ISSN 2047-3338

Java Based High Performance of Parallel Processing Application with JavaParty

Mohammad Javad Abbasi[1], Somayeh Koupaeimalek[2], Suren Khachateryan[3] and Muhammad Shafi Abd Lattif[4]
[1]Department of Computer Science, State Engineering University of Armenia
[2,4]Universiti Technologi Malaysia
[3]American University of Armenia

***Abstract*— To achieve better performance in parallel processing, which is implemented based on java application. In this article, we work on high performance of parallel processing in a local network that computer can be connected as a form of the cluster making available Java Virtual Machine (JVM). However, java is not suitable for parallel processing, JavaParty adds Remote Method Invocation (RMI) to the java application. This article present performance of parallel processing based on java using JavaParty. The result shows the performance of parallel processing has resulted in decrees the execute time by adding JVM. Advance in JavaParty had provided easy to use tools and environment or the development of parallel application.**

***Index Terms*– Parallel processing, Java, JavaParty, Remote Method Invocation, Cluster and Execute Time**

I. INTRODUCTION

RECENT advance in popular computer technology has great advance. Hardware device are very inexpensive and power of computing continues to grow up today. The effective of cheap personal computer to get powerful computing system is to provide cluster of computers using open source software and hardware devices linked with high speed network connection system. This offer a possible solution with reasonable price and avoiding issue provide high performance in parallel computing. Parallel computing can be executed more powerful and faster when use multiple computer instead of just use one type of the same computer.

Java is one high-level of programming language that has great effect in use with high performance parallel computing. Because when write program with java language it is ideal for the safe object oriented programming type for writing faithful and constant on every scales. Unfortunately, there are many problem of parallel method in java programming language that makes it a lesser amount of favored selection. These hurdles are lack of multidimensional array, insufficient of complex number, performance of floating point and sequential of software execute. However, inter- process mechanism is mean issue problem to run java program that we called it Remote Method Invocation (RMI).

RMI made some problem during run in cluster environment with following:

- The speed of RMI is a very slow in low latency and high bandwidth environment.
- In cluster environment RMI's overhead for determine the problem in network is too verbose.
- Lower efficiently and sustainability happen when the size of program increase.
- Implemented program with RMI is boring, difficult to manage and spend more time.

JavaParty is major solution for those problems we mention above in distributed environment. Traditional java support distributed parallel with synchronizations and thread method. While multithreads are limited to single address space JavaParty extend the ability of java to distributed computing environment. JavaParty can declare class as remote which can process in different distributed machine. While javas are not able run program in one single machine remote class and their instance are easily accessible in distributed cluster. As far as, remote classes connected with high bandwidth and low latency JavaParty can be shows a distributed virtual machine over several computers. However, it is different for design traditional fault tolerance computer network communication over long distance.

II. RELATED WORK

Though for distributed computing there are many software package, just java implementation are explored in this section.

A. Javaparty

JavaParty use Remote Method Invocation for made easy port for multi-threaded to distributed cluster, it can built library to connect with another node in distributed environment. In java code we should tag the remote code to clear which code should to be run remotely in distributed cluster. The JavaParty Compiler (JPC) generates suitable code required to implement remote method invocation. One important advantage of JavaParty is that it considerably reduces the time to write a parallel Java program.

B. MPIJava

Massage Passing Interface (MPI) is an object oriented interface for distributed environment. MPI provide the high performance of communication library and application topology by providing java binding. It does not consider to the java application, where it able to work with any platform that made well-matched java language and MPI environment.

C. Manta

Manta is one java compiler that cans java codes to x86 executable that is very faster than other java implementation, same as JavaParty. Manta able supports the java language, for example garbage, collection and exception. However, manta is high efficient it does not have RMI package for emot interface.

III. JAVAPARTY

Java party a distributed java virtual machine in top of regular java virtual machine that execute on the nodes of workstation cluster .however this techniques can really be applied to object-oriented and since, in general, it's not in array-based .more over parallelism dose not stem from for all loop, do across-loop or do all-loop but instead expressed by mean of thread object to make object accessible from other node.

Java's remote method invocation (RMI) used instead of insert many verbose RMI command manually in to his multi-threaded java application to port is from single workstation to cluster ,java party allows him to declare class to be remote .java party then generate all the necessary RMI command automatically.

