# Performance Studies of Remote Method Invocation in Java

George Koutsogiannakis, Marios Savva
Dept. of Computer Science
Illinois Institute of Technology
Chicago, IL, 60616-3793, USA
{koutsogiannakis, msavva}@iit.edu

J. Morris Chang
Dept. of Electrical and Computer Engineering
Iowa State University
Ames, Iowa 5001, USA
morris@iastate.edu

*Abstract*

*In 1999 Sun Microsystems and IBM introduced a new version of Java's Remote Method Invocation (RMI), called Remote Method Invocation over Internet Inter-Object Request Broker (ORB) protocol (RMI over IIOP), with the release of JDK1.3. The new RMI API uses a different transport protocol than the original RMI, that is compliant to Common Object Request Broker Architecture's (CORBA) Internet Inter-ORB (IIOP) specification. The original RMI uses the Java Remote Method Protocol (JRMP) as the transport protocol. This paper illustrates that IIOP is less efficient in the transmission of data than JRMP in spite of the fact that it uses less TCP packets. The goal is to allow developers to decide when to use JRMP versus RMI over IIOP when Java to Java object communications are involved and performance cost is of issue.*

keywords: Java Performance, Remote Method Invocation Performance, Internet Interoperability, Internet Inter_ORB Protocol, Distributed Garbage Collection

## 1. Introduction

In 1997 Remote Method Invocation (RMI) was introduced in JDK1.1.Remote Method Invocation (RMI) is a Java based technology that allows a developer to create distributed object applications using the Java Remote Method Protocol (JRMP). In 1999 Sun and IBM cooperated in introducing a new RMI transport protocol, called Remote Method Invocation over Internet Inter-ORB (Object Request Broker) Protocol (RMI over IIOP). This new protocol is compliant to Common Object Request Broker Architecture's (CORBA) IIOP specification. The scope of this paper is to evaluate the performance of the two RMI transport protocols based on empirical data obtained with a RMI benchmark software module that we developed. Empirical data was obtained by using multiple invocations of a varying in size array of integers and a varying in size string of characters. The benchmark software allowed the capture of time at the application layer. A network management software module called Ethereal, was used in order to analyze the network layer communications (TCP packets). The time measured by the benchmark included the data carrying RMI communications transactions for each group of invocations but not any RMI communications set up time.

RMI allows the transfer of objects as well as primitive types by value or by reference. The transferring of data is accomplished through the *serialization* utility of Java. RMI also provides a name registry database of remote objects which runs on the server side as well as a mechanism to garbage collect distributed objects. RMI is restricted to Java to Java object communications and its JRMP protocol is not compliant to CORBA specification. RMI's performance is degraded primarily because of the computational cost of the *serialization* procedure and the extra communications required by the *Distributed Garbage Collection (DGC)* mechanism.

When using RMI over IIOP to produce Java technology based distributed applications, there is no need for learning an Interface Definition Language (IDL). A developer can decide on the proper RMI protocol depending on the applications and the performance requirements.Comparative performance measurements of JRMP RMI versus RMI over IIOP are not available when utilization of multiple invocations and variable amounts of data of different data types is made.

## 2. Motivation

The need to know the cost of using RMI as a function of the number of invocations and the amount of data transmitted for various data types, is important to the developer. Developers need to make a decision based on the performance criteria provided in the application's specification.The availability of benchmark software was researched and few were found for specific applications pertaining to the JRMP protocol. The results listed in the published papers were not specific enough to allow usage by a developer to predict performance based on the number of invocations and the amount of data transmitted. No comparison data of the performance of the two protocols was found as a function of the number of invocations and the amount of data transmitted.

The decision to benchmark the two types of RMI protocols was also based on the non availability of data that will explain the communication transactions and the cost involved with each transaction.

### 2.1 Previous Work.

Work has been performed by various researchers in terms of measuring (benchmarking) and improving the performance of RMI. M. Phillipsen et all [1] produced an improved *serialization* facility for RMI and released it as a new RMI framework under the name *KARMI*. Their serialization simplifies the data type information carried by the stream.The group also developed their own benchmark software and measured performance of KARMI claiming a 90% improvement over the standard RMI. Our data contradicts one of their conclusions that the overhead cost is constant for large array objects. V. Krishnaswamy et all [2] revised the transport layer of RMI in order to utilize UDP message delivery instead of TCP and introduced a multicast protocol which is used to maintain consistency of cached objects. They also revised the reference layer of RMI in order to allow caching of objects at a client node. Their experimental data showed mixed results with marginal improvements in the case of object caching and with negative results in the case of using UDP instead of TCP protocols.

