# Performance Evaluation of Popular Distributed Object Technologies for Java

Satoshi Hirano† Yoshiji Yasu†† Hirotaka Igarashi†††

†HORB Open and Electrotechnical Laboratory
hirano@etl.go.jp
http://ring.etl.go.jp/openlab/horb/

††HORB Open and High Energy Accelerator Research Organization(KEK)
Yoshiji.YASU@kek.jp

†††HORB Open and Aoyama Gakuin University
hiro@noa.sipeb.aoyama.ac.jp

**Abstract**

System development using Java and distributed object technology (DOT) is becoming common, and can become the standard way of doing network computing in the near future. The performance of DOTs is crucial in high-performance network computing systems. In this paper, the performance of popular DOTs for Java is evaluated in a common environment employing the fastest available PCs and 100Mbps Ethernet. We evaluate HORB, Java RMI, Voyager, two commercial CORBA IIOP implementations, and Distributed COM. For comparison a Java socket version and a C socket version are also evaluated. In order to represent the characteristics of the DOT, the performance of primitive object operations including remote object creation, remote method call, transferring arrays of objects and transferring large numerical data is measured and evaluated. No DOT won all benchmarks, but HORB showed very good performance for most benchmarks.

## 1 Introduction

Since the release of the network portable object-oriented language Java[1] by Sun in 1995, development of distributed object technologies (DOTs) for Java has been very active in academia and industry, and has resulted in the release of several DOT packages that are already being widely used today. These DOTs include HORB[2, 3, 4, 5] (the first publicly-available Java DOT) by one of this paper's authors, HIRANO Satoshi of Electrotechnical Laboratory (ETL) in 1995, some implementations of IIOP (Internet Inter-ORB Protocol)[6] of OMG CORBA, an implementation of Distributed COM[7] on Windows by Microsoft, Java RMI (Remote Method Invocation) API[8] by Sun in 1996, and the first agent ORB named Voyager[15] by ObjectSpace.

Because they make programming dramatically easier, system development using Java DOTs is becoming common, and can become the standard way of doing network computing in the near future. HORB, for example, has already been widely used in a variety of areas such as 3-tier business systems[9], mobile agent systems[10], distributed virtual reality[11], and so on. Currently, the authors of this paper are interested in using DOT for data acquisition in High Energy Physics experiments[12].

The common questions might be: How fast are those DOTs? Which DOT is fast for what? Are Java DOTs much slower than Java socket or C socket?

Through this paper, we hope to provide detailed performance characteristics of popular DOTs using microbenchmarks so that users can choose the best DOT for their purposes. We evaluated the DOTs in an environment that includes the fastest available PCs and 100Mbps Ethernet. We believe that such environments may become common for high-performance network computing in the next few years. The papers [4, 5] also evaluated some DOTs, but did not include Microsoft's DCOM and ObjectSpace's Voyager. Here, we include DCOM because it is the industry's other leading ORB and the foundation for Microsoft's Internet and component strategy. We also include Voyager, because it is currently one of the most popular ORBs. Unlike the earlier papers, this paper also evaluates the performance of numerical data transfer, which is very important for high-performance network computing.

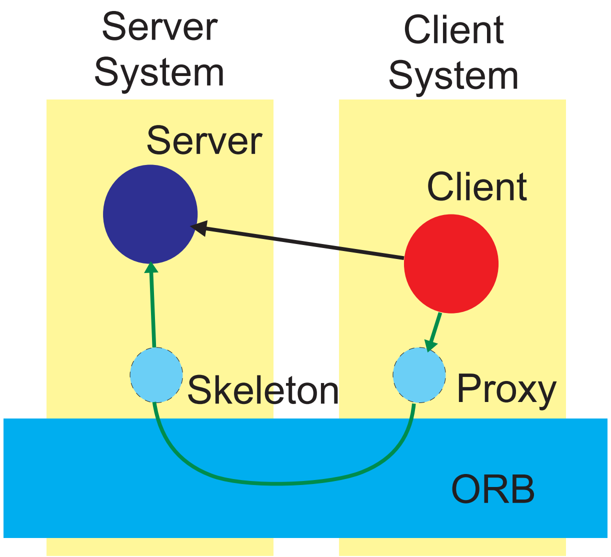


Figure 1: Realization of a distributed object using a Proxy

Since no paper has evaluated all these popular DOTs together so far, this paper may be very useful for application designers. For comparison purpose, we evaluate Java socket version and C socket version of the microbenchmark, too. To summarize briefly, the result of the evaluation showed that HORB was fastest in popular DOTs evaluated in this paper. It was as fast as C socket and Java socket.

