DD

KEK Preprint 97-153
September 1997
H/D

Prototype Performance of Distributed DAQ using HORB based on Java

Y. YASU, H. FUJII, E. INOUE, H. KODAMA
and
Y. SAKAMOTO

Sw0819

*Submitted to the 10th IEEE Real Time Conference 97
Beaune, France, September 22 - 26, 1997.*

** From April 1, 1997, High Energy Accelerator Research Organization (KEK) was newly established. The new organization is restructured of three research institutes, National Laboratory for High Energy Physics (KEK), Institutes of Nuclear Study (INS), Univ. of Tokyo and Meson Science Laboratory, Faculty of Science, Univ. of Tokyo.*

High Energy Accelerator Research Organization (KEK), 1997

KEK Reports are available from:

Information Resources Division
High Energy Accelerator Research Organization (KEK)
1-1 Oho, Tsukuba-shi
Ibaraki-ken, 305
JAPAN

Phone: 0298-64-5137
Fax: 0298-64-4604
Cable: KEK OHO
E-mail: Library@kekvox.kek.jp (Internet Address)
Internet: <http://www.kek.jp>

Prototype performance of Distributed DAQ using HORB based on Java

Yoshiji Yasu¹, Hirofumi Fujii¹, Eiji Inoue¹, Hideyo Kodama¹ and Yasunobu Sakamoto²,

¹Department of Physics, National Laboratory for High Energy Physics (KEK), 1-1 Oho, Tsukuba, Ibaraki 305, Japan

²Department of Physics, Yamagata University, Kojirakawa 1-4-21, Yamagata, Yamagata 990, Japan

Abstract

Network programming is a very important technology for next generation of distributed DAQ system. Java has powerful functionality not only in GUI but also in network programming. Execution speed of Java program is slow on Java interpreter. We have investigated various benchmark programs on Java interpreter, Java Just In Time compiler and Java compiler which generates native codes, and evaluated the performance of them in comparison with that of the C native codes. We found that the performance of the Java compiler was nearly as good as that of the C native codes. Hence, a prototype of Java-based DAQ has been developed. Our goal is to establish 3 tier (DAQ client, DAQ server and DAQ database) model for the DAQ.

I. INTRODUCTION

The next generation of DAQ needs network-based DAQ, WEB-based GUI as DAQ client, object oriented programming, distributed object oriented database and platform independence. This requirement lead us to Java programming, distributed object oriented technology and object oriented database. We also studied network-based data acquisition systems[1, 2].

We expect that Java will be a major programming language and Java programming will be everywhere in HEP computing. Java has excellent features such as a pure object oriented programming language, a unified API (JDK), an easy programming scheme for GUI and network, and is platform independent.

HORB[3, 4] is a distributed object oriented language plus object request broker. It has powerful features; Dynamic remote object creation/connection, URL-based object naming, object transfer (synchronous and asynchronous), security with a distributed access control list. There is no modification of Java syntax. The HORB architecture is shown in Fig.1. A client object in a client system calls methods of the server object in a sever system as if the server object exists locally.

Among many database systems, object oriented database like ObjectStore/PSE[5] is one of the very simple one to use. The sample program is shown in Fig.2. The main method is the "main", which calls "ObjectStore.initialize" method, "createDatabase" method and "readDatabase" method in this sequence. The "createDatabase" creates data of two modules, ADC

Fig. 1 HORB architecture

and Scaler. After "Transaction.begin" method in the method is called, the object of two modules is guaranteed to be persistent by calling "tr.commit" method. The persistent object is saved partially. This means that only the modified data are saved. This feature is useful for a real-time application.

II. JAVA BENCHMARK

Java has excellent features, but the execution speed of Java program on Java interpreter is too slow for the DAQ server, which is explained in later section. Then, Java performance of Java interpreter(java), Java Just In Time compiler (kaffe) and Java compiler(jmake of TowerJ[6]) were measured, and then they were evaluated in comparison with the execution speed of the program compiled by C compiler (gcc). The computer used was PentiumII-266MHz/Linux2.0.29. JDK1.0.2, JDK1.1.1, TowerJ which is a product of Tower Technology Corporation, and gcc2.7.2 were used. The "jmake" can run on many operating systems such as Solaris, SunOS, AIX, HP-UX, IRIX, Windows/NT, Linux and NextStep.