K. Kono et all [3] improved the performance of RMI by introducing a variant of Sun's serialization which dynamically converts arguments into in-memory representations valid at the client's platform thus claiming an improvement of 1.9 to 3.0 times faster than Sun's RMI. The improvements introduced were based on a reduction of memory accesses required to copy and convert object structures. S. Ahuja and R. Quintao [4] performed benchmark measurements of the cost of RMI (using JRMP) versus the cost of Java Sockets and found insignificant differences. Their benchmark was based on read and write operations into a file. The amount of data per TCP packet that RMI uses was not compared to an equivalent socket implementation. S. Campadello et all [10] did an analysis of the performance cost of RMI and proposed a solution in improving the performance in wireless applications of RMI. Their solution is based in mediators and a lookup of cached objects first before a request is sent to a server. Their benchmark was based on KARMI [1] and found a four times improvement over normal RMI.

## 3. RMI/JRMP Architecture and protocols

A RMI application consists of a client program that will request the services of a remote object from the server program. The basic components consist of an interface, a *rmi server* program that represents the implementation of the interface, the *rmi registry*, the client program and a *stub*. The client will first obtain a reference to the remote object from the *registry*, which is listening at the default port 1099, by performing a lookup. Once a reference is obtained communications with the server are done via the stub.The *rmi server* is responsible for implementing the operations defined by an *interface*. These are the operations that the remotely invokable object can use. The *rmi server* binds or rebinds the remote object to the *rmi registry*. The stub is created during the compilation of the *rmi server* program and it is transferred to the client side [5].

A better understanding of the communications involved can be obtained by examining RMI's layered architecture and comparing it to the OSI model (Fig.1). The application layer is the same as the application layer of the OSI model and contains the respective application programs. The stub/skeleton layer is responsible for managing the remote object interface between the client and the server. When the client issues a remote method invocation the stub in essence opens a socket to the server on the port specified in the stub itself. The stub then sends the RMI header information, that is the hostname of the server and the port number that the server is listening for remote calls on the object. The stub marshals (serializes) and unmarshals the arguments sent over the connection. The Remote Reference Layer is responsible for interpreting and managing references to the remote service objects. It connects using a unicast link (point to point) connection. It also supports the activation of dormant services (Java 2 SDK), threading, and garbage collection. The Transport Layer is using by default the TCP protocol. This is where the JRMP protocol is located. On top of the TCP/IP, RMI implements the Java Remote Method Protocol (JRMP) specified by Sun. The JRMP makes use of two other protocols: Java Object Serialization and HTTP.

## 4. RMI over IIOP Architecture and protocols

RMI over IIOP provides the functionality specified by CORBA. Developers using Java technology do not have to write Interface Definition Language (IDL) modules. Objects can be serialized and passed by value between applications. RMI over IIOP is based on two specifications of the Object Management Group: *Java Language Mapping to OMG IDL Specification* [7] and CORBA/IIOP 2.3.1 Specification [8].Compilation of the server implementation program using the *iiop* option of the RMI compiler, generates a *tie* module. The *tie* module is used by the server to communicate with the client via the IIOP protocol. The function of the IIOP protocol is to translate any Object Request Broker (ORB) messages to TCP/IP.