After describing the overview of the popular DOTs for Java in Section 2, the performances of primitive object operations are evaluated in section 3. We evaluated the performance of the DOTs with respect to remote object creation/connection, remote method call, transferring arrays of objects and transferring large numerical data.

# 2 Overview of distributed object technologies

This section presents an overview of the popular DOTs which are evaluated in this paper.

## 2.1 HORB

HORB is a lightweight pure-Java ORB (Object Request Broker) that aims to seamlessly extend Java as a distributed object-oriented language[2, 3, 4, 5].

```
class Server {
  Video play(String title) {...}
}
```

List 1: Remote object of HORB

```
Server_Proxy s = new Server_Proxy("horb://www.horb.org/");
video = s.play("Apollo 13");
```

List 2: Remote object creation and remote method call

The process of 1) creating a remote object of the `Server` class of List 1 on a machine called `www.etl.go.jp`, and then 2) calling the `play()` method of the remote object, can be written as simply as in List 2. The minor difference (i.e., the magic words) between distributed and non-distributed programming is shown: to create a `Server` object on a remote host, one creates a `Server_Proxy` object instead, specifying the host name as a URL.

HORB holds the following characteristics:

- Programming is extremely simple and the Java language specification has not been altered. Distributed execution becomes possible by adding a touch of "magic" to a Java program as shown in list 2.
- No IDL (Interface Definition Language) is needed.

- Since the runtime is coded in JDK 1.0 Java, operation is ensured on all Java systems and Java-enabled browsers. The horbc compiler can generate external object serializer classes for JDK 1.0, (which does not have a built-in object serializer). Programmers may also choose to use the built-in object serializer of JDK 1.1, if they don't need compatibility with JDK 1.0.

The main distributed object functions are as follows: dynamic remote object creation, remote object connection, remote method call (synchronous or asynchronous), mobile objects and object transfer (by value or by reference), distributed garbage collection, access control lists, hook methods, distributed persistent objects, distributed object management and WWW integration.

For the realization of distributed objects, HORB uses the Proxy object method[14] (Fig. 1). Client objects access local Proxy objects in place of remote objects. A Proxy object provides equivalent method of a remote object to a client by using the remote object through ORB, which processes communication. Proxies and Skeletons are generated by the HORBC compiler. This architecture is essentially common to all DOTs.

### 2.2 Java RMI

RMI (Remote Method Invocation)[8], by Sun, is an ORB API for Java. A sample implementation of RMI is included in JDK 1.1 (Java Development Kit distributed by Sun). The current RMI defines very basic distributed object functions such as remote object connection, remote method call and object transfer.

In RMI, remote object definitions have to be coded as Java interfaces instead of IDL code. Implementing remote objects requires extending or implementing the `Remote` interface, the `UnicastRemoteObject` class, and the `Serializable` interface. Sun has announced that RMI would support CORBA in the future.

### 2.3 Voyager

Voyager[15], by ObjectSpace, is a very popular agent ORB for Java. It provides mobile agents, persistence, group communications, network class loading, directory service, etc, as well as a Java ORB. Group communication is supported by the Space architecture, which allows programmers to build distributed "rooms" in which objects reside. A programmer can send messages to objects in a Space as they were a single object.

Voyager is interoperable with CORBA systems. With Voyager, a programmer can use any existing Java class as a CORBA server, automatically creating CORBA IDL files from Java. Java program can also connect to existing CORBA servers.

The architecture of Voyager uses the reflection mechanisms of JDK 1.1 to remove the need for skeletons (server side stubs). This, however, may affect performance.

### 2.4 CORBA

CORBA[6] is a standard distributed object architecture specified by Object Management Group (OMG). Unlike the above ORBs, CORBA is not dependent of language or machine type. Remote object definitions are coded in the OMG IDL language. The CORBA core provides basic functions such as remote object connection and remote method call, while the CORBA services and CORBA facilities provide extended functions. The IIOP (Internet Inter- ORB Protocol)[6], an on-wire protocol, was defined in order to realize interoperability between vendors. The current IIOP (version 2.1) does not have the functionality for sending objects by value. Thus, very few CORBA products support passing objects by value; they extend IIOP independently. Therefore, this function is not interoperable among vendors. However, it will be specified by OMG soon.

The object models of OMG IDL and Java are different, since the object model of IDL was defined before the birth of Java. Thus, programmers have to deal with the impedance mismatch between them. One way to overcome this problem is to have a Java to IDL compiler, like VisiBroker does.

### 2.5 DCOM

Microsoft, with its ActiveX on Windows, has the largest share in the object market. ActiveX components are based on Microsoft's COM (Common Object Model). COM is a binary level interface between objects that is similar to the virtual method table of C++. DCOM (Distributed COM) has been developed in order to use COM objects remotely. With DCOM, current ActiveX components written for local execution can be used remotely without any modification.

