we developed a set of benchmarks by using two implementations of CORBA (TAO and ORBacus) and we compared them to those of the message passing libraries MPI (LAM and MPICH) and PVM.

We developed three types of benchmark, the first consists of a classical “ping-pong” of variable data sizes. The results showed that for small data sizes, MPICH and PVM present the best communication times but they start to degrade quickly. ORBacus maintains a good performance with TAO until 32KB. For big data sizes, LAM and TAO present the best performances.

In the second type of benchmark, we want to model the behavior of the server when it communicates simultaneously with several clients. The results showed, by increasing the number of clients, the robustness of TAO compared to the others, then the similar performances of ORBacus and LAM and finally degradations of the performances for MPICH and PVM.

The last benchmark, which consists of several “ping-pong” between client/server couples, aims to model the behavior of the ORB when it has to manage several communications between several objects at the same time and to compare this behavior with those of MPI and PVM daemons. This benchmark illustrated the performance of TAO and noticed the difference in performances between the set gathering TAO, ORBacus and LAM and the second set containing PVM and MPICH.

We notice that CORBA present competitive implementations in term of communication performances, compared to the message passing libraries. The ORB can better manage several communications simultaneously than the message passing libraries.

Based on these benchmark results, the use of CORBA for the development of distributed and grid computing applications seems interesting and can be obtained by adding a layer of abstraction and developing tools to the top of CORBA which implement various services and primitives for distributed computation. That seems very beneficial and important since one ensures in addition to the interworking and the portability, good performances. RESEDA (INRIA-Lorraine, France) project contributes in this research orientation. Our contribution concerns the development of a message passing library in CORBA environment called MPC (Message Passing in CORBA) [3] [2]. This library is based on the use of the methods invocation mechanism on remote objects of CORBA.

In addition, the benchmarks which we carried out present a starting point for other tests, in particular by using a quick networks. Within this framework, we will try to use VTHD [1] (Very High Broadband Network Service) which is a high performance network that provides nationwide high capacity interconnection facilities among laboratories at the IP level and that supports experiments for new designs for networking.

References