IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 19, Issue 2, Ver. V (Mar.-Apr. 2017), PP 70-75
www.iosrjournals.org

Big Data Analysis on Heterogeneous Distributed Systems using Remote Method Invocation

Dr. Mamta C. Padole
Department of Computer Science and Engineering, The Maharaja Sayajirao University of Baroda, India

Abstract: *Big Data Analysis has become very essential for organizations to arrive at timely decisions, for progressive growth and keeping with the competition. To analyse huge amount of data generated through various business processes, require High Performance Computing machines. But, owning HPC machines and maintaining them, requires large investment and experienced skill set. Moreover, it is difficult to install any proprietary and customized software, on special purpose HPC machines. To overcome all these issues, Distributed Systems can be implemented with the help of cumulative processing power available from large number of under-utilized individual workstations that are commonly available in all organizations. The paper emphasises on executing applications over heterogeneous distributed systems using Java Remote Method Invocation.*

Keywords:*Big Data Analysis, Heterogeneous Distributed Systems, Remote Method Invocation, Java RMI*

I Introduction

Big data analysis is inevitable to attain efficient decisions. Efficiency in decisions can be achieved only when analysis reports are accurate, timely and provide future insight. But business processes generate huge amount of data also referred as "Big Data". Any data that is huge in *volume*, comprising of *variety* of data, is generated at great *velocity* or is of great *value* provided its *veracity* i.e. accuracy can be assured, may be referred as Big data [1]. Big data may consist of structured, unstructured and semi structured data. The data may be of various types like text be it alphabets or numeric, images, audio, video [2]. Big data includes massive structured and unstructured data which is beyond the capacity of commonly used software tools to collect, curate, manage, and process in a definite time [3].

The source of Big Data may be variety of applications that include commercial applications like stock market, scientific applications like atomic reactions, gene/DNA sequencing, statistical data, Defence applications like border surveillance through radars that detect movements of objects, Health care applications generating cardiograms, mammograms, MRI scan reports, Production applications like Production pipeline information and work in progress, Delivery Logistics like FeDex, DHL parcel delivery status tracking, Government applications like consensus data, Aadhar data, tax estimation/computation, land mapping, Internet Search, Social Media communications, Internet of Things (IoT) etc. Sources are innumerable. The size of data generated may be of the scale of petabytes, exabytes or zetabytes, in just few seconds. The variety of data may be text consisting of alphabets, numbers and dates, images like photographs, biometric data like thumb impression, IRIS scan from a single application like Aadhar. It may be possible that in future they may also incorporate person's voice or DNA sequencing data in Aadhar for health, security or forensics purpose. Graphical data like Cardiograms from health care applications etc. Thus, a single application like Aadhar Identity, for an entire nation, generates enormous amount of data with variety of data sets, each type having the need for accurate creation and secured storage.

To implement all the analysis on this variety of voluminous data, one needs to have a very good resource setup that is computationally cost effective, memory storage efficient, and easy to maintain. Computational processing power based High Performance Computational machines are available but they are cost intensive and difficult to maintain. To avoid overheads, the computational power provided by Distributed Computing Systems should be exploited.

A Distributed Computing System (DCS) is a collection of processors that exist in ordinary office computers and are connected over network. These processors communicate to each other by message passing and are used for the execution of resource intensive applications [4]. Traditionally, DCS involves computing or processing using spatially distributed systems [5] implemented over the homogeneous set of computers. Each of these computer may have similar hardware architecture and operating system, also referred as Homogeneous Distributed Computing System (HDCS). But, practically distributed systems are comprising of machines having different processors, memory and an operating system, each connected via a high-speed network. Such distributed computing system with different type of hardware architecture and the operating systems is known as Heterogeneous Distributed Computing System (HeDCS), also referred as Heterogeneous Distributed System (HeDS).

Large number of software are available to execute Big Data applications and manage Heterogeneous Distributed Systems such as Apache Hadoop, Apache Spark, HTCondor from Wisconsin University, Matlab Parallel and Distributed toolbox etc. But, these software need specific setup to implement. Eg: It is required to convert all algorithms on MapReduce format in Apache Hadoop. Apache Spark has very rich library in python, but many applications that are in use, may not be convertible into Python, thus making it inconvenient to use.

