Performance evaluation of communication software systems for distributed computing

To cite this article: R A Fatoohi 1997 Distrib. Syst. Engng. 4 169

View the article online for updates and enhancements.

Related content
- The implementation of a high-performance ORB over multiple network transports
  Sai-Lai Lo and Steve Pope
- The design and performance of MedJava:
  Experience of developing performance-sensitive distributed applications with Java
  Prashant Jain, Seth Widoff and Douglas C Schmidt
- Experiences on integration of network management and a distributed computing platform
  Sakari Rahkila and Susanne Stenberg

Recent citations
- Online weather data analysis and visualization tools for applications in ecoinformatics
  Krisanadej Jaroensutasinee et al
Performance evaluation of communication software systems for distributed computing

R A Fatoohi†

Computer, Information, and Systems Engineering, San Jose State University, San Jose, CA 95192, USA

Abstract. In recent years there has been an increasing interest in object-oriented distributed computing since it is better equipped to deal with complex systems while providing extensibility, maintainability and reusability. At the same time, several new high-speed network technologies have emerged for local and wide area networks. However, the performance of networking software is not improving as fast as the networking hardware and the workstation microprocessors. This paper gives an overview and evaluates the performance of the Common Object Request Broker Architecture (CORBA) standard in a distributed computing environment at NASA Ames Research Center. The environment consists of two testbeds of SGI workstations connected by four networks: Ethernet, FDDI, HiPPI and ATM. The performance results for three communication software systems are presented, analysed and compared. These systems are: BSD socket programming interface, IONA’s Orbix, an implementation of the CORBA specification and the PVM message passing library. The results show that high-level communication interfaces, such as CORBA and PVM, can achieve reasonable performance under certain conditions.

1. Introduction

One of the most interesting emerging technologies is object-oriented distributed computing. Current technology based on systems like the Distributed Computing Environment (DCE) and using the Remote Procedure Call (RPC) mechanisms does not provide adequate support for building a real, complex, scalable, distributed computing environment. For example, the DCE model uses a procedure-oriented distributed model while the CORBA model is based on the use of objects and uses an object-oriented distributed model. Also, despite the fact that RPC is an important part of most distributed systems including the object-oriented systems, it is still a relatively low level interface to the distributed computing environment. The Common Object Request Broker Architecture (CORBA) specification provides the mechanisms for objects to interact transparently.

Another emerging technology is high-speed networks. Emerging networks have peak transmission rates of about one hundred megabytes per second. Combined with high connectivity (switched network), the aggregate network bandwidth can reach several gigabytes per second.

The distributed object technology along with high-speed networks represent two key components of the information infrastructure for next generation applications. The object-oriented technology provides the extensibility, maintainability and reusability that are needed in software engineering while the high-speed network technology provides the required speed to transmit information between systems. These technologies are essential in many applications such as multimedia, virtual manufacturing, digital library, air traffic control simulation and virtual computing. At this time, it is important to evaluate these technologies and identify their limitations since some products, based on these technologies, have started to appear in the commercial market.

One of the few studies of the performance of CORBA on high-speed networks is the work by Schmidt et al [9] where they compared the performance of BSD sockets and two implementations of CORBA (Orbix from IONA Technologies and ORBeline from Post Modern Computing) over Ethernet and ATM networks using two SPARCstations. They showed that CORBA based implementations of tcp were slower than BSD socket implementations. Also, they reported good results using the ACE toolkit.

This paper is the first step in evaluating distributed systems and analysing different components of these systems. Here the emphasis is on examining the performance of communication software systems under different conditions and identifying factors that influence their performance. The communication tests were chosen to study such factors as: message size, socket buffer size and data type. Two communication systems are considered here: Orbix, an implementation of the CORBA specification, and the PVM message passing library. The
performance results from these two systems are compared with the results obtained from using the BSD socket programming interface.