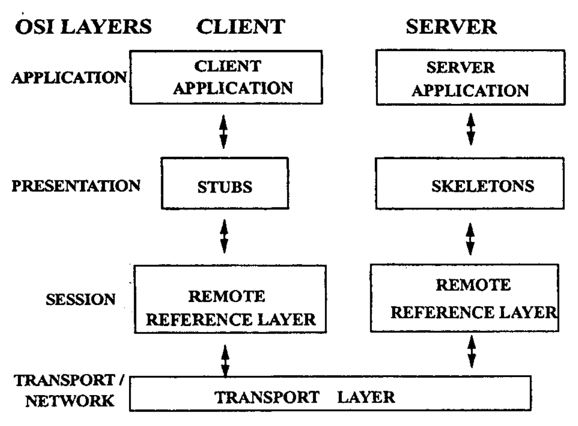

FIGURE 1. RMI Architecture

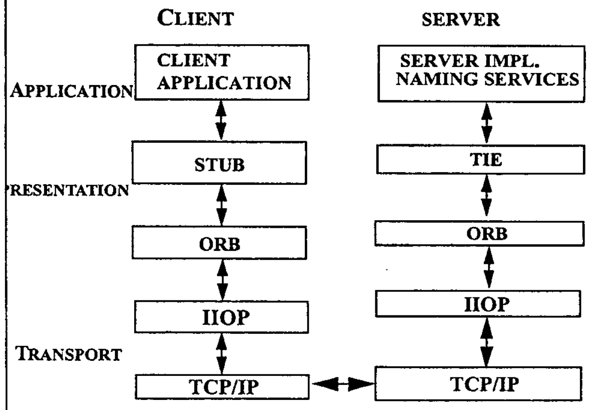

FIGURE 2. IIOP PROTOCOL ARCHITECTURE

The differences between the two protocols (JRMP and IIOP) start at the top layer (Fig. 2). The CORBA COS (Common Object Services) Naming Service has to be used. A default name server is provided with the JDK under the name *tnameserv.* This nameserver listens at port 900. In order for the COS Naming Service to be used, the Object Request Broker (ORB) must know the port of a host running a naming service.Java edition 1.2 and higher includes an ORB written in Java.The purpose of ORB is to make it possible for clients to connect with objects servicing requests on the server side.An application gains access to the CORBA environment by initializing itself into an ORB. Only the ORB with which the remote object is published understands the actual object and its references. The ORB can invoke operations and calls to objects located within other ORBs.This gives rise to a requirement for an Inter-ORB bridge and as a result the IIOP requirement was produced.

## 5. Benchmark description

The goal of the benchmark software was to accumulate data based on two variables; invocations and amount of data bytes transmitted. This is a synthetic approach that was chosen over a real world application benchmark approach because it would allow the generation of a more generalized performance prediction algorithm. The benchmark allows the user to invoke an array of integers or a string of characters.These two data types appear quite often in an application and therefore they could be good predictors of performance.

The client software provides an algorithm where the data carried by the object invoked grows by an increment defined by the user, as the software loops through a number of remote method invocations defined by the user. The number of loops grows by a user defined increment up to a maximum number.

### 5.1 Algorithm description

The algorithm can be formulated and the number of invocations and the amount of data can be calculated before a benchmark execution, using a derived formula. The number of characters or integers transmitted after all the loop iterations are completed can be mathematically represented. There are two loops involved. The inner loop starts with a specified number of invocations and repeats them while increasing the number of data transmitted.The outer loop repeats the inner loop for an increased number of invocation loops.Let *n* be the number of times that the string size or the array of integers will be increased during one set of loop invocations (inner loop).Also let l represent the number of times loop invocations are increased (outer loop). The total number of data types transmitted is given by:

$$lx((l+1)/2)xLIxDIxnx(n+1)/2)$$

where LI is the user defined increment by which the loops will be increased (outer loop) and DI is the increment by which the data size will be incremented during each set of specific number of loop invocations (inner loop). The total number of data is multiplied either by 4 (for integers) or by 2 (for characters) in order to obtain the number of data bytes. The time taken to invoke data during one set of outer loop iterations is recorded by the software. There is one more algorithm in the software that allows the recording of the time taken to do the very first invocation of data. This allows us to extrapolate any set up time needed by the Java Remote Method protocol.

Another important parameter besides the TotalDataTypes parameter is the total number of invocations made. *The total number of invocations is:*

$$(l \times (l+1)/2) \times LI \times n)$$

. The performance of the RMI application is strongly dependent on the two parameters; *TotalData* and *TotalInvocations.*

## 6. Performance measurements

The performance of the two RMI protocols was accomplished using a Windows NT (NT 4.0) environment on the client side and a Linux (Red Hat 6.2) on the server side.The client was a 186 MHz Pentium Pro machine with 128 MB of RAM whereas the server was a 650 MHz Pentium 3 machine with 128 MB of RAM.The two machines were connected via 100 B Ethernet.A series of benchmark executions were performed. Each time the number of invocations was held constant and the amount of data transacted was varied.The benchmark software recorded the round trip time for each invocation.Information on the TCP packets for each benchmark execution was also saved using the network management software (Ethereal).