The RMI programming in Java and multi-threading concept of Java are very useful in development and implementation of applications on distributed systems [6].

Java is a platform independent, open source programming language having rich collection of APIs useful in variety of applications. It also has efficient web application development APIs. The multithreading capabilities of Java is competitive. Java RMI is a very efficient API to implement applications that need to be distributed. Since, Java is platform independent, it is very useful in Heterogeneous Distributed Systems. Moreover, if required in the application, Java has the capability to call Native code also. This makes Java programming language and in particular, Java RMI, the very effective technology to implement applications involving Big Data Analysis on Heterogeneous Distributed Systems. The best feature of implementing Heterogeneous Systems using Java RMI is its scalability and transparency. It can also distribute existing applications running in any language with native code, without the need for modifications. Data partitioning and distribution can also be made dynamic based on the available processing nodes at a given instance of time.

The paper discusses in detail, features of Java RMI and how it can be used in implementing distributed applications

II Heterogeneous Distributed Systems

The alternative solution to High Performance Computational Resources is a Distributed Systems Approach. It has emerged as a viable alternative to specialized parallel computing or high performance computing. By harnessing the spare clock cycles of idle machines [7], it is possible to emulate the computing power offered by a specialized parallel machine at a fraction of the cost.

Distributed System is the collection of independent computers that appear to the users of the system as a single computer. [8]

- It is a system in which hardware or software components located at networked computers communicate and co-ordinate their actions only by message passing. [9]
- Modern-day distributed systems are implemented on machines having different processing capacity, varying size of memory ability, having different hardware architectures and varying operating systems, each connected via a high-speed network. Such distributed computing systems with different type of hardware architecture and the operating system is known as Heterogeneous Distributed Computing Systems (HeDCS), also referred as Heterogeneous Distributed Systems (HeDS).

  The type of applications that are generally considered to be suitable for computing over distributed systems have the capability to fully exploit 'coarse-grained parallelism'. It means that there is a possibility to partition the application into independent tasks or processes that can be computed concurrently. Typically these types of problems must display a high 'compute-to-data' ratio to make it worthwhile sending the data over a network rather than computing locally [7]

  It is feasible to implement computing using distributed systems, wherever the use of idle resources over the network is possible. The feasibility [10][11] is due to:
- Individual workstations are becoming increasingly powerful.
- The communications bandwidth between workstations is increasing as new networking technologies and protocols are implemented.
- Individual workstations are easier to integrate into existing networks than special-purpose HPCs or parallel computers.
- The software development tools for workstations are more mature as opposed to proprietary solutions for parallel computers – primarily because of non-standard nature of many parallel systems.
- Individual workstation are inexpensive and readily available alternative to specialised high performance computing platforms.
- Scalability of processing power is easier as nodes can be added easily as well as the processing power of each node can be enhanced

III Java RMI

The Java RMI (Java Remote Method Invocation) is an API that offers a mechanism to develop distributed application in Java. Through RMI, it is possible to invoke methods on an object, running in another JVM. It means that, normally method calls are transfer of control to memory addresses within the shared memory space on a single computer. But using RMI, the method call is taking place to the memory address

residing on a different computer i.e. unshared memory references. This remote communication between the applications using Java RMI, is possible due to two objects stub and skeleton.
The stub is an object that acts as a gateway on the client side. All the outbound requests are transmitted through it. It resides on the client machine and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It establishes a connection with remote JVM.
2. It marshals the parameters sent to the remote JVM.
3. It receives the result
4. It un-marshals the return value or exception, and
5. It finally, returns the result to the requesting client.

The skeleton is an object, acts as a gateway for the server. All the incoming requests to the server, are transmitted through it. When the incoming request is received by the skeleton, it performs as:

1. It reads the parameter for the remote method
2. It un-marshals the parameters
3. It invokes the method on the actual remote object, and
4. It receives the result value
5. It marshals result
6. It sends the marshalled result to the requestor.

