International Journal of Innovative Research in Computer Science & Technology (IJIRCST)
ISSN: 2347-5552, Volume-6, Issue-6, November 2018
DOI: 10.21276/ijircst.2018.6.6.1

Effective Techniques to Improve Network Load Balancing for Parallel Computation Using RMI

Nazmin Akter, Fuad Ahmed, Nusrat Jahan Shanta

Abstract— **For Parallel Computing Java Remote Method Invocation (RMI) provides a high performance flexible type of procedure. In current Java usage, RMI system is not very much efficient for communication. In this paper we have presented Distributed Parallel Computing using efficient multithreading algorithm and RMI system within 8 hosts. The hosts will carry out Prime number calculations that will also show how many prime numbers are there in the given number (user input). In single host this calculation will take much time & instructions are executed one after another. We have aimed at distributing the task among multiple hosts where the task will execute concurrently. For concurrency we have used multithreading. The results also show that the proposed system's performance increases when we use more highly configured PC.**

***Index Terms*—Concurrency, distributed parallel computing, multiple hosts, multithreading, *R*MI**

I. INTRODUCTION

Software development is advancing at a praiseworthy faster pace, Many software industries are developing applications that are made for the same purpose, we can only say which one of them is optimum based on their performance, and the faster it is the more likely it will get noticed and praised by the users. Thus it is more important for software professionals to develop the software using parallel computing which saves time, allowing the execution of applications in a shorter wall-clock time [2] and Java's RMI offers interesting opportunities to developers to build such applications.

Manuscript received November 09, 2018.

Nazmin Akter, Department of Computer Science & Engineering, Metropolitan University Zindabazar, Sylhet-3100, Bangladesh (e-mail: nazmin@metrouni.edu.bd).

Fuad Ahmed, Department of Computer Science & Engineering, Metropolitan University Zindabazar, Sylhet-3100, Bangladesh

Nusrat Jahan Shanta, Department of Computer Science & Engineering, Metropolitan University Zindabazar, Sylhet-3100, Bangladesh

RMI's design core goal is to support highly flexible programming of distributed applications including a seamless integration with Java's object model, heterogeneity and flexibility [5]. Threads provide an efficient and effective paradigm for utilizing tightly coupled systems and RMI [3]. This paper shows the result of how multiple hosts can concurrently perform the task significantly for faster using the RMI system than a single host where calculations take much time & instructions are executed one by one.

II. REMOTE METHOD INVOCATION

A. Introduction of RMI

RMI is a mechanism that comprises two separate programs, a server and a client. The server creates some remote objects, makes references to these objects accessible, and waits for clients to invoke methods on these objects. The client program obtains a remote reference to one or more remote objects on a server and then invokes methods on them. In this mechanism the server and the client communicate and pass information back and forth. It is referred as distributed object application. It is relatively easy to use & remarkably powerful technology and also allows the programmers to develop distributed Java programs with the same syntax and semantics used for non-distributed programs.

B. RMI Architecture

The RMI architecture defines how objects behave, how and when exceptions can occur, how memory is managed, and how parameters are passed to, and returned from, remote methods. RMI architecture creates a system that extends the safety and robustness of the java architecture to the Java architecture to the distributed computing world.

C. Interfaces: The Heart of RMI

The architecture is based on one important principle: the definition of behavior and the implementation of that behavior. RMI allows the code that defines the behavior and the code that also implements the behavior to remain separate and to run on separate JVMs [4]. In RMI, the definition of a remote service is coded using a Java interface. The implementation of the remote service is coded in a class. Therefore, the key to understanding RMI is to remember that interfaces define behavior and classes define implementation.

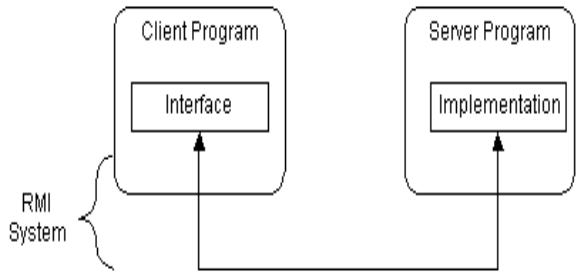

Fig 2A: the diagram illustrates this separation.

It supports two classes that implement the same interface. The first class is the implementation of the behavior, and it runs on the server. The second class acts as a proxy/stub for the remote service and it runs on the client. A client program makes method calls on the proxy object, RMI sends the request to the remote JVM, and forwards it to the implementation. Provided by the implementation are sent back to the proxy and then to the client's program.

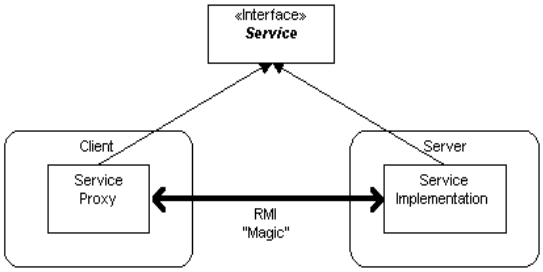

Fig 2B: diagram of RMI system implementing same interface.