A. Linked List

A benchmark program to execute Linked-List algorithm was used. The Linked-List algorithm is used in a buffer management. The number of the list is 1000 and 1000 iterations were repeated inserting them to the tail/head and removing them from the head/tail. Table.1 shows the result. The execution speed of the program on Java interpreter in JDK1.1.1 was faster than that in JDK1.0.2. The optimization "-O" of "javac" was not effective. The speed of the program on "kaffe" was faster than that on the interpreter, but not sufficiently fast for the DAQ server. The speed of the program compiled by

```

class Module {
    String name;
    int station_number;
    Module[] module;
    Module(String name, int station_number);
    this.name = name;
    this.station_number = station_number;
}
public static void main(String args[]) {
    String db_name = "module.odt";
    ObjectStore.initialize(null, null);
    Database db = createDatabase(db_name);
    readDatabase(db);
}
static Database createDatabase(String db_name) {
    Database db;
    db = Database.create(db_name,
        Database.allRead | Database.allWrite);
    Transaction tr = Transaction.begin(Transaction.update);
    Module scaler = new Module("scaler", 5);
    Module adc = new Module("adc", 1);
    Module module [] = {scaler, adc};
    db.createRoot("Module", module);
    tr.commit();
    return db;
}
static void readDatabase(Database db) {
    Transaction tr =
        Transaction.begin(Transaction.readOnly);
    Module[] m = (Module[])db.getRoot("Module");
    for(int i = 0; i < m.length; i++) {
        System.out.print("module : "+m.getName());
        System.out.println(" "+m.getStation());
    }
    tr.commit();
}
public String getName() { return name; }
public int getStation() { return station_number }
}

```

Fig. 2 A sample program of ObjectStore/PSE

“jmake” including optimization was nearly as good as that with “gcc” including optimization.

B. Linpack and Whetstone

Linpack and Whetstone are famous benchmark programs for a scientific computation. The programs can be used to evaluate high performance computation. Table.2 shows that both “java” and “kaffe” performed poorly. But the performance of “jmake” reached that of “gcc” for Linpack while that of “jmake” was almost as good as that of “gcc”. The Linpack program was compiled with double precision and 100 million whetstone instructions were processed on the Whetstone program.

C. Data Copy

High speed data-copy is required for the DAQ because multiple data-copies occur in the DAQ. Table.3 shows the result. The “java” in JDK1.1.1 was used while “jmake” for JDK1.0.2 was used. The UNIX system call “memcpy” was an excellent performer, but “java” was too slow.

Table 1
Performance of Linked-List algorithm

JDK1.0.2	Execution Time
javac; java	40.5 sec.
javac -O; java	40.5 sec.
jmake;	3.41 sec.
jmake -optimize;	0.81 sec.
JDK1.1.1	Execution Time
javac; java	14.0 sec.
javac -O; java	14.0 sec.
javac; kaffe	3.40 sec.
javac -O; kaffe	3.40 sec.
C compiler	Execution Time
gcc	1.22 sec.
gcc -O	0.56 sec.

Table 2
Performance of Linpack and Whetstone

	Linpack	Whetstone
JDK1.0.2	Performance	Execution Time
javac; java	0.63 Mflops/s	1.60 sec.
javac -O; java	0.63 Mflops/s	
jmake;	16.4 Mflops/s	
jmake -optimize;	50.1 Mflops/s	
JDK1.1.1	Performance	Execution Time
javac; java	1.55 Mflops/s	6.71 sec.
javac -O; java	1.55 Mflops/s	
javac; kaffe	5.5 Mflops/s	
javac -O; kaffe	5.5 Mflops/s	
C compiler	Performance	Execution Time
gcc -O;	54.9 Mflops/s	0.69 sec.

Table 3
Performance of Data Copy

Data Length in bytes	Java for-loop in MB/s	TowerJ for-loop in MB/s	C native for-loop in MB/s	C native memcpy in MB/s
256	1.1	100	203	237
1024	1.1	105	210	632
4096	1.1	105	211	1078
16384	1.1	100	172	191
65536	1.1	100	170	190
262144	1.1	68	46	98
1048576	1.1	46	46	56

A high performance data-copy function available in java program, is necessary.