JavaParty provided the multi –threaded java program to a remote environment such as heterogeneous. Java's language supports the threads and synchronization methods. While multi-threaded Java programs are incomplete to a single address space, JavaParty extends the capabilities of Java to distributed computing environments.

The regular way of porting a parallel application to a discrete condition is the use of a communication library. Java's Remote method invocation (RMI) reduces the execution of communication protocols unnecessary, but still indications to increased program difficulty. The motives for increased complexity are the limited RMI capabilities and other practicality that must be fulfilled for formation and access of remote objects.

A. Java RMI Architecture

Fig. 1 shows the different layers of the Java RMI architecture. These layers include stubs/skeletons, reference layer, and remote transport layer.

1) Stubs and Skeletons: In javaparty RMI can use stubs and skeletons to communicat with remote objects. A stub for a remote object acts as a client's local representative or proxy for that object.

On the server side the skeleton object inverse transformation parameter list and result value by restoring by correct type of the parameter, invoking the request method on

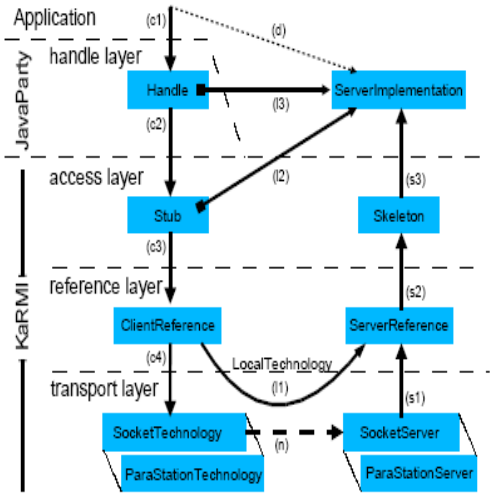

Fig. 1: Java RMI Architecture

the server implementation object and convert in the result back to generic representation to the caller.The caller invokes a method on the local stub which is responsible for carrying out the method call on the remote object. In RMI, a stub for a remote object implements the same as set of remote interfaces that a remote object implements.

2) Reference Layer: The reference layer responsible for uniquely addressing remote object and dispatching incoming request to the correct server the object number combination with internet address and port ,the object was export on the identities the object in distributed environment .

3) Transport Layer: The transport layer it is responsible caching connection to remote machine using the appropriate network technology . the transport technology also defines the wire protocol that is used for the communication .there are two transport technology's implemented for KRMI ,on for regular TCP/IP socket called Socket Technology and on for the Para station network layer over Myrinet[1] communication hardware.

IV. COMPILING, AND RUNNING APPLICATION

A. JavaParty Syntex

Java language modifier with remote that we called it JavaParty. A class declaration can be prepended with this remote modifier to state a class a remote class [9]. Hear we shows sample code of remote class.

```
public remote class hello {
      /** instance variable of remote
      class */
      public int x;
      public void foo() { ... }
      public static int y;
      public static void bar() { ... }
  }
```

As you see in sample code there is not different between remote class code and non-remote class code. Instance, in remote class just add remote syntax for communication with other object and also they can link together. In below show

other instances of instantiating and opening an adaptable of a remote class remain assumed as follows:

```
hello r = new hello()
created remote object
r.x = 42;
r.foo()
hello.y = 13;
class
hello.bar();
```

The example of working in JavaParty is used to test whether an object is to be an illustration of a convinced class at runtime. Remote objects may be allocated to variables that are confirmed to be of type java.lang.Object [9]. This is shown in the bellowing code.

```
Object obj = new hello();
remote class R.
if (obj instanceof hello) { ... }
```

B. Setup

In order to run and compile the JavaParty, the JavaParty package must be installed in your java directory. Requirements, downloads and setup registration are available from [9].

C. Compiling Applications

When javaparty has been registered, java application program can be written with javaparty language code and run this code. We can compile javaparty code in any PC with java 1.4.2 and installed javaparty package. To validate the process, a version of Hello World called HelloJP provided by [9] will be used:

```
package examples;
public remote class HelloJP {
public void hello() {
// Print on the console of the virtual
machine where the
// object lives
System.out.println("Hello JavaParty!");
   }
public static void main(String[] args) {
for (int n = 0; n < 10; n++) {
HelloJP world = new HelloJP();
world.hello();
}
}
   }
```

Hear we use following stage to compile the HelloJP code:

1) Save the HelloJP with HelloJP.java file name.