Remote objects are defined in Microsoft ODL (Object Definition Language). The MIDL compiler and the JactiveX command generates Java stub files from the ODL definition. Like CORBA, there is an impedance mismatch between object models. Microsoft announced COM+ so that users could use remote objects without writing ODL[16].

# 3 Performance evaluation

The performance of each distributed object technology described in the previous section is shown in this section. We are developing a suite of microbenchmarks for distributed objects[1]. By using the benchmark program, the following primitive object operations, which we believe represent the characteristics of the DOTs well, were evaluated:

- Remote object creation and remote object connection
- Remote method call
- Object array transfer
- Large numerical data transfer

Although the benchmark reveals performance characteristics, we must admit that the analysis of these characteristics in this paper are unavoidably superficial. This is because it is difficult to analyze the reasons behind certain results without having the source code for the DOTs.

With the intent of evaluating the DOTs in what will most likely be the common environment for high-performance network computing in the next few years, we ran the benchmarks on an environment employing the fastest available PCs connected via 100Mbps Ethernet. The environment was as follows:

- CPU: 2 PentiumII 266MHz computers with 64MB memory each
- OS: Windows NT Workstation 4.0 SP3
- Network: 100Mbps and TCP/IP (100Base-TX switching hub, Half Duplex)
- Java VM: Microsoft's JIT and Sun's JIT
- Java Compiler: JDK1.1.4
- DOTs: HORB version 1.3.b2, Sun's JDK1.1.4 for RMI, Voyager 2.0 beta1, VisiGenic's VisiBroker for Java 3.0 (Java-implemented CORBA IIOP), IONA's OrbixWeb 2.0.1 (Java-implemented CORBA IIOP), and Microsoft SDK for Java 2.0 for DCOM.
- C socket and Java socket versions of the benchmark were also measured for comparison.

For the experiments, we ran a client program on a client computer and a server program on a separate server computer. Microsoft's Just-In-Time (JIT) compiler (the 'jview' command) was used on both computers to run Java Socket, HORB, RMI, Voyager and DCOM, since DCOM, RMI and Voyager did not work on Sun's JIT. We evaluated two CORBA IIOP implementations, VisiGenic's VisiBroker and IONA's OrbixWeb. Although there are many CORBA products, these two are among the most popular. Sun's 'java' command with Sun's JIT compiler option was used for VisiBroker and OrbixWeb, since the 'jview' command did not work for them. The Java source files for client and server programs were compiled with JDK's javac compiler with the optimization option '-O'. To avoid the influence of outside network activity, the evaluation environment was isolated from other networks.

## 3.1 Remote object connection and remote object creation

First, the performance of the remote object connection was evaluated. In this test, a client object connects to a remote object that was registered on a server a priori. We evaluated HORB, RMI, Voyager, VisiBroker and OrbixWeb. We couldn't find a way to connect to an existing DCOM object.

Figure 2 shows the execution time for one remote object connection. The vertical axis is the execution time in msec. VisiBroker was the fastest among them, while HORB, RMI and OrbixWeb have similar performance.

All DOTs except HORB and RMI do connection multiplexing[2]; that is, some connections to some remote objects are multiplexed onto one TCP/IP connection. Thus, they can save the time that would otherwise be required for establishing a TCP/IP connection. In contrast, HORB and RMI do not use connection multiplexing in order to keep the speed of accessing remote object as fast as possible, but have to pay the cost of creating a TCP/IP connection for connecting to each remote object. The results show that the overhead is negligible.

[1]The benchmark will be available at http://www.horb.org.
[2]RMI has connection multiplexing, but RMI does not use it.