D. TCP/IP Socket

The measurement was done on 2 computers via a switching HUB of 100Mbps fast Ethernet. One of them was the PentiumII-266MHz with SMC9332 fast ethernet PCI card including DEC Tulip chip and another was

PentiumP5-120MHz/Linux2.0.29 with 3COM905XL fast Ethernet PCI card. The switching HUB was ACCTON SwitchHub-8e, which in turn was a MAC level bridge. Its maximum bandwidth was 260Mbps. Fig.3 shows essence of the program written in Java. There were two

```

Client:
Socket client = new Socket(host, port);
DataOutputStream out =new
    DataOutputStream(client.getOutputStream());
for( int i = 0; i < loop; i++)
    out.write(bufw, 0, length);
Server:
ServerSocket server =new ServerSocket(port);
Socket client = null;
client = server.accept();
DataInputStream in =new
    DataInputStream(client.getInputStream());
for( int i = 0; i < loop; i++) {
    realcount = in.read(buf, 0, length);
    |
}

```

Fig. 3 Essence of the TCP/IP programs

measurement cases. One was the client and the server on the PentiumII and another was the client on the PentiumP5 and the server on the PentiumII. Various length data set was transferred from the client to the server. The Java used was JDK1.1.1. In both cases, the overhead of the execution on "kaffe" was larger than that on "java", but the execution speed on "kaffe" was faster than that of "java". For large length of data, the speed on "java" was 9 MB/sec. The result in the case of the transfer between the PentiumII and the PentiumP5, is shown in the Table.4.

Table 4
Transfer speed with TCP/IP Socket

Data Length in bytes	Java in μ sec.	Kaffe in μ sec.
1	33.7	51.0
256	62.7	82.3
512	91.9	105
1024	143	157
2048	238	255
1048576	110700	109340

E. HORB

The measurement was done in the same condition as that for the TCP/IP Socket. The execution time of the remote object creation, the remote object connection and the remote method calls in several conditions were measured. In case 1, there were the client and the server on the PentiumII. In case 2, there were the client on the

PentiumP5 and the server on the PentiumII. The result is shown in Table.5. The remote object creation took 11

Table 5
Performance of HORB

	Case 1	Case 2
Remote Object Creation	10.9 msec.	64 msec.
Remote Object Connection	7.0 msec.	60 msec.
Remote Method Call		
Variable Parameters	in msec.	in msec.
1 parameter	0.65	0.91
2 parameters	0.65	0.95
3 parameters	0.67	1.00
4 parameters	0.69	1.06
5 parameters	0.71	1.09
6 parameters	0.76	1.13
Object Passing	in msec.	in msec.
Data[1]	1.17	1.94
Data[10]	2.42	8.64
Data[20]	3.89	16.15
Data[30]	5.32	24.17
Data[40]	6.77	31.73
Data[50]	8.26	39.42

msec and the remote object connection took 7 msec in the case 1. The remote object creation took 64 msec and the remote object connection took 60 msec in the case 2. A publication[3] emphasizes the remote-object-connection speed on HORB was twice faster than that of JavaRMI, which is included in the JDK1.1.1 distribution kit. In the remote method call, the parameter was one "int" variable. It did not take long even if the number of the parameter was increased. In the object passing of the remote object call, the execution time increased according to the depth of the array, which was an "int" variable defined in a Data class. The performance of the object passing on HORB was better than that on JavaRMI[4].

III. PROTOTYPE OF JAVA-BASED DAQ

A. Java to C and FORTRAN interface

For the Java-based DAQ, the CAMAC routine written in C language can be used by using Java interface to CAMAC library. FORTRAN program may be required for using CERNLIB as the DAQ analyzer. Sample programs written in Java for calling C and FORTRAN programs are shown in Fig.4. For the CAMAC, "cc.open" is called for the CAMAC initialization and "cc.camac" is called for getting/putting data from/to CAMAC module while the constructor "ISAcc7000" is called before that. "ana.init" and "ana.exec" call the FORTRAN program via Java to FORTRAN interface written in Java when the FORTRAN program using CERNLIB is to be called.

B. DAQ Model

The DAQ model was investigated for the Java-based DAQ. Buffer manager has an important role in the

```

Java Interface to CAMAC Library :
ISAcc7000 cc = new ISAcc7000();
cc.open(CrateNo, Base, IRQ);
status = cc.camac(..., data, ... );
System.out.println("Scaler data");
System.out.println( data[0] );

Java Interface to FORTRAN Program :
Anal ana = new Anal();
ana.init();
ana.anaexec( array );

FORTRAN Program :
subroutine anainit()
common /pawc/hmemory(..)
call hbook1(...)
end
subroutine anaexec( array )
common /pawc/hmemory(..)
call hf1(...)
end

```

Fig. 4 Sample programs for calling C and FORTRAN programs

DAQ. There are data flow model, message flow model and process management model in the DAQ model. At first stage of the Java-based DAQ project, UNIDAQ[7] model was adopted as the data flow model and the process management model. That is, the UNIDAQ buffer manager (NOVA) was used while the Java interface to the NOVA library was developed. As the message flow model, UNIDAQ made "ask command" and "status path". The message flow model of the Java-based DAQ adopted HORB. The remote object call and the daemon object made the implementation of the model simple. The Fig.5 shows a sample program of an collector server and "ask command". The ask command calls the methods in the server remotely.