D. How RMI works?

On a host machine, a server program creates a remote service by first creating a local object that implements that service. Next, it exports that object to RMI. When the object is exported, RMI creates a listening service that waits for clients to connect and request the service. After exporting, the server registers the object in the RMI Registry under a public name. On the client side, the RMI Registry is accessed through the static class Naming. It provides the method lookup () that a client uses to query a registry. The method lookup () accepts a URL that specifies the server host name and the name of the desired service [7]. The method returns a remote reference to the service object.

III. THE PROPOSED MODEL

A. Creating Distributed Application by using RMI

We have developed a distributed application by doing the general following steps:

1. Interface definition for remote service. (PrimeNumber.java)
2. Implementation of remote service. (PrimeNumberImpl.java)
3. Stub & Skeleton files.
4. A server to host the remote service. (PrimeServer.java)
5. A RMI Naming Service that allows clients to find the remote services.
6. A client program that needs the remote services. (PrimeClient.java)

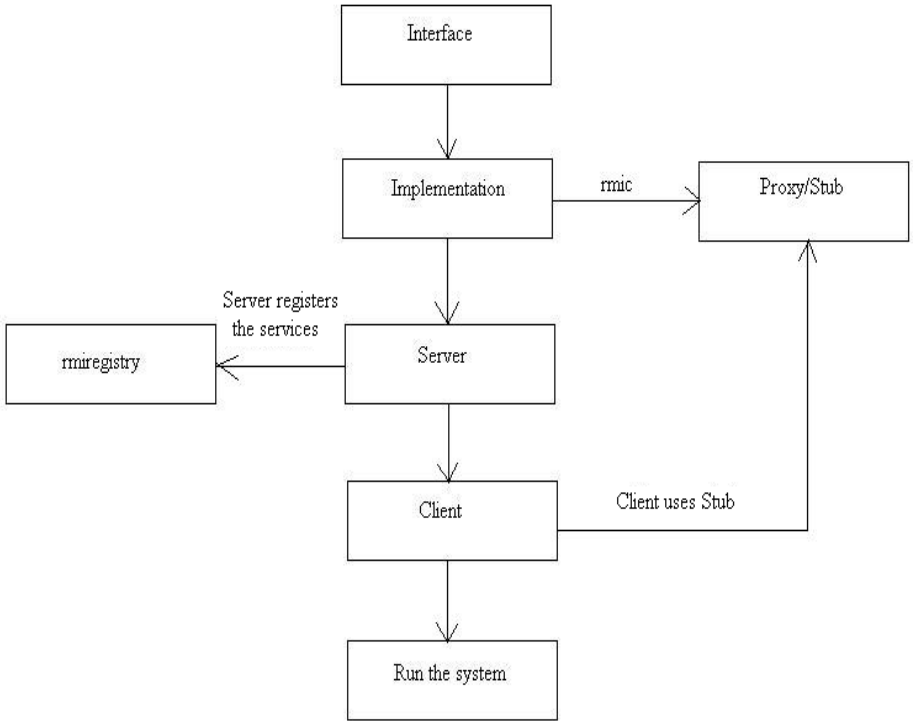

Fig 3A: Steps of RMI system

B. Steps to build RMI system

We have built the RMI system by doing the general following steps:

1. At first, we have written a java code for the interfaces (PrimeNumber.java) & compile it. We have implemented same interface in the server & client, where the stub class of client & the implementation class (PrimeNumberImpl.java) of server use this interface defines the behavior.
2. We have written a java code for the implementation classes which runs in server. We have implemented the behavior in the implementation, the behavior that we defined in the interface.
3. Then we have run the RMI compilers, rmic, which runs on the implementation class. It generates stub (PrimeNumberImpl_Stub) as we used jdk 1.5 versions.
4. We have written a java code for the remote service host program (PrimeServer.java) which contains only the service.
5. We have written a java code for the client program & compiled it .The client sends the request to do its task & receives the result.
6. Lastly, we have installed & run the RMI system. For this we started the registry: RMI *registry* on each server which contains the information of the remote service of each server. The client gets the service information from the RMI registry & hooks up with its desired server. Then we have run the servers & send the request from the client host.

C. Running RMI system

To start this system we need to start three consoles one for server, one for client & one for RMI Registry. As we have used 8 hosts as servers, thus each host starts two consoles one for server & one for RMI registry. And one host would start one console for client. Start the registry by entering the following: RMI *registry* the registry will start running. The next consoles of each server start their individual services by entering the following: *java Prime Server* It will start loading the implementation into memory & wait for a client connection. In the last console of client starts the client program
java Prime Client

IV. INPUT AND OUTPUT

Our system takes a single integer numbers as input that iterate from 1 to the input number to find the prime numbers and output will be the time it took to successfully find the prime numbers. We gave 50000, 100000, 150000, and 200000 as the input numbers. We first tested it out on a single host, then on the eight hosts, where we distributed the task evenly on the hosts and received the outputs that are given on the table.