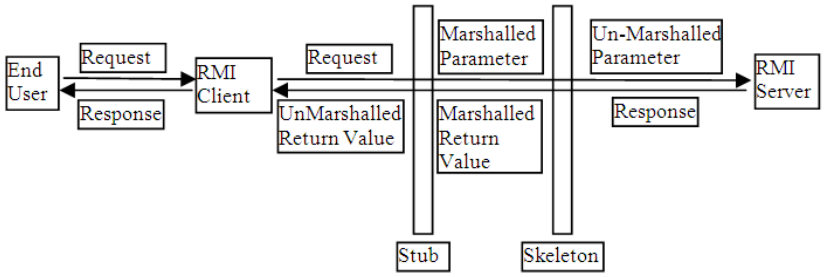

Fig. 1. RMI Communication System

To implement RMI, one needs to write 4 Java Programs. These 4 Java programs may be defined as
1) Interface program
2) Implementation program
3) Server program
4) Client program

-An Interface program contains just the declaration of an Interface
-Interface contains the declaration of method which is executed on remote Server. It extends Remote Interface of java.rmi package.
-Interface program contains the declaration of remote method which is actually running on the server side. Since the remote method's definition, does not exist on the client side, the .class file of Interface should also be there on the client side, with the help of whose object the remote method is invoked.
The Interface should extend (Interface to Interface inheritance is through extends keyword) the Remote interface which is defined in java.rmi package

-An Implementation program contains the class that defines the method declared in the Interface program
- Implementation class has the actual definition of the method which is invoked from Client.
-Server is a program which listens to the requests of the Client.
- Implementation is the class which actually contains the definition of the remote method. This method extends UnicastRemoteObject class which is defined in the java.rmi package. It also implements the Interface which we have created of our program.

-Server listens to Client requests using:
Naming.bind or Naming.rebind("URLReferenceString, ObjectNameofImplClass);

In the Server program we create the object of Implementation class.
This object is bound with the URLReferenceString in the bind/rebind method described above.
bind/ rebind methods throw checked exception of type RMIException. So it is necessary to write it in try-catch block.

-Client is the java program which invokes the remote method.
-Client invokes the remote method with help of :

Naming.lookup("rmi://HostIPAddr:PortNo/URLReferenceString"
URLReferenceString is the string mentioned in Naming.bind(ReferenceString, ObjectNameofImplClass);

This lookup method (is a static method in Naming class of java.rmi package) is typecasted to the Interface Object & assigned to Interface Reference.

With the help of this Interface Reference we invoke remote method whose declaration is given in Interface & hence is known to Client.

As shown in the code below we initialise the serialVersionUID static variable to 0, stating that the while the stub marshals the object, it uses the class definition which is available in the Class object i.e. the definition of class whose object it is marshalling has not been changed after it was created.

```
/*
import java.rmi.*;

import java.rmi.server.*;
public class Implementation extends UnicastRemoteObject implements Interface
{
        static final long serialVersionUID=0;
*/
```

IV Implementation And Deployment Of Java Rmi Code

Once the programs are written as mentioned above,

1) Compile all 4 .java files using command:
> javac *.java

2) Create a stub & skeleton files for the remote method using:
> rmic Implementation

Here the remote method belongs to Implementation class file.

rmic is the RMI Compiler program available in JDK software.
If we use the command as above, it will create 2 files

Implementation_Stub.class
Implementation_Skel.class

If we use the command as:

> rmic -v1.2 Implementation

It will not create skeleton (_Skel) file, which is not needed after jdk 1.2 version.

3) Start the RMI Registry so as the enable the RMI protocol services.
The RMI Registry will run as services or background process in Windows or Linux respectively.
This can be done as:

> start rmiregistry - On Windows
or

> rmiregistry & - On Linux.
On Linux, it will return the pid of the process, as the return status.

4) Once the rmiregistry has been started, start the java Server program which is written by us, as a background program, which will listen for the client requests.
Give command as:

> start java Server - On Windows
or

> java Server &

5) Now open another terminal, & run the Client program to invoke the Server program containing the remote method.
Give
> java Client

In above, give the set of parameters like the HostIPAddr of the machine on which the server is running & any other parameters that the method needs.

By default the RMI Registry listens on port 1099.

If you need to change the port mention it when you run rmiregistry as:
> start rmiregistry XXXX - where XXXX is a new port no.
or

> rmiregistry XXXX &

You need to mention this port no. in Naming.lookup method in its URL as "rmi://HostIPAddr:portno/URLReferenceString"

Deployment:
When you have to deploy the RMI programs on Client & Server,

The Server side should contain the following .class files:
Server
Interface
Implementation
Implementation_Stub,
Implementation_Skel files
Any other .class files which the server may need in the program for defining Server side Business Logic

The Client side should contain:
Client
Interface
Implementation_Stub
&
Any other .class files which the client may need in the program.