2) Create a directory named classes where you wish to exist in your use classes to.

```
mkdir classes
```

3) Compile the code.

```
jpc -d classes HelloJP.java
```

D. Running Applications

Hear we show how to run HelloJP:

1) From the location of classes, set the class path. For clusters without a distributed file system, a reliable copy of all the classes with the same path must be copied to each machine.

```
setenv CLASSPATH classes
```

2) Run the JavaParty application.

```
jpc examples.HelloJP
```

Remote objects are formed on the VM and the output message is printed on the head machine console.

```
Hello JavaParty!
```

V. SIMULATION RESULT

In our test we use eight pc and all had Intel operator Dual-core x86, 64bit CPUs running as 2600 MHz. The names of this pc are jp1, jp2, jp3, jp7and jp8. All pc had 1 GB of ram.

The main goal of this process is characterizing the whole system or the subsystem in order to find the potential performance.

In this paper we measure benchmark for three purpose: measuring system performance, measuring the subsystem performance, and measure the change of system performance after and before comparisons.

We can measure benchmark the ability of pc to execute of test. In order the results are recorded and they able to use as reference to compare with the other benchmark measurement.

In any time the software and hardware has been change the benchmark can completed for comparison in after and before changing. For improvement we often compared with money and time cost to make improvements in benchmark. However there are more open source program for cluster and most of them designed for MPI and heterogeneous. In JavaParty web site there is available benchmark program for heterogeneous and MPI which is easy to run and compiled. In first examination of benchmark, it shows us that most of this benchmark compile and run too easy and quick to notification any significant improvements with the heterogeneous cluster. This was more expected outstanding to the fact that these benchmarks were calculated to run on PC from the late 90s. Even though disappointment with not actuality capable to apply best of the benchmarks, one, named electrostatic algorithm, presented a more consistent, longer run time offering a more precise and accurate benchmark. A new routing algorithm on a grid is tested. It is based on electrostatic interactions between obstacles and a conventional particle, whose motion trajectory denotes a path between two given grid cells. The algorithm inherits the power of maze routing in that it is able to route huge matrix with various obstructions. The large memory requirement of the conventional maze algorithm is alleviated through successive net refinement, which constrains the maze searching to small regions. The algorithm shows advantages in routing huge configurations with sparse layout of the obstacles.

The furthest loop has been parallelized using a cyclic distribution for load balance. After the program completes, the elapsed time is shown.

The Electrostatic algorithm JavaParty code was downloaded from [9]. It was then compile by `jpc` and run by `jprm, jpvm`.

The simulation was run 40 sequential times with 2 PC, 3 PC, 4 PC, 6 PC and 8 PC. The nasty and normal eccentricity of the usage time for each set was calculated. The simulation results are shown in Table 1 and are shown in Figure 2. It seems from these results that the speed-up of electrostatic algorithm is depending to the number of PC involved in the accomplishment time. Comparing the mean execution times, adding two more PC to the original two reduced the execution time meaningfully from 68.5 seconds to 57.3 seconds. Adding six PC to the original two declined the run time to 19.4 seconds.

Table 1: electrostatic Result Summary

PC No	2	3	4	6	8
Time	68.5	57.3	43.5	31.3	19.4
Sum	29	18.7	16.8	11.1	4.87

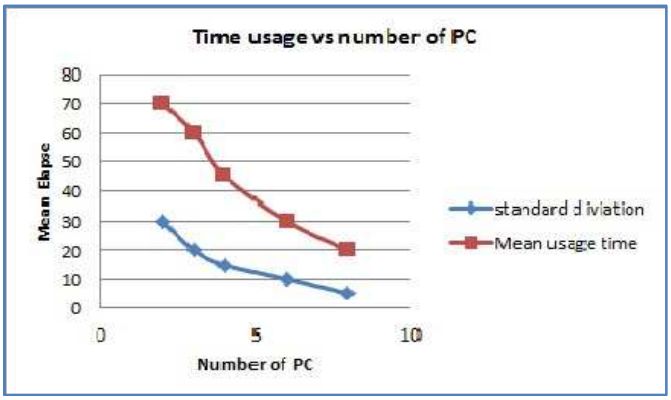

Fig. 2: Execution times for electrostatic routing

During the running simulation some of the machine did not continuously run. For instance, when jp1 and jp2 were configured to run and compile the electrostatic algorithm rarely jp1 CPU's assist in the execute time of the code. Due to this happening, the normal deviation significantly going higher than estimated, particularly when we use fewer PC for running.