There are many similarities and differences between CORBA and PVM. Both CORBA and PVM are based on the BSD socket programming interface. Also, both systems consist of communication libraries and a run-time system (daemon). In addition, both systems can be configured to run on a single processor, for testing and debugging, as well as on heterogeneous platforms of different machines and networks. Moreover, both systems can be used as high-level network programming interfaces for building a large distributed system. Finally, both systems provide similar functionalities in moving data, except that PVM is richer in group communication.

The main difference between PVM and CORBA is that PVM, like DCE, is based on a procedure-oriented distribution model while CORBA is based on an object-oriented distribution model. Also, PVM was designed for networks of workstations as well as parallel computers while CORBA was designed for distributed computing using a client/server model. Both systems, with certain modifications and added services, can work in both environments. Despite these differences, a study of these systems would give an insight of the performance issues that users might face.

Experiments were performed on two testbeds consisting of Silicon Graphics Inc. (SGI) workstations connected by four networks at the Numerical Aerodynamic Simulation (NAS) Systems division at NASA Ames Research Center. These networks are: a 10 Mbit/s Ethernet, a 100 Mbit/s FDDI, an 800 Mbit/s HiPPI and a 155 Mbit/s ATM.

The paper starts with an overview of CORBA and PVM, followed by a brief description of the hardware platforms and the networks. Next, the performance results for the three communication software systems are presented and modelled. Finally, we offer some concluding remarks.

2. Context

2.1. CORBA

The Common Object Request Broker Architecture (CORBA) is a standard for transparent communication between application objects. The CORBA specification is being developed by the Object Management Group (OMG) [8, 10, 11]. The OMG is an organization of over 700 software vendors and users involved in the development of object technology for distributed computing systems. The OMG does not produce any software, only specifications which come from OMG members who respond to Requests for Proposals (RFP).

In 1990, the OMG introduced a reference architecture for object oriented applications called the Object Management Architecture (OMA). The OMA consists of four components: Object Request Broker (ORB), Object Services, Common Facilities, and Application Objects. The OMA object model is a client/server model where the servers are objects that provide services to clients. The clients obtain services by invoking operations on server objects.

The ORB, a key piece of the OMA, is a mechanism that provides transparency of object location, activation and communication. It also provides interoperability between applications in heterogeneous distributed environments. Object Services, which sit close to the ORB, provide basic services for using and implementing objects. Among the Object Services are Object Naming, Object Events, Object Life cycle and Object Concurrency Services. Common Facilities, which sit close to the user, provide some generic functions for specific requirements. Among the Common Facilities are System Management, User Interface and Presentation Facilities. Unlike Object Services, Common Facilities are optional. Finally, Application Objects are objects specific to certain applications, which may be built from other objects such as Object Services and Common Facilities.

The CORBA specification, introduced in 1991, describes the interfaces and services that ORBs must have; i.e., CORBA is basically the technology adopted for ORBs. CORBA provides a clean model where the interface of an object and its underlying implementation are separated; clients do not need to know how or where servers are implemented. Server objects are visible only through interfaces and object references. Interfaces describe the services provided by an object and object references identify objects. An ORB uses object references to identify and locate objects to forward requests to them.

The CORBA specification has several components: ORB Core, Interface Definition Language (IDL), Dynamic Invocation Interface (DII), Interface Repository and Object Adapters. The CORBA 2 specification, introduced in late 1994, added more components, mainly inter-ORB interoperability and the Dynamic Skeleton Interface (DSI). The DSI provides a run-time binding mechanism for servers.

The ORB Core handles requests for remote objects. It uses object references to locate objects, activates them (if they are not already active), delivers the request to them, transfers control to them, and finally returns any output values to the client.

IDL is a declarative language for describing the interfaces of CORBA objects with a syntax resembling that of C++. It is a subset of C++ with C++ implementation constructs removed and with extensions for distributed programming. It supports multiple interface inheritance. IDL interfaces contain attributes and operations used to define services provided by objects. An IDL compiler maps IDL constructs into a specific programming language (such as C++ or C) based on CORBA language bindings. The compiler generates the client and server code, called client stubs and server skeletons, needed to implement the interface.