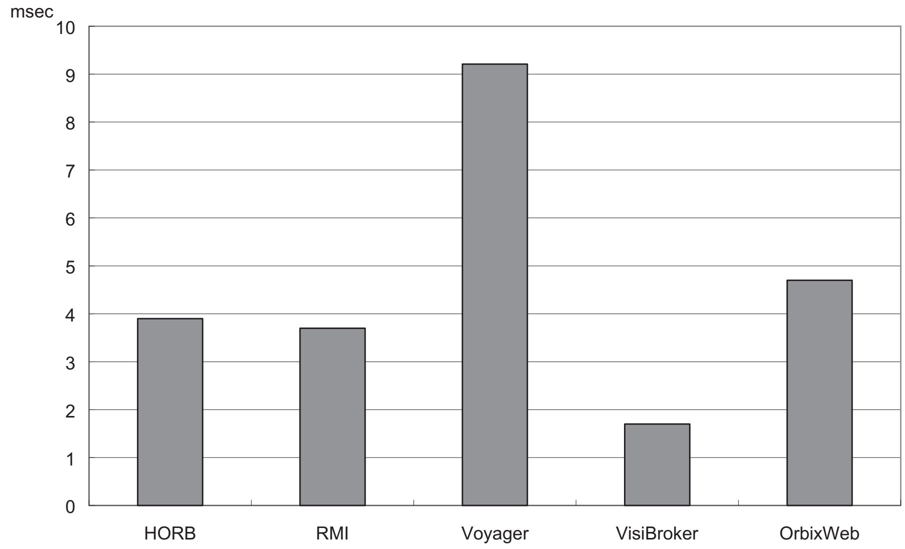


Figure 2: Remote object connection

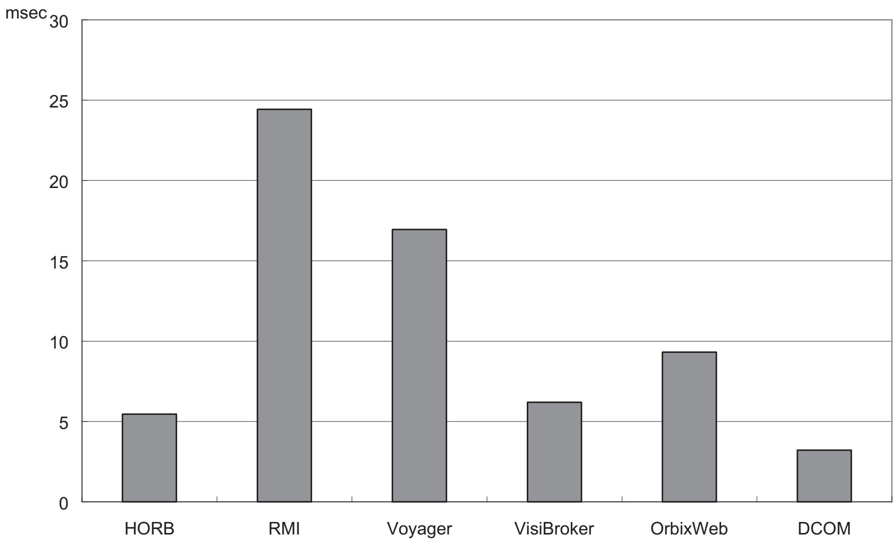


Figure 3: Remote object creation

Next, the performance of the remote object creation was evaluated. A client object dynamically creates a remote object on a server. While HORB, DCOM and Voyager are equipped with a function that dynamically creates remote objects, RMI and CORBA are not[3]. Therefore, we had to simulate the object creation for RMI and CORBA as following three steps; first, a client connects to an existing remote object, then the existing remote objects generates a new object and returns it as a remote object reference to the client.

The execution time of one remote object creation is shown in Figure 3. The vertical axis is the execution time in msec. DCOM was fastest because the server name was written in the registry of the client machine. Thus, it did not need to search the remote machine where the remote object existed on. In contrast, RMI, VisiBroker, and OrbixWeb were slower than DCOM because they had to search the remote machine where the remote object existed on using a registry process on the network. In case of HORB, although it also does not need to search the remote machine, it requires two round trips—one for connecting to the server, and one for creating a new remote object by calling a constructor of the remote object explicitly. The call to the constructor is optional. That is, if a client calls a remote method without calling a constructor, the default constructor is called implicitly to make a remote object. In that case, the execution time would be as fast as the time of DCOM, since HORB can omit one round trip.

[3]RMI in JDK 1.2 and Life cycle support in CORBA Services have a such feature.