Graphs of performance cost (transmission time) versus number of bytes transmitted were plotted for specific number of invocations. One can view the excel tables with all the data information by going to the web site dedicated to this research: *http://213.47.150.110/notes/* .The graphs are shown in Fig. 3 through Fig. 14. The lines in the chart represent a linear regression of the data points involved. Each point on a chart represents the total time for all loop iterations from one benchmark run. The number of TCP packets versus amount of data for a constant number of invocations was also plotted. Data was obtained for 20, 160 and 560 constant invocations while the size of both the Array of integers and the string data types were varied.

## 7. Data Analysis

The charts reveal that the IIOP protocol consistently takes longer to transmit the same number of bytes with same number of invocations for both integer and character types (see Fig. 3, 4, 6, 8, 9, and 13).The charts also show that the IIOP protocol uses on the average less TCP packets to transmit the same amount of data with the same number of invocations (see Fig.5, 7, 10, 11,12 and 14). As an example, the file pertaining to 20 invocations with 24KB of array integer data was analyzed.

In the case of IIOP the total number of TCP packets was 160. Communication started between the client and the Name Server at a cost of 6 packets. The naming context and the proper protocol identification were established during this communication.In the next level of communication the client asks the ORB to resolve the name of the server. The name server gets resolved and a port is established for remote invocations. Another 6 packets were expended between the client and the server in identification information.Invocations from the client begin with packet number 22 and continued until all data had been transferred.The actual data transferred, including the invocations, took 137 packets. During each invocation the packets transmitted were as follows:

client --> server Invocation packet.
server-->client responds with the identification numbers of the objects involved.
server-->client data packets.

The number of packets carrying data was 97 out of the 137 packets transmitted between the client an the server during invocations. The ratio of data transferring packets over the total number of packets transmitted was 71%. An analysis of the times devoted towards the client/server communications for data exchange versus all other overhead communications shows that the ratio of data communications versus total time was 331ms/ 1094 ms or 30% (for the 137 packets). This demonstrates that a substantial amount of time is devoted to set up communications. A similar analysis of the JRMP data file revealed that 250 TCP packets were used. Communications start between the client an the registry. The client obtains the object reference and the port number the server is listening to. Additional set up communications take place between the server and the client where the IP addresses are confirmed, the stub ID and Object ID are passed over and DGC information is exhanged. The client issues a *dirty call* in packet 20 and requests a *lease* period.The server responds with the lease and the VMID (Virtual Machine ID of the client) confirmation. Data exchange communications begin with packet 35. A minimum of 5 packets of overhead data is consumed during each invocation, before data is transmitted. The actual data carrying packets in this example were 101 out of 216 or 47%. The time consumed for set up at the beginning up to the time invocations are ready to start was 1034 ms out of the 1194 ms for the entire file execution. The data transfer took only 160 ms as it is confirmed by the benchmark software (14%) Therefore 86% of the time was spent in setting up the transactions before any data is transmitted.The DGC communications take substantial amount of that time, as it is confirmed by S. Campadello et all at the University of Helsinski, Finland [10]. The overall time including set up time, is almost equal to the time it took by the IIOP protocol for the same number of invocations and IIOP data (1041 ms versus 1194 ms). The data also shows that there is