In the IDL to C++ mapping, each interface is mapped into a C++ class while each operation is mapped into a C++ member function. Each read–write attribute is mapped into two member functions (get and set) whereas each read-only attribute is mapped into a get function only.

Interface operations may be invoked either statically through the compiler generated stubs or dynamically through DII. The DII is a mechanism for clients to
make calls on objects with no compile-time knowledge of their interfaces. It is usually more flexible but more complicated and less efficient than the static invocation interface (through client stubs).

The Interface Repository provides persistent storage for IDL interface declarations. It provides run-time information about the interface properties of objects. The Object Adapters provide the means for object implementations to access ORB services. Among the ORB services are object reference generation, object operation invocation, activation and deactivation of objects, and security. CORBA specifies that a standard adapter called the Basic Object Adapter (BOA) should be provided by every ORB.

Object invocation in CORBA can be done synchronously (blocking), asynchronously (non-blocking), or one-way (best-effort). In a synchronous communication, the sender sends a request and waits until the request either completes or fails. In an asynchronous communication (called deferred synchronous communication in CORBA terminology), the sender sends a request and proceeds with other work (does not wait) but it must check periodically for a response. Asynchronous communication is supported under the DII only. In a one-way communication, the sender sends a request and proceeds with other work without checking for a response. Here the receiver does not return a value to the sender.

2.1.1. Orbix  Orbix is a library based implementation of the CORBA specification from IONA Technologies Ltd [4]. It is implemented mainly by two sets of libraries (client and server libraries) and a daemon, orbixd. The server library can send and receive remote object requests while the client library can only send requests. The daemon needs to run on the server’s host side so it can start server processes dynamically. Orbix also has an IDL compiler, Interface Repository and many utilities. The Orbix IDL compiler generates a C++ class for an IDL interface to be used by client programs. The Orbix communication classes are implemented using TCP/IP, XDR and a simple message protocol.

Orbix runs on MS Windows as well as several Unix platforms, including Sun Solaris, SGI IRIX, IBM AIX, HP UX and DEC Ultrix. The implementation tested is Orbix version 1.3.3.

2.2. Parallel Programming System: PVM

The Parallel Virtual Machine (PVM) is a collection of public-domain system software routines that enables parallel processing on a network of heterogeneous computers as well as parallel computers [2]. It is composed of two parts: a run-time system (daemon) that resides on all of the computers participating and a set of user interface primitives that can be incorporated into C (or Fortran) code. This includes primitives for process control, message passing and synchronization between processes running on different machines.

PVM daemons communicate with one another through UDP sockets while a PVM task communicates with its daemon over a TCP connection. Also, PVM tasks can communicate with each other by establishing a direct TCP link between the tasks. Our implementation is based on PVM version 3.3.10 using a direct TCP link between PVM tasks.

PVM supports both synchronous and asynchronous communication forms. In addition to these point-to-point types of communication, PVM supports group communication such as multicast, to a set of tasks, and broadcast, to a user defined group of tasks.

3. Test platforms

3.1. Workstation hardware

Two platforms were used for this study: a cluster of SGI workstations, called DaVinci, and two SGI workstations connected by an ATM switch, called here the ATM testbed. The DaVinci cluster consists of nine (one front end system and eight compute nodes) SGI Power Challenge machines with MIPS R8000 CPUs. The front end is a four-cpu 75 Mbit/s network that uses variable sized packets. Asynchronous communication forms. In addition to these point-to- point types of communication, PVM supports group communication such as multicast, to a set of tasks, and broadcast, to a user defined group of tasks.