V Java Rmi Implementation Frameworks And Case Studies

The Java RMI can be implemented for various applications/cases using the following implementation frameworks:

- An RMI Client may call RMI Server program, which contains the code for business logic.
- To implement application distribution, an RMI client, may call RMI Server programs that are available on several nodes of processor pool of distributed systems. The call to several servers from RMI client may be handled using Java multithreading.
- An RMI Client may call RMI Server program, which in turn may call another POJO that contains the business logic.
- An RMI Client may invoke an RMI Server, which in turn can execute the native commands to find the set of available resources on that node.
- An RMI client may invoke the RMI Server, which in turn can execute any 3rd party code like matlab program. The result of the Matlab program may be stored in some file, which may be then be read by the

Java Server program and returned to the RMI Client which in turn may compile the final result and return to the client.

- The Web Client may request the Web Server for some process intensive application. To enable faster execution, the Web Server may further distribute the application. To implement distribution, the Web Server may call an RMI client, which is also part of the Web Container. The RMI Client in turn may call RMI Server programs that are available on several nodes of processor pool of distributed systems.
- The simplest of all case studies is Matrix Multiplication, where each row-column product can be performed on a separate node, in case of large sized or multi-dimensional matrices.
- An application having many independent functions to be executed, can be implemented using Java RMI. Eg: In Railway Reservation System, function for searching a train and making a reservation on some different train are independent of each other. These two requests can be handled using two different function calls performed from different RMI Servers.
- In case of Big Data, where the data is too large, it can be partitioned into different sets of data. The processing can be done on each partitioned set, on different node. The results from each node will be acquired back, combined into the single result and returned to the end user.

VI Conclusion

In this paper, a contemporary topic of research on Big Data is discussed and how it may be analysed without the need for High Performance Computing machines integration. The Heterogeneous Distributed System is considered as the better alternative to HPC for processing Big Data. The paper discusses Heterogeneous Distributed Systems. Java RMI is the much simpler way of implementing applications over Heterogeneous Distributed Systems. Java RMI achieves platform independence, scalability, transparency and hence suitable approach for implementing applications on Heterogeneous Distributed Systems. Java RMI is exemplified using various case studies. Therefore, future research can focus on providing a roadmap or framework for implementing variety of applications concerning big data analysis and dealing with various challenges in the mentioned applications.

References

[1] Turn Big Data into Big Value, A Practical Strategy, Intel White Paper, 2013.
[2] A. Garg, M. Padole, Big Data: A Stimulus in Business Analytics, IOSR Journal of Computer Engineering (IOSR-JCE), 18(5), 2016, 61-67
[3] M. Schroeck, R. Shockley, J. Smart, D. Romero-Morales, and P. Tufano, Analytics: the real-world use of big data: How innovative enterprises extract value from uncertain data, Executive Report, IBM Institute for Business Value and Said Business School at the University of Oxford, 2012.
[4] Topcuoglu, S. Hariri, and M.Y. Wu., Performance-Effective and Low – Complexity Task Scheduling for Heterogeneous Computing, IEEE Trans. Parallel and Distributed Systems, 13(3), 2002, 260 – 274
[5] M. C. Padole, Distributed Computing for Structured Storage, Retrieval and Processing of DNA Sequencing Data, International Journal of Internet and Web Technology, 38(1), 2013, 1113-1118
[6] M. Padole, Distributed Approach to Pattern Matching in Genomics, Doctoral Dissertation, The Maharaja Sayajirao University of Baroda, India, 2014
[7] M. Baker, R. Buyya, Cluster Computing at a Glance, 1999
[8] A. Tanenbaum & M. Steen, Distributed Systems: Prinicples & Paradigms, (Prentice Hall, NJ)
[9] G. Colouris, et. al., Distributed Systems: Concepts & Design, (Addison Wesley, Pearson Edu. Inc, USA), 2012
[10] L. Turcotte , A Survey of Software Environments for Exploiting Networked Computing Resources, 1993
[11] T. Anderson, D. Culler, and D. Patterson, A Case for NOW(Network of Workstation, IEEE Micro, 15(1), 1995, 54-64