C. Performance of Java-based DAQ

The round-trip time of the Java-based DAQ was measured. One collector server can read a buffer pointer and then put the pointer in the buffer manager. The collector server was written in Java and run on "java". It took 120 μ sec while that written in C language took 90 μ sec as shown in the Table.6.

Table 6
Performance of data-copy

Java	C native
120 μ sec.	90 μ sec.

D. Three-tier model of the Java-based DAQ

The prototype of the Java-based DAQ was developed. Our goal is to establish 3-tier model of the Java-based

```

Collector Server :
public static void main(String args[])
    throws HORBException {
    HORBServer hs = new HORBServer(port);
    Process_IF collector = new Collector();
    HORBServer.registerObject("Process_IF",
        collector,"collector", 0);
    novajava nova = new novajava();
    novaid = nova.open("/tmp/nova", -1, 1);
    while(true) {
        |
        status = nova.get(novaid, pointer, size);
        user_event(pointer, size);
        status = nova.put(novaid,pointer,size, 70);
        |
    }
}

public void begin(...){...}
public void resume(...){...}

Interface definition :
interface Process_IF {
    public void begin(...);
    public void resume(...);
    |
}

Ask command :
public static void main(String args[]) {
    |
    url = "horb://" + host + ":" + port + "/" + daemonObject;
    Process_IF server = new Process_IF_Proxy(url);
    if(command.compareTo("begin")==0)
        server.begin(...);
    if(command.compareTo("resume")==0)
        server.resume(...);
    |
}

```

Fig. 5 Template of Java-based DAQ

DAQ. The 3-tier model consists of a DAQ client tier, a DAQ server tier and a DAQ database tier. Recent progress in Java computing in industry resulted 3-tier model of business application. The relative importance of the client computation such as WEB client increases more and more. There is the same situation in the DAQ. As DAQ client, there are run control, event display, status display, data viewer, etc. WEB browser such as Netscape Navigator or Microsoft Internet Explorer will play a major role in the DAQ. The DAQ server must not only have high quality and high reliability but also should be well-matched to the WEB and the database. The distributed object oriented technology based on Java, is suitable for building the DAQ servers, which consists of collector, recorder, event builder, analyzer, WEB server, etc. In the DAQ database, there are DAQ configuration parameters, raw data, run summary data, analysis data, etc. The object oriented database being very simple for the usage, is convenient for the real-

time environment, that is, the DAQ system.

IV. CONCLUSION

Java, distributed object oriented communication technology (HORB) and object oriented database (ObjectStore/PSE) was chosen as the next generation of DAQ, that is, Java-based DAQ.

Java benchmark has shown that Java compiler (TowerJ) had good performance.

Prototype of Java-based DAQ was developed.

Our goal is to establish 3 tier model of the Java-based DAQ.

At next stage of the Java-based DAQ, new data flow model and new process management model will be investigated, and the message flow model will be enhanced. The management model of a distributed database in the Java-based DAQ will also be studied. On real-time environment, Java-based DAQ must run. Recent real-time (micro) OS will be investigated for the Java-based DAQ.

V. ACKNOWLEDGMENTS

We appreciate Dr. Hirano and Dr. Igarashi for their help. We would like to thank Prof. Nomachi, Prof. Sakamoto, Prof. Watase and Prof. Kondo for their encouragement.

VI. REFERENCES

- [1] Y.Yasu et al., A Study of Network-based Data Acquisition System, KEK-PREPRINT-97-6, April 1997. 4pp. Talk at CHEP'97, the International Conference on Computing in High Energy Physics, Berlin, Germany, 7-11 April 1997.
- [2] <http://kuhes1.pn.scphys.kyoto-u.ac.jp/konoe/>
- [3] Satoshi Hirano "HORB:Distributed Execution of Java Programs" Worldwide Computing & Its Applications '97, March, 10-11, 1997 Tsukuba, Japan
- [4] <http://ring.etl.go.jp/openlab/horb/>
- [5] <http://www.odi.com/>
- [6] <http://www.twr.com/>
- [7] M.Nomachi et al., UNIDAQ, Proceedings of the International Conference on Computing in High Energy Physics '94, LBL-35822(1994), pp.114-116.