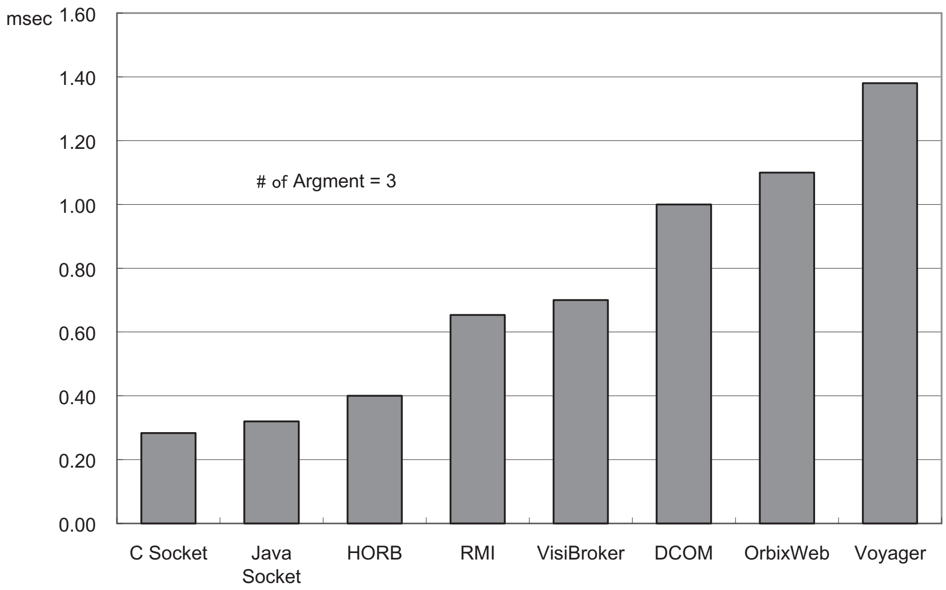


Figure 4: Remote method call

### 3.2 Remote method call

The performance of remote method call is vital for high-performance network computing, where response time is a major factor in performance. The following methods were used to evaluate the remote method call.

```
int methodA1(int a1);
int methodA2(int a1, int a2);
....
int methodA6(int a1, int a2, ..., int a6);
```

List 3: Remote method definition

A remote method receives a number of integers as arguments and returns an integer as the return value. In addition to ORBs, C socket and Java Socket versions were evaluated in order to show how ORBs are slower than socket implementations. The C Socket program is written in C, and uses the Windows socket library. The Java Socket simulates the remote method call using the Java socket library and the DataOutputStream class. Figure 4 shows the result when the number of arguments is three. The vertical axis is the execution time in msec.

As we expected, C Socket was the fastest. Following C Socket, the order was Java Socket, HORB, RMI, VisiBroker, DCOM, OrbixWeb and Voyager. . The HORB has mostly same performance as that of the Java Socket. Other ORBs were slower than HORB because they have to pay the cost for connection multiplexing. Voyager was slowest; more than three times slower than HORB. It has to pay the cost for reflection, since it does not have skeletons (server side stubs). Note that Voyager has more functionality than other DOTs. The execution time did not depend on the number of arguments. That means the cost for data copying is negligible in this test.

### 3.3 Object array transfer

Object array transfer was evaluated because it is used in many cases such as matrix calculations. A class containing an integer, shown in List 4, was used as the object to be transferred. The number of integers is only one because we want to measure the performance of object transfer, not integer transfer. The remote method can accept any length of the object array.

```
class Data { int a; }
class Server {
   void method(Data[] array) {}
}
```

List 4: Classes for object transfer

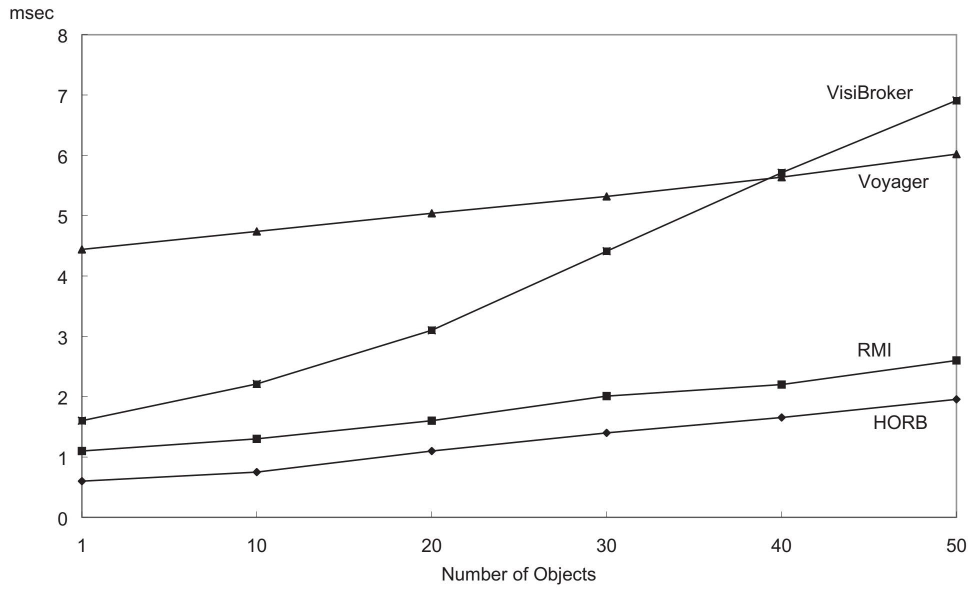


Figure 5: Object array transfer

Only HORB, RMI, Voyager and VisiBroker were evaluated, since DCOM and OrbixWeb do not have the functionality of object transfer by value. VisiBroker has its own function for object transfer, because the current IIOP does not define object transfer. VisiBroker's 'java2iiop' command generates codes for object transfer. We could not express the object transfer in an IDL file. Therefore, this functionality does not have compatibility with other CORBA products. Note that DCOM and OrbixWeb can send data as `struct`, but not as objects. Object transfer is necessary when an object has some methods or a programmer wants to use object polymorphism.