3.2. Network hardware

Briefly, Ethernet is a 10 Mbit/s broadcast bus technology while Fiber Distributed Data Interface (FDDI) is a 100 Mbit/s fibre optic token ring network. Both Ethernet and FDDI are connectionless networks. The maximum frame size for Ethernet is 1500 byte while the maximum frame size for FDDI is 4500 byte. High Performance Parallel Interface (HiPPI) is a point-to-point link that uses twisted-pair copper cables with a maximum length of 25 metres and a transfer rate of 800 Mbit/s. HiPPI is a connection-oriented network that uses variable sized packets. Asynchronous Transfer Mode (ATM) is a connection-oriented protocol that uses fixed length cells of 53 byte, with 48 byte of payload. ATM uses an adaptation layer to frame cells into 9180 byte frames. Both HiPPI and ATM are switched networks while Ethernet and FDDI are shared medium networks. However, both Ethernet and FDDI have switching technology available.
4. Performance results

4.1. BSD sockets

Several communication tests were performed on the two platforms using two C programs: ttcp and bench. Both programs use BSD sockets. The ttcp program [7] measures point-to-point data transfer throughput using either TCP or UDP protocols. It uses bulk transfer where the data flow in one direction while the sender sends a pre-specified number of messages. It has many options including: message size, socket buffer size and number of messages. There is also an environment variable TCP_NODELAY which can be set to control the buffering of data on the sending side. In this work, ttcp was chosen to measure throughput using TCP with TCP_NODELAY disabled (by default).

The bench program [6] implements two types of tests: bulk transfer and round-trip. In a bulk transfer, which is similar to ttcp, a number of messages are transferred back-to-back through the network. When the transfer completes, the receiver sends a single message back to the sender for acknowledgment. In a round-trip test, messages are sent (one at a time) from one machine to another, then echoed back. In this work, the round-trip test was chosen to measure latency with UDP because of the simple nature of the protocol.

The throughput measure (using ttcp) was conducted for different message sizes and socket buffer sizes. The message size was varied (through doubling) from 1 to 64 Kbyte. Two socket buffer sizes (8 and 64 Kbyte) were considered, except for HiPPI where 1 Mbyte buffer size was also considered since a previous work [1] showed that HiPPI needs a larger buffer to achieve good performance. Each test was run for at least 20 s to produce reliable data. All measurements were obtained under conditions of light network traffic. However, we noticed some fluctuations in the timing results.

Figure 1 shows the performance results using BSD sockets for the following networks: Ethernet, FDDI, HiPPI on DaVinci and ATM on the ATM testbed, using 8, 64 and 1024 (HiPPI only) Kbyte socket buffer sizes. The best achieved performance results are with HiPPI, especially with the 1 Mbyte socket buffer where it outperformed both ATM and FDDI by an order of magnitude and Ethernet by two orders of magnitude. Performance differences between FDDI and ATM are within 30% where FDDI outperformed ATM for the larger buffer size while the latter outperformed the former for the smaller buffer.

Figure 1 also shows that better performance was achieved using the larger buffer size, except for Ethernet where the 8K results outperformed the 64K results by over 20%. The impact of the buffer size is more significant for HiPPI where high performance was achieved only using large buffer size (1 Mbyte) and large messages (16 Kbyte and larger). The differences in performance between the 8K and 64K socket buffer results are more than a factor of two for FDDI while it is less than that for ATM. The impact of the socket buffer size can be attributed to the TCP window size since TCP breaks up the data into segments. Larger window sizes allow the transmission of multiple TCP segments (fill the pipe) before an acknowledgment arrives.

A simple linear regression model was developed for the ttcp results. The model is based on the following equation: 

\[ T = b_0 + b_1 \times m, \]

where \( T \) is the predicted cost for a message of length \( m \), \( b_0 \) is the intercept of the line, and \( b_1 \) is the slope. For round-trip measurements, \( b_0 \) could be considered as the cost of a zero-byte message but for bulk transfer, as in ttcp, \( b_0 \) has no real meaning. The inverse of \( b_1 \) is the maximum achievable throughput, \( r_{\text{max}} \), for that test. The goodness of a regression is measured by the coefficient of determination, \( R^2 \). The higher the value of \( R^2 \), the better the regression; for a perfect model \( R^2 \) is 1.

The regression coefficients \( b_0 \) and \( b_1 \) are estimates for a single test. In order to obtain better estimates, the 90% confidence intervals for \( b_1 \) and \( r_{\text{max}} \) are computed [5]. The meaning of the 90% confidence intervals for \( r_{\text{max}} \), for example, is that we can state with 90% confidence that the maximum achievable throughput of a network is between two values and the chance of error in this statement is 10%.