VI. Conclusion

This Paper presented that while executing a parallel Java method with JavaParty, the execution time can be reduced by adding so many extra virtual machines. For increase the performance and QoS profile we can use cluster interconnect to arrange the JavaParty traffic between PC. During this research we understand the JavaParty is elegant way to implement a heterogeneous cluster with java application. In comparison of JavaParty and RMI, the JavaParty is easier than RMI to work with it. Similarly matched to traditional RMI, JavaParty programs also become accustomed more compliantly to numerous network situations and can achievement locality. In this implementation, we presented techniques to realize local access to transparent remote JavaParty object within the same order of magnitude as regular java method invocation. JavaParty's improved runtime structure and serialization is more rapidly and more capable associated with Java's normal environment providing a more suitable package for a carefully related group.

References

[1] M. Philippsen and M. Zenger, "JavaParty: Transparent Remote Objects in Java,"IEEE Concurrency, Vol. 9, No. 11, pp.1125-1242, 1997.

[2] W. Stallings, Operating Systems. 4th ed., Upper Saddle River: Prentice Hall,2001, p. 779.

[3] Manta: Fast parallel java. Retrieved May 20, 2007 from www.cs.vu.nl/manta.

[4] B. Haumacher, M. Philippsen, and C. Nester. A More Efficient RMI for Java.Concurrency: Practice and Experience, 12(7):495-518, May 2000.

[5] P. Gray and V. S. Sunderam. IceT: Distributed Computing and Java. Retrieved May 9, 2007 from http://citeseer.ist.psu.edu/192886.html.

[6] Michael Philippsen and Bernhard Haumacher. Locality optimization in JavaParty by means of static type analysis, In Proceedings of the EuroPar'98, Southampton, England, September 2-3, 1998, September 1998.

[7] Matthias Jacob, Michael Philippsen and Martin Karrenbach. Large-Scale Parallel Geophysical Algorithms in Java: A Feasibility Study, In Concurrency: Practice and Experience 10(11-13):1143-1154, John Wiley & Sons, Ltd., Chichester, West Sussix, September-November 1998.

[8] Matthias Gimbel, Michael Philippsen, Bernhard Haumacher, Peter C. Lockemann and Walter F. Tichy. Java as a Basis for Parallel Data Mining in Workstation Clusters, In Proceedings of the 7th International Conference on High Performance Computing and Networking, HPCN Europe 1999 of Lecture Notes in Computer Science Volume 1593, pages 884-894, Amsterdam, April 12-14, 1999, Springer Verlag, Heidelberg, Germany, 1999.

[9] Christian Nester, Michael Philippsen and Bernhard Haumacher. A More Efficient RMI for Java, In Proceedings of the ACM 1999 Conference on Java Grande, San Francisco, USA, June 12-14, 1999, pages 152-159, 1999.

[10] Bernhard Haumacher and Michael Philippsen. More Efficient Object Serialization, In Proceedings of the International Workshop on Java for Parallel and Distributed Computing of Lecture Notes in Computer Science Volume 1586, pages 718-732, San Juan, Puerto Rico, April 12, 1999, Springer Verlag, Heidelberg, Germany, April 1999.

[11] Bernhard Haumacher and Michael Philippsen. Exploiting Object Locality in JavaParty, a Distributed Computing Environment for Workstation Clusters, In Proceedings of the Compilers for Parallel Computers (CPC2001), Edinburgh, Scotland, UK, June 27-29, 2001, pages 83-94, June 2001.

[12] Bernhard haumacher and Michael philippsen, Efficient local calls to potentially remote object in JavaParty,International conference of parallel processing in spania.