The results of object transfer is shown in Figure 5. The horizontal axis indicates the number of objects within an array to be transferred to the remote method. The vertical axis is the execution time in msec. This figure shows that HORB was the fastest. When HORB transfers an object, a serializer object (external serializer) that serializes the object to a stream form is created and used. Although the external serializer has a drawback, i.e., it cannot transfer private variables, it is highly optimized[5] as follows:

1. To reduce the cost of creating the serializer object, a serializer cache is introduced. When objects of the same class are continuously transferred, the same serializer object is used rather than creating a new serializer object each time.

2. When the class name of an object has to be transferred and the same class has just been sent, a tag indicating "the class of this object is the same as before" is sent. This reduces the number of class name transfers.

The other DOTs use the built-in object serializer defined in JDK 1.1, i.e., java.io.ObjectOutputStream. The reason of the difference in performance is not clear.

### 3.4 Numerical data transfer

High-performance network computing applications such as data acquisition systems for high energy physics experiments require high bandwidth of numerical data transfer. In such systems, a large amount of data is manipulated over the network. More concretely, large arrays of byte, int or double are transferred. Therefore, we are highly interested in whether or not the DOTs have enough performance for numerical data transfer, since so far, TCP/IP socket programming has been used for transferring large numerical data for speed.

```
class Server {
   void methodB(byte[]   array) {}
   void methodI(int[]    array) {}
   void methodD(double[] array) {}
}
```

List 5: Methods for numerical data transfer

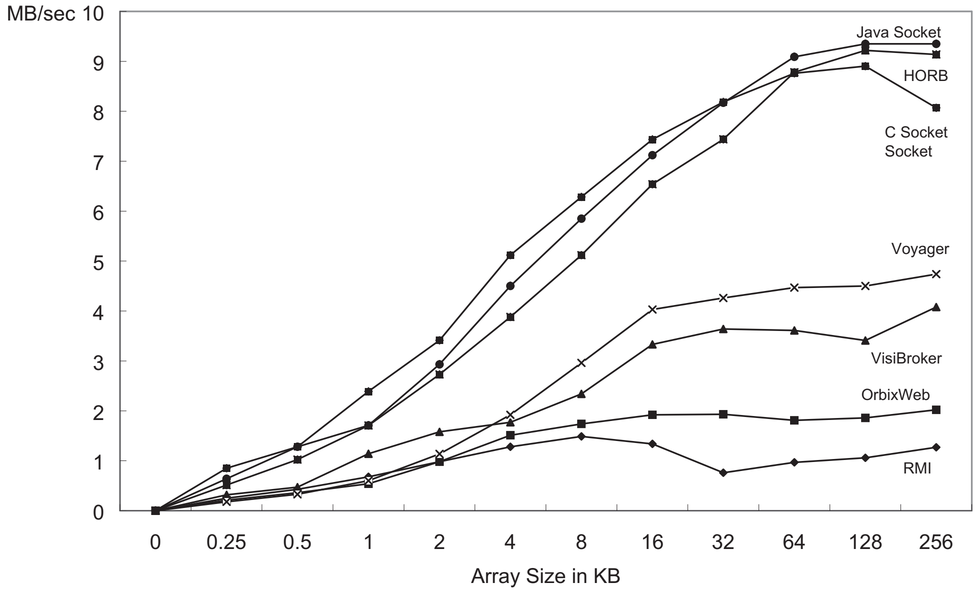


Figure 6: byte array transfer

For the evaluation of numerical data transfer, we used three methods that take a byte array, an int array, and a double array, as arguments, as shown in List 5. The size of array is undefined at the compile time.

The performance not only of HORB, RMI, and CORBAs, but also of a C socket version and a Java socket version were evaluated. The Java Socket version used the DataOutputStream class with the BufferedOutputStream class to act like the DOTs version . The buffer size was 32KB. The C socket version is equivalent to the Java socket version, but is written in C. It does byte reordering for the network byte order. We couldn't measure the performance of DCOM because of a flaw in the tools and Microsoft's JavaVM. We got some errors during compilation and execution.