Table 1 lists the values of \( r_{\text{max}} \), the 90% confidence intervals for \( r_{\text{max}} \), and \( R^2 \) obtained from applying the regression model on the observed results of figure 1 for the specified buffer sizes. The values of \( R^2 \) show that the predicted results matched the observed results very well. Table 1 also shows the startup latency, \( t_0 \), which is the time required to send a message of minimum size and was measured using the round-trip test of the bench program (using an eight-byte message).

The results of figure 1 and table 1 show that Ethernet, FDDI, HiPPI and ATM have achieved over 80% of their peak rates provided that the limiting factor for the ATM performance is the EISA bus.

4.2. CORBA/Orbix

Performance of Orbix version 1.3.3 was measured using two Orbix programs: ttcp/Orbix and timer. The program ttcp/Orbix [9] is a modified version of ttcp which replaces all C socket calls with stubs and skeletons generated from two CORBA IDL definitions for messages: sequence and string. Unbounded IDL sequences are basically dynamically sized arrays while a string is a sequence of...
char. The program ttcp/Orbix, like ttcp, measures the bulk transfer rate of a network for different message sizes and socket buffer sizes. This is accomplished because Orbix provides a mechanism to change the buffer size through a user-defined callback function. For more details about the code, see [9].

The timer program, originally written by Mokkapati, measures both bulk transfer and round-trip rates for different data types. This program was modified to measure the round-trip time for a zero-length message and throughput for variable length data types.

<table>
<thead>
<tr>
<th>Network</th>
<th>( r_{max} ) (Mbit/s)</th>
<th>( r_{90%} ) confidence intervals (Mbit/s)</th>
<th>( R^2 )</th>
<th>Buffer size (Kbyte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet</td>
<td>8.3</td>
<td>8.3, 8.3</td>
<td>1.000</td>
<td>550</td>
</tr>
<tr>
<td>FDDI</td>
<td>89.4</td>
<td>88.5, 90.4</td>
<td>1.000</td>
<td>561</td>
</tr>
<tr>
<td>HiPPI</td>
<td>716.5</td>
<td>710.2, 722.8</td>
<td>1.000</td>
<td>593</td>
</tr>
<tr>
<td>ATM</td>
<td>70.0</td>
<td>69.8, 70.4</td>
<td>1.000</td>
<td>360</td>
</tr>
</tbody>
</table>

Table 1. Network parameters using ttcp/C.

A linear regression model, similar to the BSD sockets model, was developed for the Orbix results. The results of applying the model for the 64 Kbyte buffer are listed in table 2. Also, the startup latency, defined in the previous section, is given in the table. The latency was computed using the round-trip test of the timer program (zero and one byte messages).

The results of figure 2 and table 2 show that Ethernet, FDDI and ATM under Orbix achieved over 70% of their peak rates while HiPPI achieved only 16% of its peak rate. The problem with HiPPI is that it needs large buffer sizes to run efficiently and that could not be achieved under Orbix. Latency under Orbix is about three times that of the BSD sockets for all networks.

A comparison between Orbix and BSD sockets results for the same socket buffer size shows a drop in performance of up to a factor of three depending on the message and socket buffer sizes. The values of \( r_{max} \) for FDDI, HiPPI and ATM under Orbix are about 70% to 85% of their values under BSD sockets for the same socket buffer size whereas the differences are insignificant for Ethernet.

The CORBA overhead can be attributed to many factors, including: data copying, presentation layer conversion, demultiplexing and memory allocation [9]. The UNIX profiler prof was used to give some insight into the sources of CORBA overheads. The prof program gives some estimates of the amount of time spent in every function of a program. Even though the prof results were not conclusive and some abnormality was observed, it was noticed that a reasonable percentage of the execution time was spent in memcpy. This shows that there were many memory copy operations and the IDL stubs and sequences may copy data several times.