S.No	Number	Single Host (time)	Eight Hosts (time)
1	50000	1.839	0.623
2	100000	5.094	1.7695
3	150000	11.86	3.712
4	200000	17.097	4.5863

Table 1: Time difference between Single Host and Eight Host to output the prime numbers

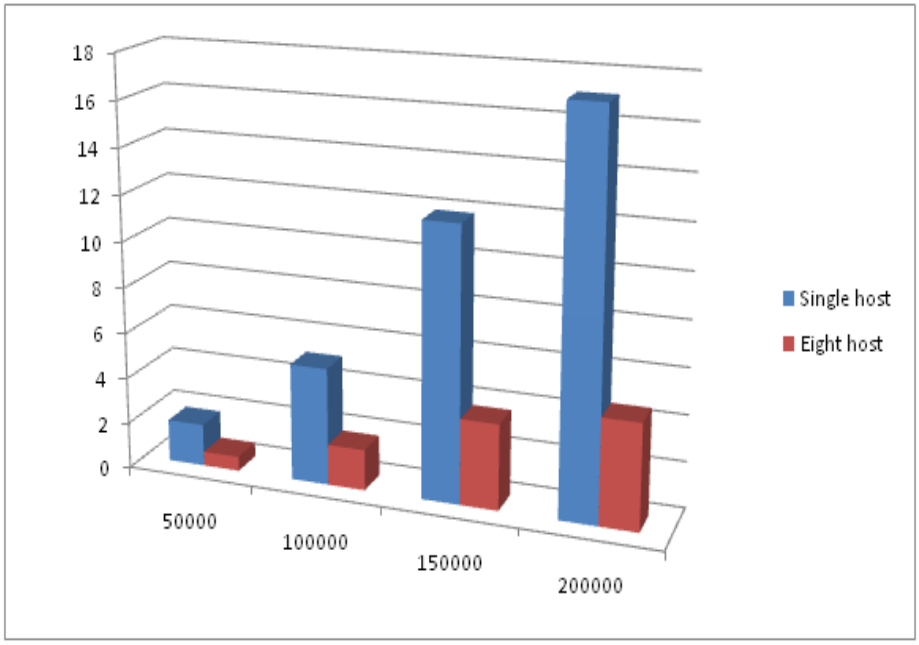

Fig 4A: Time difference between Single Host and Eight Host to output the prime number

V. CONCLUSION

By doing this research it is clear that how a task can be distributed among the multiple hosts. Firstly the given task is broken apart into discrete pieces of work then it can be solved simultaneously. As we have used parallel computing, it executed multiple program instructions and solved almost three times faster with multiple compute resources than with a single compute resource. Also we have explained that an efficient RMI implementation is a good basis for load balancing and writing high-performance parallel applications.

VI. REFERENCES

[1] M. Ozaki, Y. Adachi, Y. Iwahori, and N. Ishii, "Application of fuzzy theory to writer recognition of Chinese characters", International Journal of Modeling and Simulation , Vol. 18, No. 1, 1998, pp. 11–16.

[2] W. J. Book, "Modeling design and control of flexible manipulator arms: A tutorial review", Proceedings of the 29th IEEE International Conference on Decision and Control , San Francisco, CA, USA, 1990, pp. 500–506.

[3] R. E. Moore, Interval analysis, Englewood Cliffs, NJ: Prentice Hall, 1966.

[4] D. S. Chan, Theory and implementation of multidimensional discrete systems for signal processing, doctoral dissertation, Massachusetts Institute of Technology, Cambridge, MA , USA , 1978.

[5] Eggen, Dr. Roger and Maurice Eggen, "Efficiency of Distributed Parallel Processing using Java RMI, Sockets, and CORBA." .

[6] Chandra Kopparapu, Load Balancing Servers, Firewalls & Caches, Wiley, ISBN 0-471-41550-2

[7] J. Waldo," Remote procedure calls and Java Remote Method Invocation,"IEEE Concurrency, July–September 1998, pp. 5–7.

[8] R. Alonso, ''The Design of Load Balancing Strategies for Distributed Systems,'' Future Directions in Computer Architecture and Software Workshop, pp. 1-6, Seabrook Island, SC, May 5-7, 1986.

[9] K. Baumgartner and B. W. Wah, ''GAMMON: A Load Balancing Strategy for a Local Computer System with a Multi-access Network,'' Trans. on Computers, vol. 38, no. 8, pp. 1098-1109, IEEE, Aug. 1989.

[10] An article on Network Load Balancing Services http://en.wikipedia.org/wiki/Network_Load_Balancing_Services

[11] Zaki, M., Li, W., Parthasarathy, S. "Customized Dynamic Load Balancing for a Network of Workstations". Proceedings of HPDC '96, 1996.

[12] Springer Berlin / Heidelberg, Network and Parallel Computing, ISBN 978-3-540-2981-6

[13] Design Patterns, by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (The Gang of Four)

[14] Sun's RMI FAQ

[15] http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