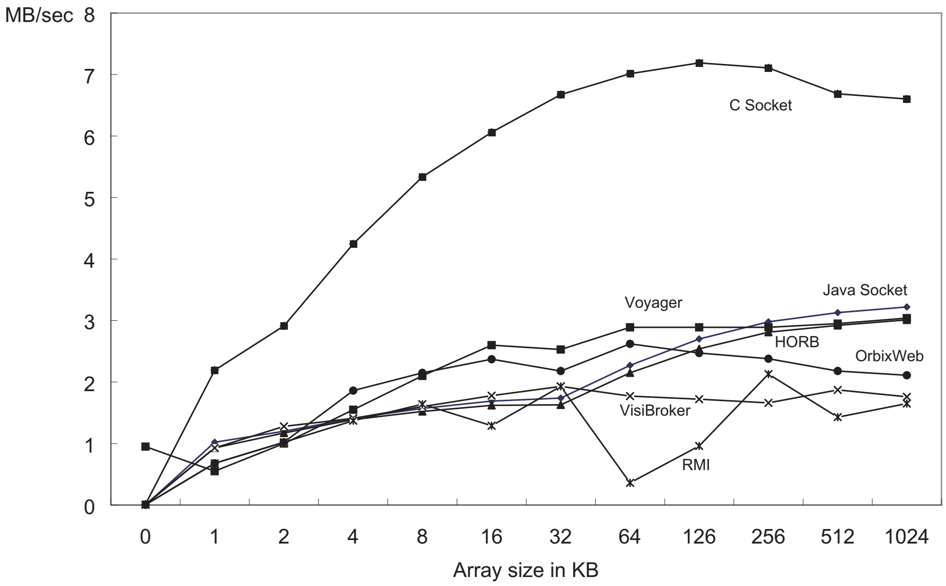


Figure 7: int array transfer

Figures 6, 7, and 8 show the results of byte array, int array, and double array transfers respectively. The horizontal axis indicates the size of an array to be transferred to a remote method in KB. The vertical axis is the bandwidth of transfer in MB/sec.

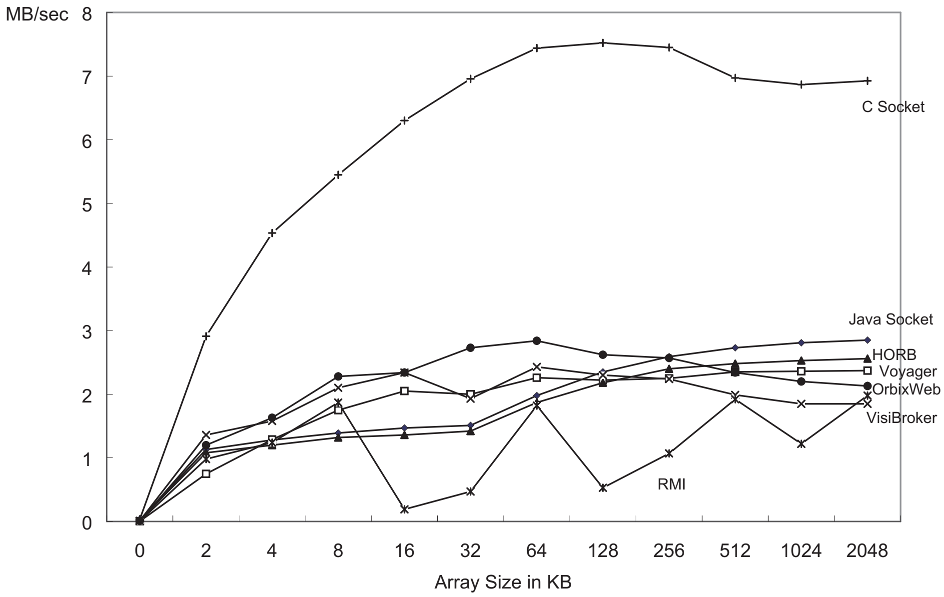


Figure 8: double array transfer

For byte array transfer, Java Socket and HORB showed very high data transfer rate and the performance of HORB was similar to that of Java Socket. These saturated at near 9.4MB/sec. This result is very encouraging. The data transfer rate of C Socket is similar to these two. Thus, we can say that the data transfer rate of HORB is near the maximum data transfer rate of the 100Base-TX network. That is, it is network bound for byte array transfers. On the other hand, the data transfer rates of other DOTs were low. Note especially that the peak data transfer rate of RMI was only 1.3MB/sec.

For int and double array transfer, the data transfer rates of Java Programs were far worse than that of C Socket. Furthermore, the performances of Java programs except RMI were not varying, especially for small array size. We can say that these operations were CPU bound and the cost of byte reordering and data copy was very expensive compared to that of C. All DOTs except HORB use the system object serializer, ObjectOutputStream and ObjectInputStream. ObjectOutputStream does its own buffering. The buffer size is 255 bytes for Microsoft's JIT and 1024 for Sun's JIT. In the case of RMI, the buffer size is only 255, since it runs only on Microsoft's JIT (RMI did not run on Sun's JIT). Furthermore, the data is buffered doubly in both ObjectOutputStream and BufferedOutputStream. These are the reason why RMI was slow and had large fluctuations. We expect RMI to run much faster if it can run on Sun's JIT.