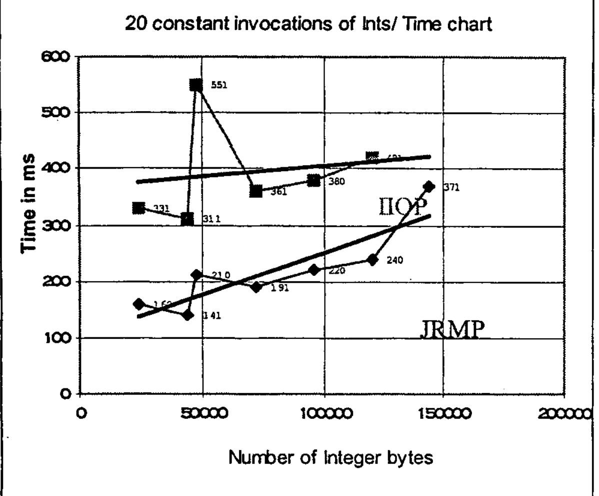

**FIGURE 3.** INTEGER COST OF 20 INVOCATIONS.

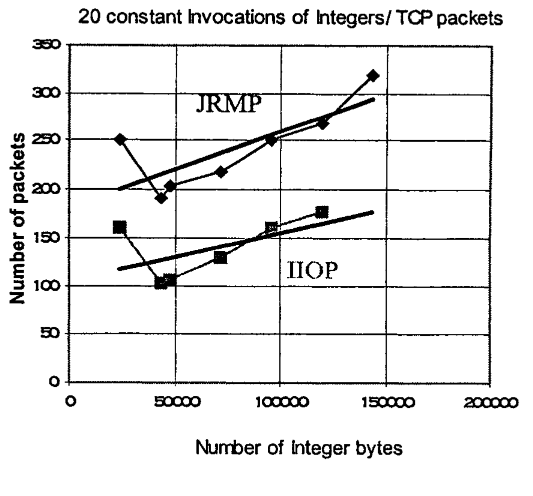

**FIGURE 5.** INTEGER COST OF 20 INVOCATIONS (TCP).

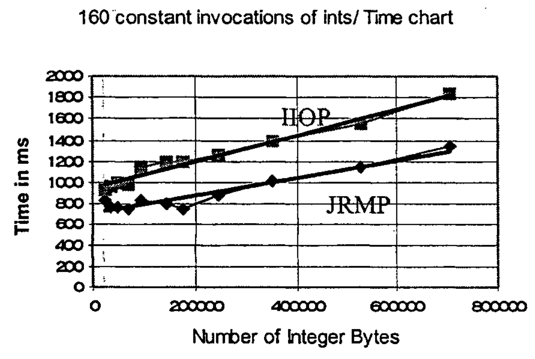

**FIGURE 4.** INTEGER COST OF 160 INVOCATIONS.

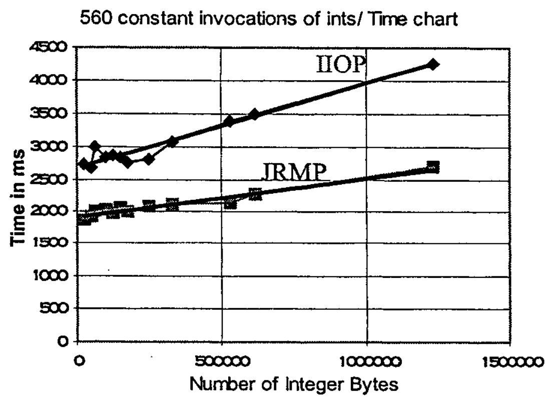

**FIGURE 6.** INTEGER COST OF 560 INVOCATIONS.

some difference in the performance when integers are involved versus characters, based on the same number of bytes and invocations. Analysis of the data in the packets reveals that the maximum amount of data carried by RMI (JRMP) TCP packet during invocation is 1078 bytes whereas an IIOP TCP packet carries approximately a maximum of 1800 bytes.

This explains the fact that there are less TCP packets devoted to data transfer during each invocation in IIOP. Unfortunately, IIOP consumes more time for data exchange because of the ORB layer and its serialization procedure. IIOP goes through additional data type conversions from IDL to Java and vice versa even when Java remote objects are involved and conversion would not seem necessary. Serialization of Arrays takes additional serialization data in IIOP and as a result additional time. Figures 15 and 16 summarize the time differences and set up cost respectively. The numbers in parenthesis in Figure 15 are the difference in time when equal number of data bytes are considered for both integers and characters. The data indicates that the relative time difference between IIOP and JRMP grows in an non-linear fashion as the number of invocations and the amount of data grows. It also seems that IIOP becomes more efficient in transmitting characters than transmitting an equivalent number of array of integers bytes. Overall IIOP, nevertheless, is still less efficient than JRMP in the transmission of characters.The set up times (time before invocations begin) between the two protocols is also different with IIOP requiring less time.