Table 3 shows performance variations under Orbix for transferring sequences of struct and char using the timer program. In IDL, as in C and C++, a struct data type allows related data types to be packaged together. These results show that the use of struct, instead of char, caused a significant performance drop for all networks using both

![Figure 2. Network performance using ttcp/Orbix.](image-url)
Table 3. Network performance (in Mbit/s) using Orbix (16 Kbyte message size).

<table>
<thead>
<tr>
<th>Network</th>
<th>Bulk transfer</th>
<th>Round-trip</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>char</td>
<td>struct</td>
</tr>
<tr>
<td>Ethernet</td>
<td>6.7</td>
<td>5.4</td>
</tr>
<tr>
<td>FDDI</td>
<td>84.4</td>
<td>15.6</td>
</tr>
<tr>
<td>HiPPI</td>
<td>84.0</td>
<td>14.3</td>
</tr>
<tr>
<td>ATM</td>
<td>60.0</td>
<td>18.0</td>
</tr>
</tbody>
</table>

bulk transfer and round-trip communications. This drop is due to the overhead in marshalling and demarshalling structs under Orbix. This was also observed [3] using SPARCstations.

4.3. PVM

Performance of the four networks under the PVM message passing library version 3.3.10 was measured using a C program, called ring; see [1] for more details. In ring, the processors form a ring where each processor receives a message of prescribed length from a previous processor and sends the same message to the next processor. Only one message goes around the ring at any given time. This program measures point-to-point performance and latency of a network.

Figure 3 shows the performance results for the four networks under PVM for message sizes ranging between 1 and 64 Kbyte. Under PVM, HiPPI outperformed the other networks but both HiPPI and ATM achieved only small fractions of their peak rates. The ATM performance under PVM peaked at around 25 Mbit/s using a 16 Kbyte message and then dropped significantly for larger messages (not shown in the figure).

A linear regression model, similar to the BSD sockets and Orbix models, was developed for the PVM results. Table 4 lists the results of applying the model on the PVM results. Latency under PVM is less than 1 ms for all networks. It is below the latency under Orbix.

Performance of PVM is slightly below Orbix for FDDI and HiPPI while it is significantly below Orbix performance with ATM. One of the reasons for these differences is the ability to change the socket buffer size under Orbix while early experiments in changing it under PVM did not show a performance improvement.

The PVM overhead can be attributed to many factors, including: buffer management, connection management and state maintenance. These overheads are more apparent in new networks (such as HiPPI and ATM) than in traditional networks (such as Ethernet and FDDI) since software packages (such as PVM) are designed, tested and optimized more for the traditional networks than for the newer ones.

5. Concluding remarks

High-level network programming interfaces, such as CORBA, provide many advantages over low-level, non-typesafe programming interfaces, such as the BSD sockets. Among these advantages are extensibility, maintainability and reusability. These high-level interfaces have been traditionally less efficient than the low-level interfaces, because of layers of software, especially on high-speed networks. This study showed that it might still be the case but reasonable performance can be achieved. Also, users might be willing to accept a certain performance penalty given all the benefits they are gaining from using these high-level interfaces. However, users have to be aware of some of the performance restrictions that are associated with the use of certain data types. For example, the use of IDL strings as well as structs carries some performance penalties compared to the use of sequences and chars.

Performance differences between CORBA and PVM are not that significant. The choice between the two depends on many other factors including the programming model, whether it is procedure-oriented or object-oriented. Their performance on high-speed networks suffers due to software overheads. However, performance of high-level interfaces is not fixed—it keeps improving. There have been many studies on how to improve communication software, such as compiler optimization techniques and using light-weight protocols. Some of these optimizations are being utilized in commercial network interfaces and operating systems.

The emphasis of this study is performance of communication software systems, rather than real applications. More work needs to be done using real applications to have a better understanding of the limiting factors of communication software at the application level.
Acknowledgments

I would like to thank Sandra Johan of NASA Ames and Noemi Berry of MRI, Inc. for many technical discussions about CORBA and ATM. I am also thankful to IONA Technologies Ltd for providing me with an evaluation copy of Orbix 1.3.3. This work was funded through NASA contract NAS 2-14303.

References