# 4 Discussion and Conclusion

This paper evaluated the performances of popular distributed object technologies, including HORB, RMI, Voyager, two popular CORBA IIOP implementations and DCOM. In order to present the characteristics of DOTs, we evaluated the performance of primitive object operations such as object connection and creation, remote method call, object array transfer, and data transfer of large numerical data. The performance characteristics were varied very much. No DOT won all benchmarks. HORB showed good performance for most object operations, especially for remote method call. All DOTs have to improve their performance for the int array transfer and the double array transfer.

We would like to point out three important remarks obtained from the above evaluation:

First, the simplicity of buffer management is very important for high-performance network computing. All DOTs except HORB have the feature of connection multiplexing. Connection multiplexing requires the DOTs to pay the cost of buffer management and data copying. In the remote method call test, the cost of the buffer management degraded the speed of all DOTs except HORB. The cost of the data copying was invisible in the test, since the number of arguments did not affect the execution time. On the other hand, in the large numerical data transfer test, the cost for data copying degraded the bandwidth largely for the DOTs (Figure 6).

Second, from the user's viewpoint, the fact that a DOT is as fast as sockets, means we can use the DOT for data paths as well as for control paths in applications. For example, this fact changes the design of the data path in our data acquisition system. HORB can be used as the data path instead of socket programming.

Third, the current JIT technology has not matured enough to catch up with the speed of 100Base-T network. A faster network such as Gigabit Ethernet would not improve the performance largely. We may have to use a better JIT or a native code compiler when using faster networks.

We would also like to mention some observations we have made about the ease of writing programs using DOTs. Although it is difficult to express ease-of-use as a value, we can say that there is a large variance in ease-of-use among the DOTs. From our experience, HORB and Voyager are the easiest to use, RMI is next, and DCOM is the most difficult to use and has many limitations.

**Acknowledgments** The authors would like to thank the HORB Open consortium and the HORB mailing lists participants, especially Dr. Hiromitsu Takagi for his analysis of ObjectStream and Mr. Luis Sarmenta for helping with the grammar. HORB Open was funded by ETL.

HORB is available freely with the source code from http://ring.etl.go.jp/openlab/horb/. The development of HORB is open. The use of HORB for research and development is strongly welcomed.

# References

[1] Gosling, j., Joy, B. Steele, G., The Java language specification, Addison-Wesley, 1996

[2] HIRANO, S.: HORB: a distributed object oriented language for worldwide programming, WOOC'96, 1996 (In Japanese)

[3] HIRANO, S.: Constructing a portable ORB for distributed execution of Java, SwoPP'96 IPSJ Technical Report 96-OS-73, pp55-60, 1996 (In Japanese)

[4] HIRANO, S., HORB: Distributed Execution of Java Programs, Worldwide Computing and Its Applications, Springer Lecture Notes in Computer Science 1274 pp.29-42, 1997

[5] HIRANO, S., Preliminary Performance Evaluation of a Distributed Java, HORB, Object Based Parallel and Distributed Computing, 1997.

[6] Object Management Group, The Common Object Request Broker: Architecture and Specification 2.1, 1997

[7] Brown, N., Kindel, C., Distributed Component Object Model Protocol – DCOM/1.0, Internet Draft, 1996

[8] Sun Microsystems, Remote Method Invocation Specification, 1997

[9] Duan, N., Distributed Database Access in a Corporate Environment Using Java, 5th Int. WWW Conference, 1996

[10] Yamanaka, A., Nakajima, S., Tomono, M., Tonouchi, T., A HORB-Based Network Management System, ICODP'97, 1997

[11] Hagiwara, Y., Watanabe, S., Maki, Y., Distributed scene simulation using VRML and Java, 22th Civil Engineering System Symposium, pp.167-170, 1997 (In Japanese)

[12] Y.Yasu, H.Fujii, E.Inoue, H.Kodama, Y.Sakamoto : Prototype Performance of Distributed DAQ using HORB based on Java., Real Time 97, the 10th IEEE Real Time Conference 97, BEAUNE, FRANCE, 22-26 September 1997

[13] Gamma, E., Helm, R., Johnson, R., Vlissides, J., Design Patterns, Addison-Wesley, 1995

[14] Shapiro, M., Structure and Encapsulation in distributed Systems: The Proxy Principle, ICDCS, pp.198-205, 1986.

[15] ObjectSpace, VOYAGER Core Technology User Guide, 1997

[16] Mary Kirtland, Object-Oriented Software Development Made Simple with COM+ Runtime Services, Microsoft Systems Journal, Vol.12, No.11, 1997