## 7.1 Conclusions

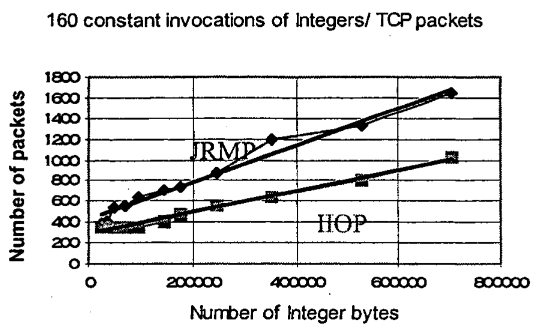

**FIGURE 7.** INTEGER COST OF 160 INVOCATIONS (TCP).

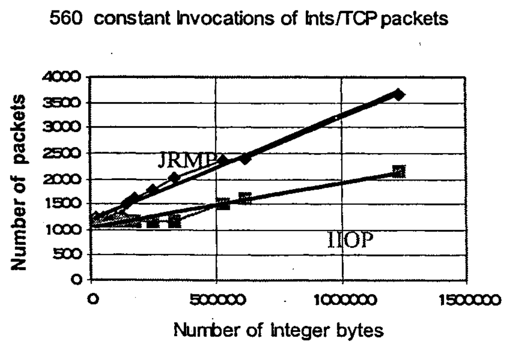

**FIGURE 10.** INTEGER COST OF 560 INVOCATIONS (TCP).

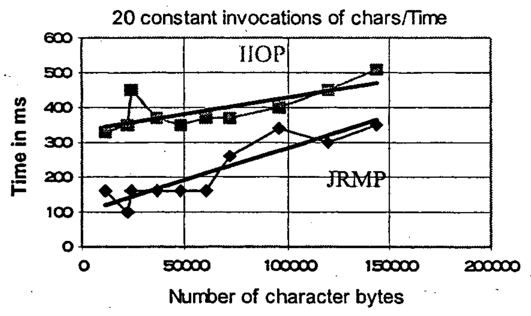

**FIGURE 8.** CHARS COST OF 20 INVOCATIONS.

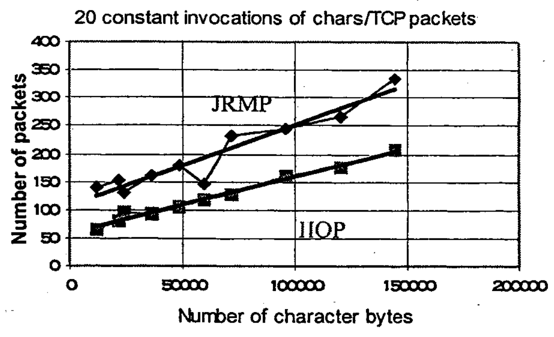

**FIGURE 11.** INTEGER COST OF 20 INVOCATIONS (TCP).

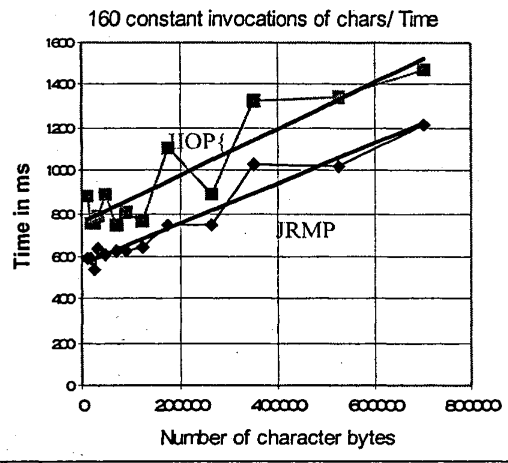

**FIGURE 9.** CHARS COST OF 160 INVOCATIONS.

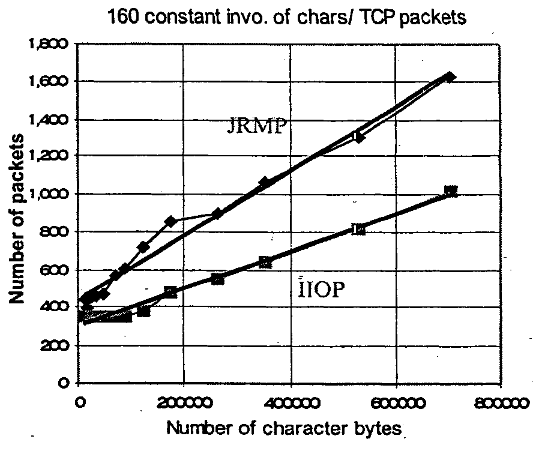

**FIGURE 12.** INTEGER COST OF 160 INVOCATIONS (TCP).

The empirical data collected indicates that data delivery is accomplished more efficiently in time by the JRMP protocol. During the delivery of data, however, the JRMP protocol uses more packets that the IIOP protocol because of the additional transactions needed during each invocation.It seems that a lot of the same information is being exchanged during each invocation in the JRMP protocol. The set up transactions also take a lot more time in the JRMP than in the IIOP protocol. This is due to the fact that the IIOP protocol does not have to deal with a Distributed Garbage Collector (DGC). Although RMI uses the algorithm inherited

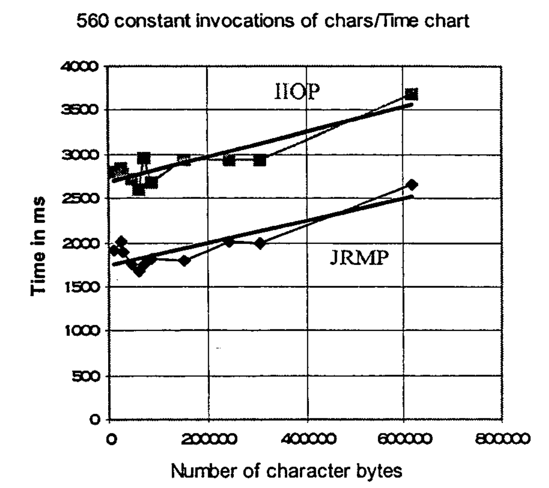

**FIGURE 13.** Chars cost of 560 invocations.

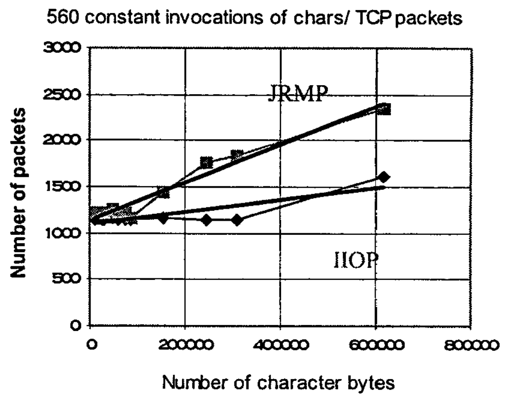

**FIGURE 14.** integer cost of 560 invocations (TCP).

from *Modula 3,* another, more efficient, algorithm of the many described in S. E. Abdullahi and G. A. Ringwood' s paper on the categorization of Distributed Garbage Collection algorithms [11] could be chosen.

The difference in data delivery times between IIOP and JRMP widens as the number of invocations increases and the number of data increases (see Fig. 15). The cost of 20 invocations for an array of integers shows that JRMP is on the average 176 ms faster than IIOP. When 560 invocations are involved, JRMP is almost 1 second faster than IIOP in the delivery of data. Since set up time transactions stay the same for both protocols regardless of the number of invocations and the amount of data to be transferred, the conclusion is that for large amounts of data using the JRMP protocol would be more cost effective. The overall time for large amounts of data is lower than the set up time for benchmark files run under the JRMP protocol. For smaller amounts of data the IIOP protocol will be more effective since the set up time is significantly less that of JRMP. Substantial overhead is transmitted during each invocation which is repetitive. One could conclude that a definite improvement would be to reduce the amount of overhead by not transmitting the repetitive overhead information. The client will have to notify the server that a considerable amount of invocations are expected and that the overhead information could be sent only every so many invocations. That will reduce the number of packets and increase efficiency.

In conclusion this paper showed that JRMP is more efficient in the transmission of data than IIOP in spite of the fact that more TCP packets are transmitted over the network when JRMP is used.

The recomendation therefore to developers is to use IIOP for small amounts of data where the time for the transmission of data is small by comparison to the set up time for JRMP. For larger amounts of data JRMP should be preferred. This assumes of course that performance is the only criterion. The network speed should also be considered carefully since more datya packets will be transmitted if JRMP is chosen.

## 8. Future work

Future work is centered around the optimization of the performance of both types of RMI. The serialization mechanism for both protocols and the DGC for JRMP are areas that can afford improvement.The addition of an efficient DGC algorithm for IIOP will be examined. Additional investigation is needed that will compare performance of homogeneous systems versus heterogeneous systems (since serialization routines are platform depended). The development of a predictive algorithm for the performance of RMI will be continued.

## 9. References

[1] M. Phillipsen, B. Haumacher, More Efficient Object Serialization. International Workshop on Java for Parallel and Distributed Computing, San Juan, Puerto Rico, April 12, 1999.

[2] V. Krishnaswamy, E. Bommaiah, D. Walter, G. Riley, S. Bhola, B. Topal, M. Ahamad, Efficient Implementations of Java remote Method Invocation (RMI). Proc. of the 4th USENIX Conference on Object-Oriented Technologies and Systems (COOTS'98), 1998.

| | INTEGERS | | | CHARACTERS | | |
|---|---|---|---|---|---|---|
| INVOCATIONS | 20 | 160 | 560 | 20 | 160 | 560 |
| NUMBER OF DATA TYPE IN 1000 | 6 - 30 | 6-176 | 6 - 308 | 6 - 30 | 6-176 | 6 - 308 |
| DATA RANGE IN KB (EQUALIZATION OF BYTES) | 24 - 120 (24-120) | 24- 704 (204-704) | 24 - 1236 (24 - 616) | 12 - 60 (24-120) | 12-352 (24-704) | 12 - 616 (24- 616) |
| ABSOLUTE TIME RANGE- IIOP IN MS | 331 - 421 (331- 421) | 931- 1832 (931-1832) | 2724-4274 (2724-3505) | 330 - 370 (351-451) | 881-1322 (751-1472) | 2803-3694 (2843-3694) |
| ABSOLUTE TIME RANGE - JRMP IN MS | 160 - 240 (160- 240) | 840- 1343 | 1875-2695 (1875-2273) | 160 - 160 (160-301) | 591-1031 (540-1213) | 1912-2654 (2002-2654) |
| AVERAGE RELATIVE TIME DIFFERENCE IIOP>JRMP IN MS | 176 (176) | 290 (290) | 1212 (1040) | 140 (120) | 290 (235) | 965 (940) |

**FIGURE 15. SUMMARY OF PERFORMANCE DATA (COMPARISON BETWEEN IIOP AND JRMP)**

| | IIOP | JRMP |
|---|---|---|
| SET UP TIME | 0.219 SEC. | 1.034 SEC. |
| NUMBER OF PACKETS FOR SET UP | 21 | 34 |
| DATA PACKETS/OVERHEAD PACKETS | 35% | 47% |
| NUMBER OF OVERHEAD PACKETS FOR EACH INVOCATION | 2 | 6 |

**FIGURE 16. SET UP DATA COMPARISON**

[3] K. Kono, University of Electro-Communications, Efficient RMI: Dynamic Specialization of Object Serialization, International Conference on Distributed Computing Systems, 2000.

[4] S. Ahuja, R. Quintao, Performance Evaluation of Java RMI: A Distributed Object Architecture for Internet Based Applications. Proc. of the Eighth International Symposium of Modeling Analysis and Simulation of Computer and Telecommunication Systems, IEEE 2000.

[5] jGuru, Java Developer Connection, Fundamentals of RMI Tutorial, http://developer.java.sun.com/developer/onlineTraining/rmi/RMI.html.

[6] RMI Wire Protocol, Java Remote Method Invocation Specification, http://www.javasoft.com/docs/rmi/spec/rmiTOCdoc.html, Sun Microsystems 1999.

[7] Java Language Mapping to OMG IDL Specification, http://cgi.omg.org/cgi-bin/doc?ptc/00-01-08, July 2000.

[8] CORBA/IIOP 2.3.1 Specification, formal/99-10-07 incorporates Objects by Value, http://cgi.omg.org/cgi-bin/doc?formal/99-10-07, 2000

[9] Akira Andoh, Simon Nash, RMI over IIOP Tutorial, JavaWorld, http://www.javaworld.com/javaworld/jw-12-1999/jw-12-iiop_p.html, December 1999.

[10] S. Campadello, O. Koskimies, K. Raatikainen, H. Helin, Wireless Java RMI , IEEE Proceedings 2000.

[11] S. E. Abdullahi and G. A. Ringwood, Garbage Collecting the Internet: A Survey of Distributed Garbage Collection, ACM Computing Surveys, Vol. 30, No. 3, September 1998.

[12] D. Plainfosse and M. Shapiro, BROADCAST Project. A Survey of Distributed Garbage Collection Techniques, Esprit Basic Research Project 6360, November 9, 1994.

## 10. Acknowledgments

Partial support for this work was provided by the National Science Foundation through grant CCR-0098235 and ITR-0113335. We also acknowledge gratefully the support given by Chia-Tien Dan Lo and Witawas Srisa-an of the Computer Science Department, Illinois Institute of Technology.